Solutions to Practice Problems 5.2

1. The header of an event procedure has parameters (such as e and sender) that are provided automatically by Visual Basic, and the procedure is invoked when an event is raised. On the other hand, a Sub procedure is invoked 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.

5.3 Sub Procedures, Part II

In the previous section values were passed to Sub procedures. In this section we show how to pass values back from Sub procedures.

Passing by Value

In Section 5.2, 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 terminates—regardless of what changes are made to the value of 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 argument in the calling statement.

```
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 & " from event procedure")
  Triple(amt)
  lstResults.Items.Add(amt & " from event procedure")
End Sub

Sub Triple(ByVal num As Double)
  'Triple a number
  lstResults.Items.Add(num & " from Sub procedure")
  num = 3 * num
  lstResults.Items.Add(num & " from Sub procedure")
End Sub
```

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

```
2 from event procedure
2 from Sub procedure
6 from Sub procedure
2 from event procedure
```

When a variable is passed by value, two memory locations are involved. Figure 5.10 shows the status of the memory locations as the program in Example 1 executes. 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.

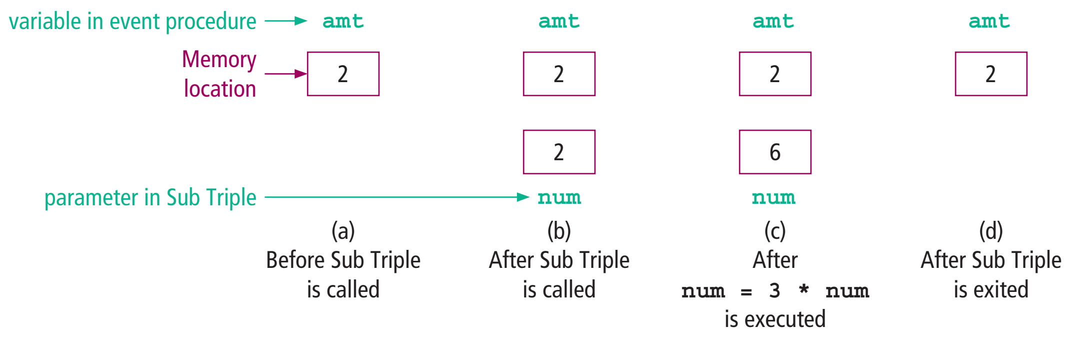

variable in event procedure
amt
Memory location
2
2
2
6
parameter in Sub Triple
num
(a) Before Sub Triple is called
(b) After Sub Triple is called
(c) After num = 3 * num is executed
(d) After Sub Triple is exited

FIGURE 5.10 **Passing a variable by value to a Sub procedure.**

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 keyword ByRef. Suppose a variable, call it *arg*, appears as an argument in a procedure call, and its corresponding parameter in the Sub procedure's header, call it *par*, is preceded by ByRef. After the Sub procedure has been executed, *arg* will have whatever value *par* had in the Sub procedure.

In Example 1, if the header 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* (Fig. 5.11(a)). When the Sub procedure is called, the parameter *num* becomes the Sub procedure's name for this memory location (Figure 5.11(b)). When the value of *num* is tripled, the value in the memory location becomes 6 (Figure 5.11(c)). After the completion of the Sub procedure, the parameter name *num* is forgotten; however, its value lives on in *amt* (Figure 5.11(d)). The variable *amt* is said to be **passed by reference**.

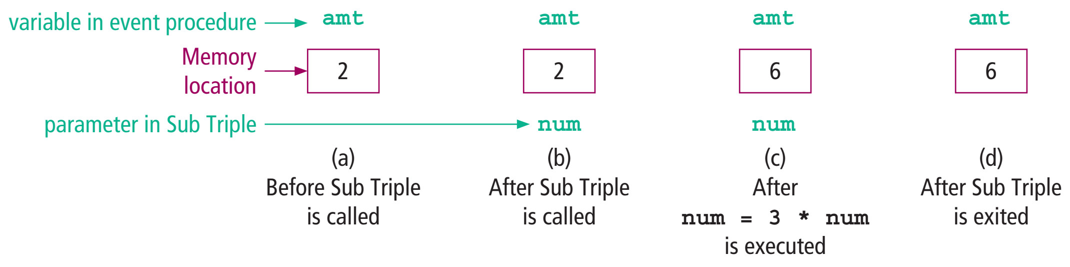
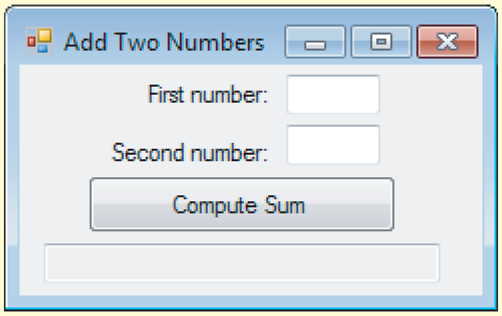

variable in event procedure
amt
Memory location
2
2
6
6
parameter in Sub Triple
num
(a) Before Sub Triple is called
(b) After Sub Triple is called
(c) After num = 3 * num is executed
(d) After Sub Triple is exited

FIGURE 5.11 **Passing a variable by reference to a Sub procedure.**

Note: The Sub procedure Triple discussed above is solely for illustrative purposes and is not representative of the way Sub procedures are used in practice. Examples 2 and 3 show typical uses of Sub procedures.

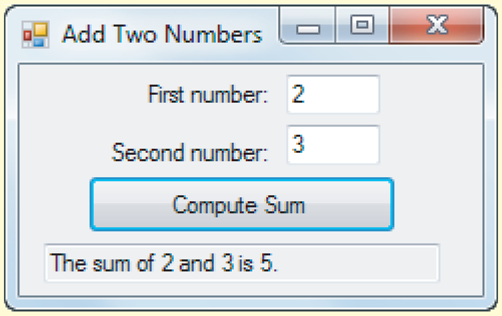
Example 2 The following program uses a Sub procedure to acquire the input. The variables *x* and *y* are not assigned values prior to the execution of the first procedure call. Therefore, before the procedure call is executed, they have the value 0. After the procedure call is executed, however, they have the values entered into the text boxes. These values then are passed by the second procedure call to the Sub procedure DisplaySum.



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

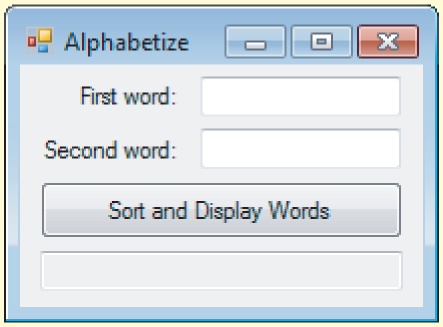
```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'This program requests two numbers and
  'displays the two numbers and their sum.
  Dim x, y As Double
  GetNumbers(x, y)
  DisplaySum(x, y)
End Sub

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 two numbers and their sum
  Dim sum As Double
  sum = num1 + num2
  txtResult.Text = "The sum of " & num1 & " and " &
                   num2 & " is " & sum & "."
End Sub
```

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



First number: 2
Second number: 3
The sum of 2 and 3 is 5.

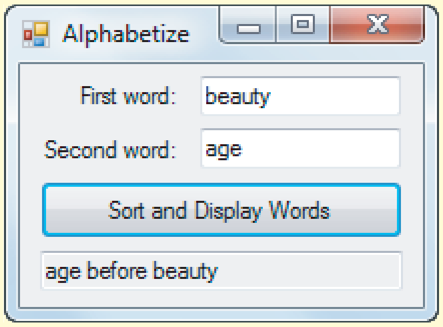
Example 3 The following program alphabetizes two words.



OBJECT	PROPERTY	SETTING
frmWords	Text	Alphabetize
lblFirst	Text	First word:
txtFirst		
lblSecond	Text	Second word:
txtSecond		
btnSort	Text	Sort and Display Words
txtOutput	ReadOnly	True

```
Private Sub btnSort_Click(...) Handles btnSort.Click
  Dim word1 As String = txtFirst.Text
  Dim word2 As String = txtSecond.Text
  If (word2 < word1) Then
    SwapWords(word1, word2)
  End If
  txtOutput.Text = word1 & " before " & word2
End Sub

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

[Run, enter words in the top two text boxes, and click on the button.]



Sub Procedures that Return a Single Value

A Sub procedure having the ByRef keyword in its list of parameters can be thought of as returning values to the calling statement. In Examples 2 and 3, the Sub procedures returned two values to the event procedure. Sub procedures also can be used to return just a single value. However, good programming practice dictates that unless the Sub procedure does more that just return a single value, it should be replaced with a Function procedure. For instance, consider the following procedure call and Sub procedure combination where the Sub procedure returns a single value, namely the sum of two numbers.

```
CalculateSum(x, y, s)

Sub CalculateSum(ByVal num1 As Double, ByVal num2 As Double,
                 ByRef sum As Double)
  'Add the values of num1 and num2
  sum = num1 + num2
End Sub
```

It should be replaced with the combination

```
s = CalculateSum(x, y)

Function CalculateSum(ByVal num1 As Double, ByVal num2 As Double) As Double
  Dim sum As Double
  sum = num1 + num2
  Return sum
End Function
```

Lifetime and Scope of Variables and Constants

When a variable or constant is declared inside a Function, Sub, or event procedure with a Dim statement, a portion of memory is set aside to hold the value of the variable. That portion of memory is released when the procedure's End Function or End Sub statement is reached. The **lifetime** of a variable or constant is the period during which it remains in memory. (A variable's value can change over its lifetime, but it always holds some value.) The **scope** of a variable or constant is the portion of the program that can refer to it. A variable or constant declared in a procedure with a Dim, Const, ByVal, or ByRef keyword is a **local variable** or a **local constant** and is said to have **local scope**. (It cannot be accessed outside the procedure.) When variables or constants declared in two different procedures have the same name, Visual Basic treats them as two different objects.

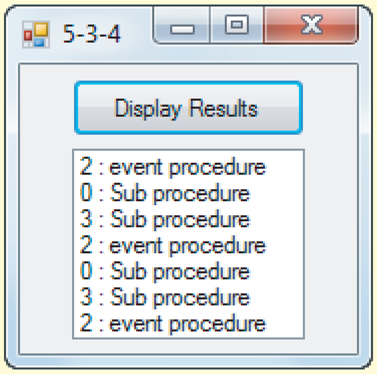
A variable or constant declared outside of a procedure has **class-level scope** and can be referred to by any procedure. A variable or constant declared inside an If or a Select Case block has **block-level scope** and cannot be accessed outside the block. Good programming practice dictates that the scope of a variable or constant be as small as possible. For a variable, this reduces the number of places in which its value can be modified incorrectly or accidentally.

Example 4 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.Add(x & " : event procedure")
  Trivial()
  lstResults.Items.Add(x & " : event procedure")
  Trivial()
  lstResults.Items.Add(x & " : event procedure")
End Sub

Sub Trivial()
  Dim x As Double
  lstResults.Items.Add(x & " : Sub procedure")
```

```
    x = 3
    lstResults.Items.Add(x & " : Sub procedure")
  End Sub
```

[Run, and then click on the button.]



```
2 : event procedure
0 : Sub procedure
3 : Sub procedure
2 : event procedure
0 : Sub procedure
3 : Sub procedure
2 : event procedure
```

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 5" uses the Visual Basic debugger to trace the flow through a program and observe the interplay between arguments and parameters.

VideoNote
Debugging procedures

Comments

1. In this textbook, passing by reference is used primarily to acquire input.
2. When an argument that is a literal or an expression is passed to a procedure, there is no difference between passing it by reference and passing it by value. Only a variable argument can possibly have its value changed by a Sub procedure.

Practice Problems 5.3

1. In Example 3, change the header of the Sub procedure to

   ```
   Sub SwapWords(ByRef word1 As String, ByVal word2 As String)
   ```

   and determine the output when the input is *beauty* and *age*.

2. When the following program in entered, Visual Basic will display a green wavy line under the argument *state* in the fourth line. However, there is no wavy line under the argument *pop*. What do you think the reason is for Visual Basic's concern?

   ```
   Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
     Dim state As String
     Dim pop As Double
     InputData(state, pop)
     txtOutput.Text = state & " has population " & FormatNumber(pop, 0)
   End Sub
   ```

```
Sub InputData(ByRef state As String, ByRef pop As Double)
  state = "California"
  pop = 34888000
End Sub
```

EXERCISES 5.3

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

1.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim name As String = ""
  Dim 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
```

2.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim country As String = ""
  Dim pop As Double
  GetFacts(country, pop)
  txtOutput.Text = "The population of " & country & " is about " &
                   FormatNumber(pop, 0) & "."
End Sub

Sub GetFacts(ByRef country As String, ByRef pop As Double)
  country = "the United States"
  pop = 313000000    'population
End Sub
```

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim state As String = ""
  Dim flower As String = ""
  GetFacts(state, flower)
  txtOutput.Text = "The state flower of " & state &
                   " is the " & flower & "."
End Sub

Sub GetFacts(ByRef place As String, ByRef plant As String)
  place = "Alaska"
  plant = "Forget Me Not"
End Sub
```

4.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim film As String = ""
  Dim year As Integer
  GetFacts(film, year)
  txtOutput.Text = film & " won the award in " & year & "."
End Sub
```

```
Sub GetFacts(ByRef movie As String, ByRef yr As Integer)
  movie = "Slumdog Millionaire"
  yr = 2009
End Sub
```

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String = ""
  Dim num As Integer
  GetFacts(word, num)
  txtOutput.Text = "The first " & num & " letters of " & word &
                   " are " & BegOfWord(word, num) & "."
End Sub

Sub GetFacts(ByRef w As String, ByRef n As Integer)
  w = InputBox("Enter a word:")
  n = CInt(InputBox("Enter a number less than the length of the word:"))
End Sub

Function BegOfWord(ByVal word As String, ByVal num As Integer) As String
  Return word.Substring(0, num)
End Function
```

(Assume the two responses are *EDUCATION* and *3*.)

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim price, markdown, salesTax, finalCost As Double
  InputData(price, markdown, salesTax)
  finalCost = CostOfItem(price, markdown, salesTax)
  DisplayOutput(price, finalCost)
End Sub

Sub InputData(ByRef price As Double, ByRef markdown As Double,
              ByRef salesTax As Double)
  price = CDbl(InputBox("Price of item:"))
  markdown = CDbl(InputBox("Percentage discount:"))
  salesTax = CDbl(InputBox("Percentage state sales tax:"))
End Sub

Function CostOfItem(ByVal pr As Double, ByVal md As Double,
                    ByVal st As Double) As Double
  Dim reducedPrice, cost As Double
  reducedPrice = pr - ((md / 100) * pr)
  cost = reducedPrice + ((st / 100) * reducedPrice)
  Return cost
End Function

Sub DisplayOutput(ByVal amount, ByVal customerCost)
  lstOutput.Items.Add("Original Price: " & FormatCurrency(amount))
  lstOutput.Items.Add("Cost: " & FormatCurrency(customerCost))
End Sub
```

(Assume the three responses are *125*, *20*, and *6*.)

7.
```
Dim inventory As Integer = 5

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim numPurchased, newInventory As Integer
  numPurchased = CInt(InputBox("Enter number of items to be purchased:"))
  UpdateInventory(newInventory, numPurchased)
  inventory = newInventory
  txtOutput.Text = "Current inventory: " & inventory
End Sub

Sub UpdateInventory(ByRef newInventory As Integer,
                    ByVal numPurchased As Integer)
  Select Case numPurchased
    Case Is <= inventory
      newInventory = inventory - numPurchased
      If newInventory = 0 Then
        MessageBox.Show("No items remaining.")
      End If
    Case Is > inventory
      MessageBox.Show("Insufficient inventory, purchase cancelled.")
      newInventory = inventory
  End Select
End Sub
```

(Assume btnDisplay is pressed twice with the response 3 given each time.)

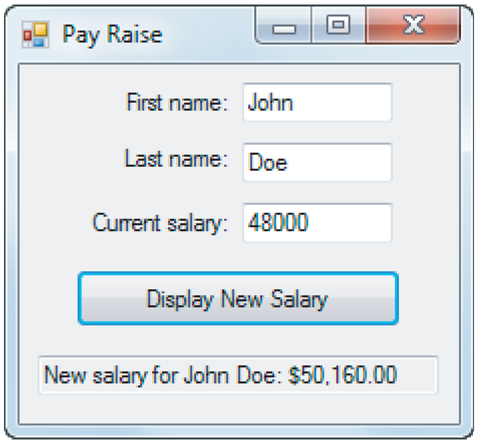
8.
```
Dim balance As Double = 100

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim deposit, withdrawal, newBalance As Double
  deposit = CDbl(InputBox("Amount of deposit:"))
  withdrawal = CDbl(InputBox("Amount of withdrawal:"))
  UpdateBalance(deposit, withdrawal, newBalance)
  balance = newBalance
  txtOutput.Text = CStr(balance)
End Sub

Sub UpdateBalance(ByVal deposit As Double, ByVal withdrawal As Double,
                  ByRef newBalance As Double)
  Select Case withdrawal
    Case Is = balance + deposit
      MessageBox.Show("Account depleted.")
      newBalance = 0
    Case Is > balance + deposit
      MessageBox.Show("Account overdrawn. Withdrawal denied.")
      newBalance = balance + deposit
    Case Else
      newBalance = balance + deposit - withdrawal
  End Select
End Sub
```

(Assume the responses are 90 and 200.)

9.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a, b, s, d As Integer
  InputData(a, b)
  Combine(a, b, s, d)
  DisplayResults(s, d)
End Sub

Sub InputData(ByRef num1 As Integer, ByRef num2 As Integer)
  num1 = 3
  num2 = 1
End Sub

Sub Combine(ByVal x As Integer, ByVal y As Integer,
            ByRef sum As Integer, ByRef difference As Integer)
  sum = x + y
  difference = x - y
End Sub

Sub DisplayResults(ByVal s As Integer, ByVal d As Integer)
  lstOutput.Items.Add("sum = " & s)
  lstOutput.Items.Add("difference = " & d)
End Sub
```

10.
```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim wholesaleCost, salePrice, percentCommission,
      salesTax, profit As Double
  InputData(wholesaleCost, salePrice, percentCommission)
  CalculateSomeValues(wholesaleCost, salePrice, percentCommission,
                      salesTax, profit)
  DisplayData(salesTax, profit)
End Sub

Sub InputData(ByRef wholesaleCost As Double, ByRef salePrice As Double,
              ByRef percentCommission As Double)
  wholesaleCost = 100
  salePrice = 300
  percentCommission = 5
End Sub

Sub CalculateSomeValues(ByVal wholesaleCost As Double,
          ByVal salePrice As Double, ByVal percentCommission As Double,
          ByRef salesTax As Double, ByRef profit As Double)
  salesTax = 0.06 * salePrice
  profit = salePrice - wholesaleCost -
           salePrice * (percentCommission / 100)
End Sub

Sub DisplayData(ByVal salesTax As Double, ByVal profit As Double)
  lstOutput.Items.Add("sales tax: " & FormatCurrency(salesTax))
  lstOutput.Items.Add("profit: " & FormatCurrency(profit))
End Sub
```

11. Write a pay-raise program that requests a person's first name, last name, and current annual salary, and then displays their salary for next year. People earning less than $40,000 will receive a 5% raise, and those earning $40,000 or more will receive a raise of $2,000 plus 2% of the amount over $40,000. Use Sub procedures for input and output, and a Function procedure to calculate the new salary. See Fig. 5.12.


Pay Raise

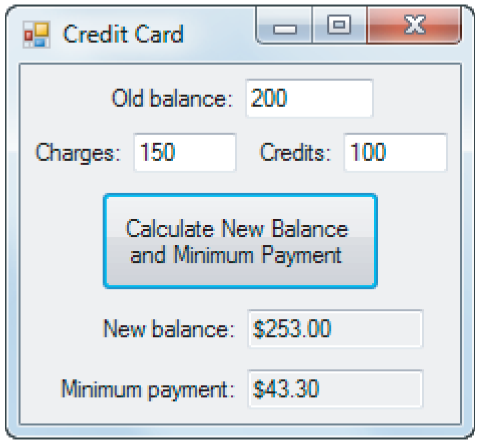
First name: John

Last name: Doe

Current salary: 48000

New salary for John Doe: $50,160.00

FIGURE 5.12 Possible output for Exercise 11.


Credit Card

Old balance: 200

Charges: 150 Credits: 100

New balance: $253.00

Minimum payment: $43.30

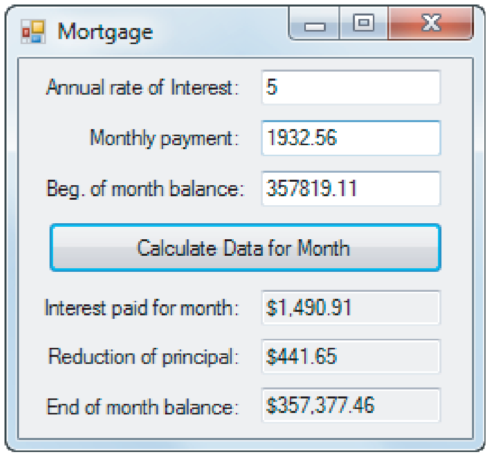
FIGURE 5.13 Possible output for Exercise 12.

12. Write a program to calculate the balance and minimum payment for a credit card statement. See Fig. 5.13. The program should use the event procedure shown in Fig. 5.14. The finance charge is 1.5% of the old balance. If the new balance is $20 or less, the minimum payment should be the entire new balance. Otherwise, the minimum payment should be $20 plus 10% of the amount of the new balance above $20.

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim oldBalance, charges, credits, newBalance, minPayment As Double
  InputData(oldBalance, charges, credits)
  CalculateNewValues(oldBalance, charges, credits, newBalance, minPayment)
  DisplayData(newBalance, minPayment)
End Sub
```

FIGURE 5.14 Event procedure for Exercise 12.

13. Write a program to calculate the monthly values associated with a mortgage. See Fig. 5.15. The program should use the event procedure shown in Fig. 5.16. The interest paid each


Mortgage

Annual rate of Interest: 5

Monthly payment: 1932.56

Beg. of month balance: 357819.11

Interest paid for month: $1,490.91

Reduction of principal: $441.65

End of month balance: $357,377.46

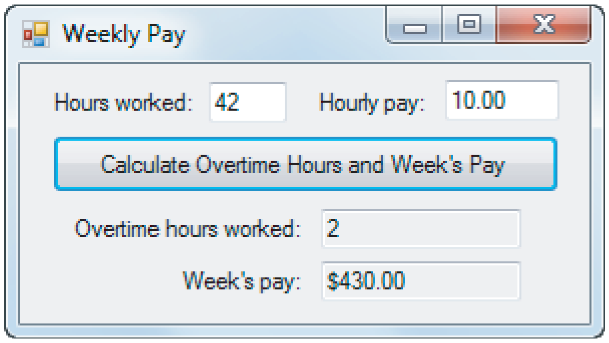
FIGURE 5.15 Sample output for Exercise 13.

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim annualRateOfInterest, monthlyPayment, begBalance As Double
  Dim intForMonth, redOfPrincipal, endBalance As Double
  InputData(annualRateOfInterest, monthlyPayment, begBalance)
  Calculate(annualRateOfInterest, monthlyPayment, begBalance,
            intForMonth, redOfPrincipal, endBalance)
  DisplayData(intForMonth, redOfPrincipal, endBalance)
End Sub
```

FIGURE 5.16 Event procedure for Exercise 13.

month is the monthly rate of interest applied to the balance at the beginning of the month. Each month the reduction of principal equals the monthly payment minus the interest paid. At any time, the balance of the mortgage is the amount still owed; that is, the amount required to pay off the mortgage. The end of month balance is calculated as [beginning of month balance] − [reduction of principal].

14. Write a program to determine a person's weekly pay, where they receive time-and-a-half for overtime work beyond forty hours. See Fig. 5.17. The program ~~should~~ use the event procedure shown in Fig. 5.18.


Weekly Pay

Hours worked: 42 Hourly pay: 10.00

Calculate Overtime Hours and Week's Pay

Overtime hours worked: 2

Week's pay: $430.00

FIGURE 5.17 Sample output for Exercise 14.

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim hours, payPerHour, overtimeHours, pay As Double
  InputData(hours, payPerHour)
  CalculateValues(hours, payPerHour, overtimeHours, pay)
  DisplayData(overtimeHours, pay)
End Sub
```

FIGURE 5.18 Event procedure for Exercise 14.

Solutions to Practice Problems 5.3

1. `age before age`
2. Since *state* is a string variable, its default value is the keyword *Nothing*. The assignment of *Nothing* to an argument makes Visual Basic nervous. Therefore, to keep Visual Basic happy, we will assign the empty string to String variables that are passed to procedures. That is, we will change the first line inside the event procedure to `Dim state As String = ""`.

   Since the default value of the numeric variable *pop* is 0, Visual Basic has no issue with passing that value to a Sub procedure.