We Share the Same Subscripts

At times, you may want to use an array to store items that are related but have different data types. For example, you may want to store the titles of your favorite books (which are strings) and their associated prices (which are numbers) in an array. But how can you store the book information in an array when all of the data in an array must have the same data type? One solution is to use two one-dimensional arrays: a String array to store the titles and a Decimal array to store the prices. Both arrays are illustrated in Figure 19-1.

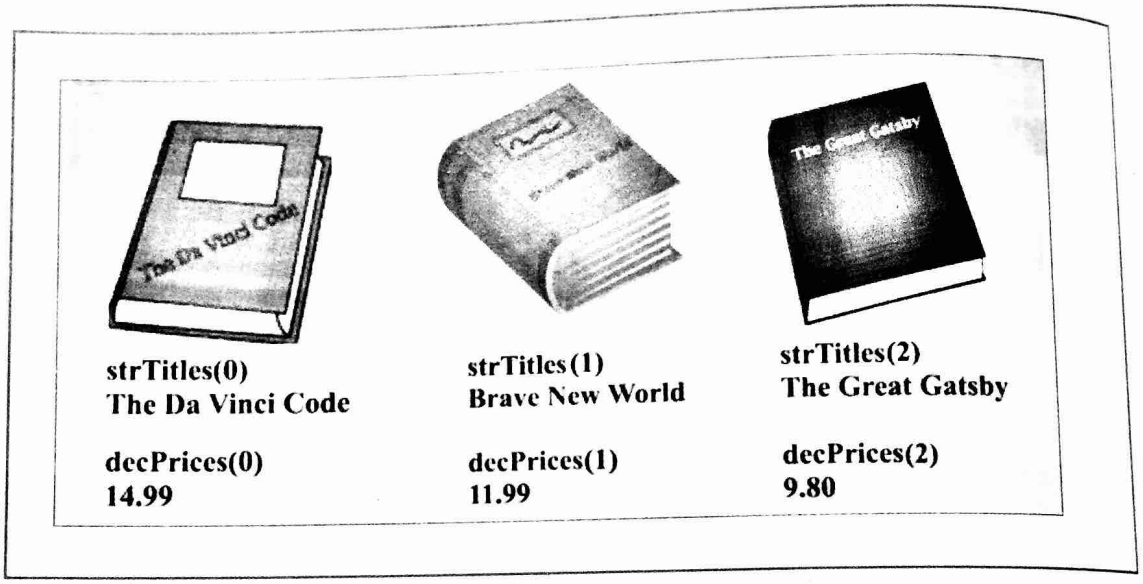

Figure 19-1 Illustration of two parallel one-dimensional arrays
Image by Diane Zak; Created with Reallusion CrazyTalk Animator

The arrays in Figure 19-1 are referred to as **parallel arrays**, which are two or more arrays whose elements are related by their positions in the arrays; in other words, they are related by their subscripts. The arrays are parallel because each element in the `strTitles` array corresponds to the element located in the same position in the `decPrices` array. For example, the first book title (The Da Vinci Code) is stored in the first element in the `strTitles` array, and its price (14.99) is stored in the first element in the `decPrices` array. Likewise, the second book title is stored in the `strTitles(1)` element and its price is stored in the `decPrices(1)` element. The same relationship is true for the third book title and its associated price. To determine a book's price, you locate its title in the `strTitles` array and then view the corresponding element in the `decPrices` array.

West Coast Emporium Application

You will use two parallel one-dimensional arrays to code the West Coast Emporium application. The arrays are illustrated in Figure 19-2. The `strStates` array contains the state IDs for the three states in which West Coast Emporium has stores, and the `intStores` array contains the number of stores in each state.

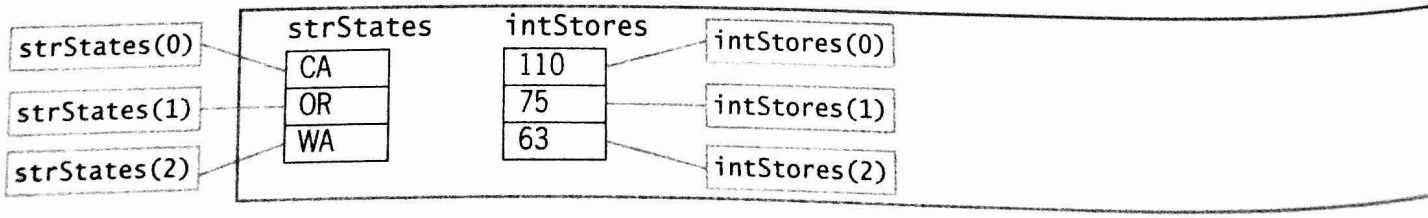

Figure 19-2 Illustration of the `strStates` and `intStores` parallel arrays

The application's interface will provide a text box for the user to enter the state ID. The application will search for the ID in the `strStates` array, beginning with the first element in the array. If it finds the ID, it will display the corresponding number of stores from the `intStores` array; otherwise, it will display the number 0. Figure 19-3 shows the planning information for the application. The algorithm contains a loop whose instructions will be repeated either until the state ID is located in the array or until the array subscript equals the number of array elements, which indicates that there are no more elements to search.

```
Output:      number of stores or 0

Processing:  three-element one-dimensional state IDs array (class-level)
             three-element one-dimensional number of stores array (class-level)
             array subscript
             search results variable

Input:       state ID to search for

Algorithm:
1. enter the state ID to search for; trim any leading and/or trailing spaces and then
   convert to uppercase
2. ensure that the search begins with the first element in the state IDs array by starting
   the array subscript at 0
3. assume that the state ID is not in the state IDs array by starting the search results
   variable at "N"
4. repeat until the search results variable contains "Y" or the array subscript equals
   the number of elements in the state IDs array:
      if the contents of the current element in the state IDs array
      match the state ID to search for, do this:
            assign "Y" to the search results variable
      otherwise, do this:
            add 1 to the array subscript
      end if
   end repeat
5. if the search results variable contains "Y", do this:
      display the corresponding number of stores from the number of stores array
   otherwise, do this:
      display 0
   end if
```

Figure 19-3 Planning information for the West Coast Emporium application

To begin coding the West Coast Emporium application:

1. Start Visual Studio 2012. Open the **West Coast Solution (West Coast Solution.sln)** file contained in the ClearlyVB2012\Chap19\West Coast Solution folder. If necessary, open the designer window.
2. Open the Code Editor window. First, you will declare the two parallel arrays. Click the **blank line** below the `' class-level parallel arrays` comment and then enter the following declaration statements:

```
Private strStates() As String = {"CA", "OR", "WA"}
Private intStores() As Integer = {110, 75, 63}
```

3. Next, you will code the btnNumberOfStores_Click procedure. The procedure will use three variables. The strSearchFor variable will store the ID to search for in the strStates array. The intSub variable will keep track of the array subscripts during the search. The strFound variable will keep track of whether the ID was found in the strStates array. Locate the btnNumberOfStores_Click procedure. Click the **blank line** below the `' declare variables` comment and then enter the following three declaration statements:

   Dim strSearchFor As String
   Dim intSub As Integer
   Dim strFound As String

4. The first step in the algorithm is to enter the state ID to search for in the strStates array. The user enters the ID in the txtState control. You will trim any leading and/or trailing spaces from the contents of the control and then convert the contents to uppercase, assigning the result to the strSearchFor variable. Click the **blank line** below the `' assign the ID to a variable` comment and then enter the following assignment statement:

   strSearchFor = txtState.Text.Trim.ToUpper

5. The second step in the algorithm starts the array subscript at 0 to ensure that the search begins with the first array element. In the btnNumberOfStores_Click procedure, the intSub variable keeps track of the array subscripts and is already initialized to 0 in its Dim statement. However, for clarity, you will include an assignment statement that assigns the number 0 to the variable. (The procedure will still work correctly without this assignment statement.) Click the **blank line** immediately below the `' found or the end of the array` comment and then enter the following assignment statement:

   intSub = 0

6. Before the search begins, the procedure will assume that the state ID is not contained in the strStates array. According to the third step in the algorithm, this is accomplished by starting the search results variable (strFound) at "N". Enter the following assignment statement:

   strFound = "N"

The fourth step in the algorithm is a loop that repeats its instructions until one of two conditions is true: either the state ID has been found in the strStates array or the subscript equals the number of array elements (which indicates there are no more elements to search). You will use the Do...Loop statement to code this loop. The For...Next statement is not appropriate in this case because you don't know the exact number of times the loop instructions should be repeated.

To finish coding the West Coast Emporium application:

1. Enter the following Do clause:

   Do Until strFound = "Y" OrElse intSub = strStates.Length

2. The selection structure within the loop compares the contents of the current element in the strStates array with the state ID stored in the strSearchFor variable. If both IDs match, the selection structure's true path will assign the string "Y" to the strFound variable to indicate that the ID was located in the array. If both IDs do not match, the selection structure's false path will increment the array subscript by 1 so that the loop can search the next element in the strStates array. Enter the following selection structure:

```
    If strStates(intSub) = strSearchFor Then
        strFound = "Y"
    Else
        intSub = intSub + 1
    End If
```

3. The value stored in the `strFound` variable indicates whether the ID was located in the `strStates` array, and it determines the appropriate information to display. Click the **blank line** below the `' and display the appropriate output` comment and then enter the following selection structure:

```
If strFound = "Y" Then
    lblStores.Text = intStores(intSub)
Else
    lblStores.Text = "0"
End If
```

Figure 19-4 shows the code entered in the form's Declarations section and btnNumberOfStores_Click procedure.

```
' class-level parallel arrays                                  form's Declarations
Private strStates() As String = {"CA", "OR", "WA"}             section
Private intStores() As Integer = {110, 75, 63}

Private Sub btnNumberOfStores_Click(sender As Object,
e As EventArgs) Handles btnNumberOfStores.Click
    ' searches an array for a state ID and then
    ' displays either the number of stores
    ' from another array or 0

    ' declare variables
    Dim strSearchFor As String
    Dim intSub As Integer
    Dim strFound As String

    ' assign the ID to a variable
    strSearchFor = txtState.Text.Trim.ToUpper

    ' search the strStates array for the ID
    ' continue searching until the ID is
    ' found or the end of the array
    intSub = 0
    strFound = "N"
    Do Until strFound = "Y" OrElse intSub = strStates.Length
        If strStates(intSub) = strSearchFor Then
            strFound = "Y"
        Else
            intSub = intSub + 1                                you can also use
        End If                                                 intSub += 1
    Loop

    ' determine whether the ID was found
    ' and display the appropriate output
    If strFound = "Y" Then
        lblStores.Text = intStores(intSub)
    Else
        lblStores.Text = "0"
    End If
End Sub
```

Figure 19-4 Form's Declarations section and btnNumberOfStores_Click procedure

To test the btnNumberOfStores_Click procedure:

1. Save the solution and then start the application. First, you will enter a valid state ID. Type **or** in the State ID box. The state ID appears in the `strStates(1)` element, and its corresponding number of stores (75) appears in the `intStores(1)` element. Click the **Number of Stores** button. The correct number of stores appears in the Stores box, as shown in Figure 19-5. (Recall that you can use the Alt key to show/hide the access keys.)

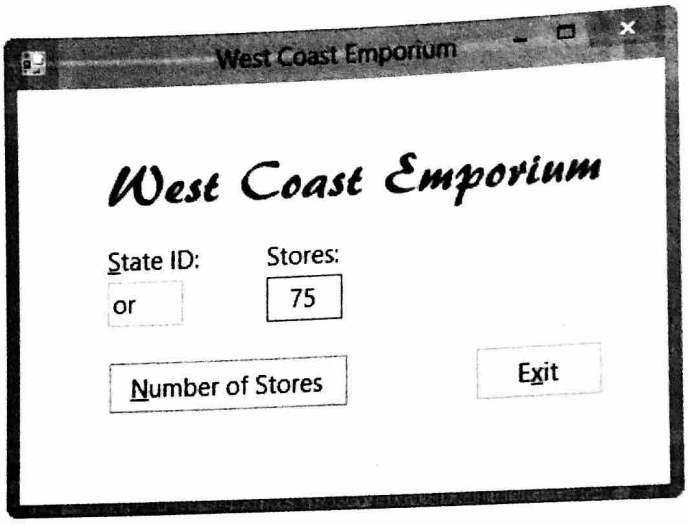

Figure 19-5 Number of stores displayed in the interface

2. Now you will enter an invalid state ID. Change the state ID to **tx** and then click the **Number of Stores** button. The number 0 appears in the Stores box.
3. Test the application several more times using valid and invalid state IDs. When you are finished testing the application, click the **Exit** button. Close the Code Editor window and then close the solution.

Will You Share That with Me?

In the applications you have coded so far, the simple (or scalar) variables were declared in procedures. This is because the variables were needed only by the procedure that declared them. At times, however, two or more procedures in an application may need to use the same variable. In those cases, you can declare the variable as a class-level variable. As you do with class-level arrays, you declare **class-level variables** in the form's Declarations section, using the `Private` keyword. Class-level variables have class scope, which means they are recognized by every procedure contained in the form's Code Editor window. Class-level variables retain their values and remain in the computer's internal memory until the application ends.

You will use a class-level variable and also a class-level array in the Test Scores application, which you code in the next set of steps. Figures 19-6 and 19-7 show the application's interface and planning information, respectively. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.)

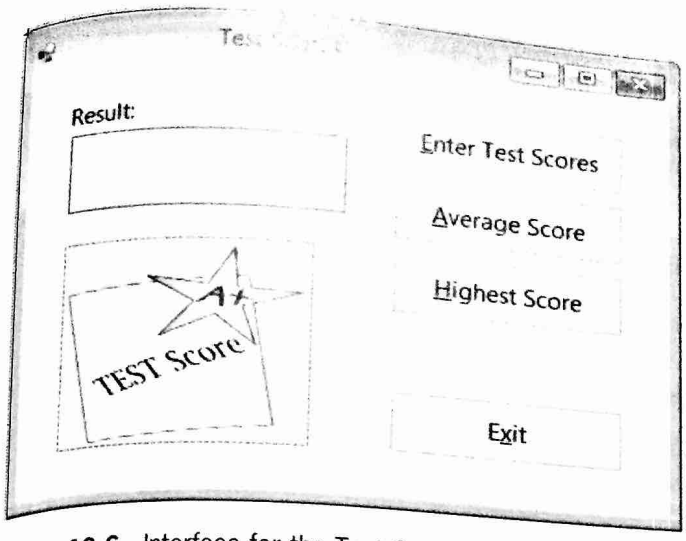

Figure 19-6 Interface for the Test Scores application

OpenClipArt.org/mazeo

```
Output:      average score or highest score

Processing:  five-element one-dimensional scores array (class-level)
             highest subscript (class-level)
             array subscript
             total scores accumulator (start at 0)

Input:       five scores

Algorithm for the Enter Test Scores button:
1. repeat for array subscripts from 0 through the highest subscript (in increments of 1):
       get a score from the user
       store the score in the current array element
   end repeat
2. remove the contents of the Result box

Algorithm for the Average Score button:
1. assign 0 to the total scores accumulator
2. repeat for array subscripts from 0 through the highest subscript (in increments of 1):
       add the contents of the current array element to the total scores accumulator
   end repeat
3. calculate the average score by dividing the total scores accumulator by the number
   of array elements
4. display the average score
```

Figure 19-7 Planning information for the Test Scores application *(continues)*

(continued)

```
Algorithm for the Highest Score button:
1. assign the contents of the first array element as the highest score
2. repeat for array subscripts from 1 through the highest subscript (in increments of 1):
       if the value stored in the current array element is greater than the highest
       score, do this:
              assign the contents of the current array element as the highest score
       end if
   end repeat
3. display the highest score
```

Figure 19-7 Planning information for the Test Scores application

Each algorithm in Figure 19-7 uses a one-dimensional array of scores. The algorithm for the Enter Test Scores button fills the array with values. The algorithms for the Average Score and Highest Score buttons use the array values to determine the average score and the highest score, respectively. For the array to be accessible by each button's Click event procedure, it will need to be declared as a class-level array. Rather than having each button's Click event procedure determine the highest subscript in the array, you will assign the highest subscript to a class-level variable that each procedure can use. As you learned in Chapter 18, you can determine the highest subscript in a one-dimensional array by subtracting 1 from the number stored in the array's Length property.

To begin coding the Test Scores application:

1. Open the **Test Scores Solution** (**Test Scores Solution.sln**) file contained in the ClearlyVB2012\Chap19\Test Scores Solution folder. If necessary, open the designer window.
2. Open the Code Editor window. First, you will declare the class-level array and variable. Click the **blank line** below the `' class-level array and variable` comment, and then enter the following declaration statements:

   Private decScores(4) As Decimal
   Private intHighSub As Integer = decScores.Length – 1

The first algorithm you will code is for the Enter Test Scores button. The algorithm prompts the user to enter five test scores, and it stores each score in an element in the `decScores` array.

To code the Enter Test Scores button's Click event procedure:

1. Locate the btnEnter_Click procedure. The procedure will use the InputBox function to prompt the user to enter a test score. You will need to declare a String variable to store the user's response. Click the **blank line** above the End Sub clause. Type the following declaration statement and then press **Enter** twice:

   Dim strScore As String

2. The first step in the algorithm is a loop that repeats its instructions for array subscripts from 0 through the highest subscript. You can use either the Do...Loop statement or the For...Next statement to code the loop. In this case, because you know the exact number of times the loop instructions should be processed, you will use the For...Next statement. Enter the following For clause:

   For intSub As Integer = 0 to intHighSub

3. Change the Next clause to **Next intSub**.
4. The first instruction in the loop gets a score from the user. Click the **blank line** below the For clause and then enter the following assignment statement:

   strScore = InputBox("Score:", "Test Score")
5. The second instruction in the loop stores the score in the current array element. To accomplish this, you will need to use the TryParse method to convert the score to the Decimal data type. Type the following statement and then click **any other line** in the procedure:

   Decimal.TryParse(strScore, decScores(intSub))
6. The last step in the algorithm removes the contents of the Result box. Insert two blank lines below the `Next intSub` clause. Type the following assignment statement and then click **any other line** in the procedure:

   lblResult.Text = String.Empty

Next, you will code the Average Score button's algorithm. The algorithm accumulates the values stored in the `decScores` array. It then calculates the average value and displays it in the Result box in the interface.

To code the Average Score button's Click event procedure:

1. Locate the btnAverage_Click procedure. The procedure will use two Decimal variables: an accumulator variable to total the scores and a variable to store the average score. Click the **blank line** above the End Sub clause and then enter the following declaration statements and comment. Press **Enter** twice after typing the second declaration statement.

   ```
   Dim decTotal As Decimal     ' accumulator
   Dim decAvg As Decimal
   ```
2. The first step in the algorithm assigns the number 0 to the total scores accumulator, which is named `decTotal`. The `decTotal` variable is already initialized to 0 in its Dim statement. However, for clarity, you will include an assignment statement that assigns the number 0 to the variable. (The procedure will still work correctly without this assignment statement.) Enter the following comment and assignment statement:

   ```
   ' start accumulator at 0
   decTotal = 0
   ```
3. The second step in the algorithm is a loop that repeats its instructions for array subscripts from 0 through the highest subscript. Here, too, you will code the loop using the For...Next statement because you know the exact number of times the loop instructions should be processed. Enter the following comment and For clause:

   ```
   ' accumulate the scores
   For intSub As Integer = 0 To intHighSub
   ```
4. Change the Next clause to **Next intSub**.
5. The instruction in the loop should add the contents of the current array element to the total scores accumulator. You can accomplish this task using either the statement `decTotal = decTotal + decScores(intSub)` or the statement `decTotal += decScores(intSub);` both statements are equivalent. Click the **blank line** below the For clause. Type the following statement and then click **any other line** in the procedure:

   decTotal += decScores(intSub)

6. The third step in the algorithm calculates the average score by dividing the accumulator's value by the number of array elements. Insert two blank lines below the `Next intSub` clause and then enter the following comment and assignment statement:

```
' calculate and display the average score
decAvg = decTotal / decScores.Length
```

7. The last step in the algorithm displays the average score. Type the following assignment statement and then click **any other line** in the procedure:

```
lblResult.Text = "Average:  " & decAvg.ToString("N1")
```

Finally, you will code the Highest Score button's algorithm. The algorithm finds the highest value stored in the `decScores` array and then displays the value in the Result box in the interface.

To code the Highest Score button's Click event procedure:

1. Locate the btnHighest_Click procedure. The procedure will use a Decimal variable to keep track of the highest score. Click the **blank line** above the End Sub clause. Type the following declaration statement and then press **Enter** twice:

```
Dim decHighest As Decimal
```

2. When searching an array for the highest (or lowest) value, many programmers begin by assigning the contents of the first array element to the variable that keeps track of the highest (or lowest) value. Enter the following comment and assignment statement:

```
' determine highest score
decHighest = decScores(0)
```

3. Programmers then use a loop to look at the second through the last element in the array. As indicated in the algorithm shown earlier in Figure 19-7, the loop contains a selection structure that compares the value in each of those elements to the value stored in the `decHighest` variable. (The selection structure doesn't need to compare the first element's value because that value is already assigned to the variable.) If the value stored in the current array element is greater than the value stored in the `decHighest` variable, the selection structure's true path assigns the array element's value to the variable. Enter the following loop and selection structure:

```
For intSub As Integer = 1 To intHighSub
    If decScores(intSub) > decHighest Then
        decHighest = decScores(intSub)
    End If
Next intSub
```

4. The last step in the algorithm displays the highest score in the Result box. Insert two blank lines below the `Next intSub` clause. Type the following comment and assignment statement and then click **any other line** in the procedure:

```
' display the highest score
lblResult.Text = "Highest:  " & decHighest.ToString("N1")
```

Figure 19-8 shows the code entered in the form's Declarations section and also in the btnEnter_Click, btnAverage_Click, and btnHighest_Click procedures.

```
' class-level array and variable                    form's Declarations section
Private decScores(4) As Decimal
Private intHighSub As Integer = decScores.Length - 1

Private Sub btnEnter_Click(sender As Object,        btnEnter_Click procedure
e As EventArgs) Handles btnEnter.Click
    ' gets test scores and stores them in
    ' the class-level array

    Dim strScore As String

    For intSub As Integer = 0 To intHighSub
        strScore = InputBox("Score:", "Test Score")
        Decimal.TryParse(strScore, decScores(intSub))
    Next intSub

    lblResult.Text = String.Empty
End Sub

Private Sub btnAverage_Click(sender As Object,      btnAverage_Click procedure
e As EventArgs) Handles btnAverage.Click
    ' calculates and displays the average test score

    Dim decTotal As Decimal      ' accumulator
    Dim decAvg As Decimal

    ' start accumulator at 0
    decTotal = 0
    ' accumulate the scores
    For intSub As Integer = 0 To intHighSub
        decTotal += decScores(intSub)
    Next intSub

    ' calculate and display the average score
    decAvg = decTotal / decScores.Length
    lblResult.Text = "Average:  " & decAvg.ToString("N1")
End Sub

Private Sub btnHighest_Click(sender As Object,      btnHighest_Click procedure
e As EventArgs) Handles btnHighest.Click
    ' calculates and displays the highest test score

   Dim decHighest As Decimal

   ' determine highest score
   decHighest = decScores(0)
   For intSub As Integer = 1 To intHighSub
       If decScores(intSub) > decHighest Then
           decHighest = decScores(intSub)
       End If
   Next intSub

   ' display the highest score
   lblResult.Text = "Highest:  " & decHighest.ToString("N1")
End Sub
```

Figure 19-8 Most of the Test Scores application's code

To test the Test Scores application:

1. Save the solution and then start the application. First, you will enter five test scores. Click the **Enter Test Scores** button. Type the following five test scores, pressing **Enter** after typing each one: **79.5**, **75**, **90**, **63.5**, and **72**. The button's Click event procedure stores the scores in the decScores array.
2. Click the **Average Score** button to display the average score (76.0) in the Result box. See Figure 19-9.

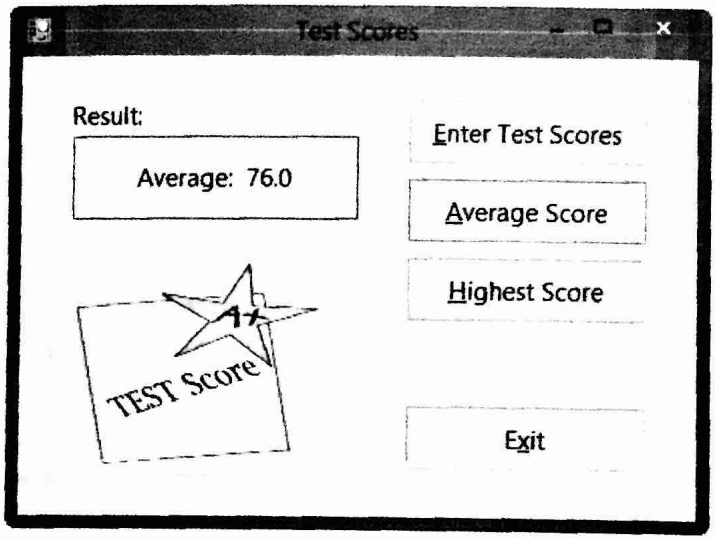

Figure 19-9 Average test score displayed in Result box
OpenClipArt.org/mazeo

3. Click the **Highest Score** button to display the highest score (90.0) in the Result box. See Figure 19-10.

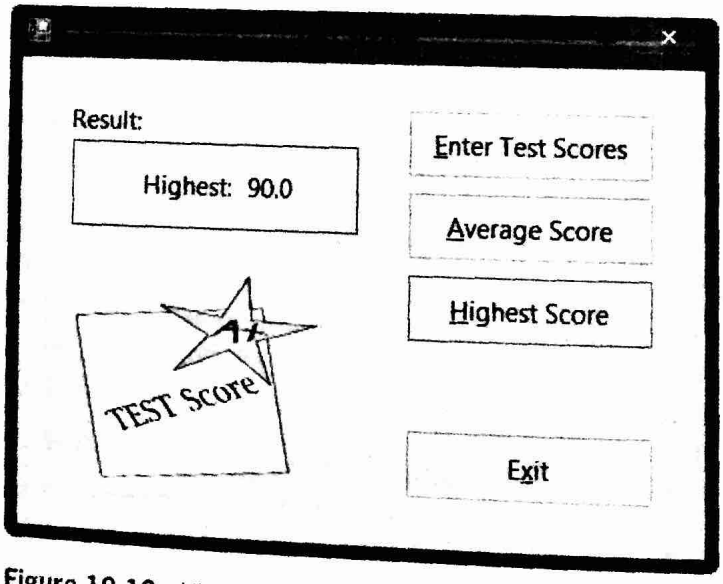

Figure 19-10 Highest test score displayed in Result box
OpenClipArt.org/mazeo

To learn more about class-level memory locations, see the Class-Level Memory Locations section in the Ch19WantMore.pdf file.

4. Click the **Exit** button. Close the Code Editor window and then close the solution.

Mini-Quiz 19-1

See Appendix B for the answers.

1. The elements in parallel arrays are related by their subscripts and data type.
   a. True
   b. False
2. The `strState` and `strCapital` arrays are parallel arrays. If Illinois is stored in the second element in the `strState` array, where is Springfield stored?
   a. `strCapital(1)`
   b. `strCapital(2)`
3. Class-level variables are declared using the `Private` keyword.
   a. True
   b. False

But I Don't Know How Many There Are

At times, you may not know the precise number of array elements needed to store an application's data. In those cases, you can use the ReDim statement to change the number of elements while the application is running. The **ReDim statement** allows you to make an array either larger or smaller. However, in most cases you will use the statement to increase the size of an array.

Figure 19-11 shows the ReDim statement's syntax and includes examples of using the statement. The optional `Preserve` keyword in the syntax tells the computer to keep the current array values when the size of the array changes. For instance, the `ReDim Preserve intNums(4)` statement in Example 1 adds two elements to the end of the `intNums` array while preserving the values stored in the first three elements. The `ReDim intNums(4)` statement in Example 2 also adds two elements to the end of the `intNums` array; however, notice that the values stored in the first three elements are not saved. As Example 3 indicates, if you use the ReDim statement to reduce the size of an array, the values in the truncated elements are not saved. An array whose number of elements changes while an application is running is referred to as a **dynamic array**.

ReDim Statement

Syntax
ReDim [Preserve] *arrayName(highestSubscript)*

Example 1
```
ReDim Preserve intNums(4)
```

Original `intNums` array:

100
120
230

Result of ReDim statement:

100
120
230
0
0

Example 2
```
ReDim intNums(4)
```

Original `intNums` array:

100
120
230

Result of ReDim statement:

0
0
0
0
0

Example 3
```
ReDim intNums(1)
```

Original `intNums` array:

100
120
230

Result of ReDim statement:

100
120

Figure 19-11 Syntax and examples of the ReDim statement

Ch19-Dynamic Arrays

You will use the ReDim statement in the ReDim application, which you finish coding in this section. The application allows the user to enter as many sales amounts as needed. It stores the sales amounts in a dynamic one-dimensional array and then displays the sales amounts in the interface.

To open the ReDim application:

1. Open the **ReDim Solution** (**ReDim Solution.sln**) file contained in the ClearlyVB2012\Chap19\ReDim Solution folder. If necessary, open the designer window. The application's interface is shown in Figure 19-12.

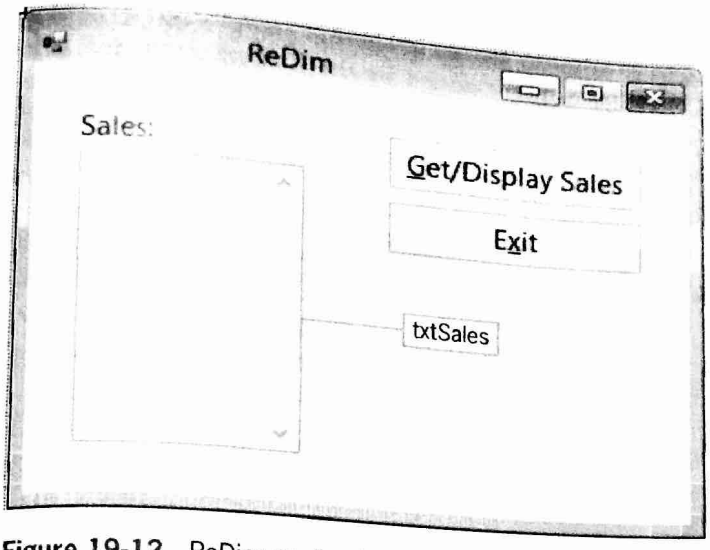

Figure 19-12 ReDim application's interface

2. Open the Code Editor window and locate the btnDisplay_Click procedure. Most of the procedure has already been coded for you. The procedure uses the InputBox function and a loop to get the sales amounts from the user. Keep in mind that the user may enter one amount, 10 amounts, or even 100 amounts. It's also possible that he or she may not enter any sales amounts.

The btnDisplay_Click procedure will store the sales amounts (if any) in an array named `decSales`. However, because the number of sales amounts the user will enter is unknown, the array will need to be declared as an empty array. An **empty array** is an array that contains no elements. You declare an empty array using an empty set of braces in the array's declaration statement. For example, you would use the statement `Private decSales() As Decimal = {}` to declare the `decSales` array as a class-level array, and use the statement `Dim decSales() As Decimal = {}` to declare it as a procedure-level array.

To declare the array:

1. Click the **blank line** below the `' class-level array` comment in the form's Declarations section, and then enter the following declaration statement:

   Private decSales() As Decimal = {}

The btnDisplay_Click procedure uses the InputBox function to get a sales amount from the user, and it stores the user's response in the `strSales` variable. The loop in the procedure is processed as long as the `strSales` variable does not contain the empty string. Before you can convert the user's input to Decimal and then store it in the `decSales` array, you first need to add an element to the array. You can accomplish this task using the statement `ReDim Preserve decSales(intSub)`. The first time the statement is processed, the `intSub` variable will contain the number 0. As a result, the computer will change the size of the array to one element. (Recall that the subscript of the first element in a one-dimensional array is 0.)

To finish coding the btnDisplay_Click procedure:

1. In the btnDisplay_Click procedure, click the **blank line** below the `' add an element to the array` comment and then enter the following statement:

   ReDim Preserve decSales(intSub)

2. Now you will convert the sales amount stored in the `strSales` variable to the Decimal data type and then store it in the element added by the ReDim statement. Click the **blank line** below the `' store the sales amount in the array` comment and then enter the following statement:

   Decimal.TryParse(strSales, decSales(intSub))

3. Finally, you will update the array subscript by adding the number 1 to the contents of the `intSub` variable. You can accomplish this task using either the statement `intSub = intSub + 1` or the statement `intSub += 1`; both statements are equivalent. If the ReDim statement in the loop is processed a second time, the `intSub` variable will contain the number 1 and the computer will change the size of the array to two elements. Click the **blank line** below the `' update the array subscript` comment and then enter the following statement:

   intSub += 1

Figure 19-13 shows the code contained in the form's Declarations section and btnDisplay_Click procedure. The code you entered is shaded in the figure.

form's Declarations section

```
' class-level array
Private decSales() As Decimal = {}

Private Sub btnDisplay_Click(sender As Object,
e As EventArgs) Handles btnDisplay.Click
    ' displays the sales amounts stored in an array

    Const strPROMPT As String =
        "Enter a sales amount. Click Cancel to end."
    Dim strSales As String
    Dim intSub As Integer

    ' start array subscript at 0
    intSub = 0
    ' get a sales amount
    strSales = InputBox(strPROMPT, "ReDim")
    Do While strSales <> String.Empty
        ' add an element to the array
        ReDim Preserve decSales(intSub)

        ' store the sales amount in the array
        Decimal.TryParse(strSales, decSales(intSub))

        ' update the array subscript
        intSub += 1

        ' get the next sales amount
        strSales = InputBox(strPROMPT, "ReDim")
    Loop

    ' display the sales amounts
    txtSales.Text = String.Empty
    For intX As Integer = 0 To decSales.Length - 1
        txtSales.Text = txtSales.Text &
            decSales(intX).ToString("N2") & ControlChars.NewLine
    Next intX
End Sub
```

Figure 19-13 Form's Declarations section and btnDisplay_Click procedure

To test the btnDisplay_Click procedure:

1. Save the solution and then start the application. First, you will enter two sales amounts. Click the **Get/Display Sales** button. Type **5000** and press **Enter**, and then type **7500** and press **Enter**. Click the **Cancel** button. The two sales amounts appear in the txtSales control. See Figure 19-14.

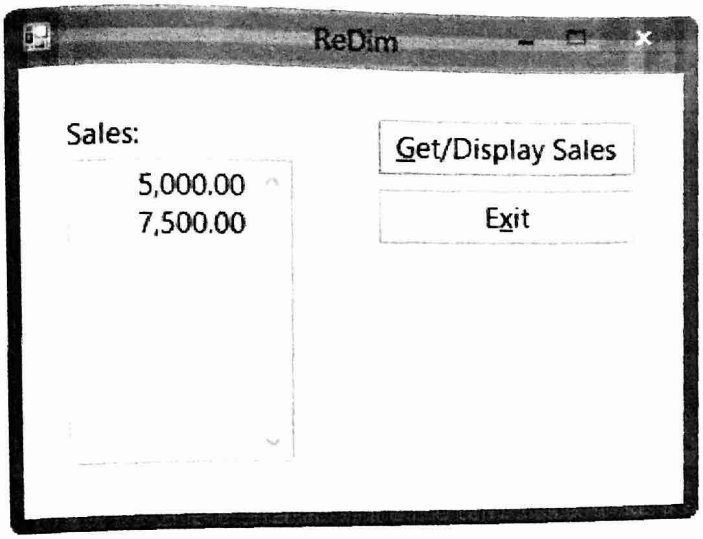

Figure 19-14 Interface showing the two sales amounts stored in the array

2. Now you will enter five sales amounts. Click the **Get/Display Sales** button. Enter the following five numbers, one at a time: **250.05**, **300**, **1000**, **25.67**, and **75**. Click the **Cancel** button. The five sales amounts appear in the txtSales control, as shown in Figure 19-15.

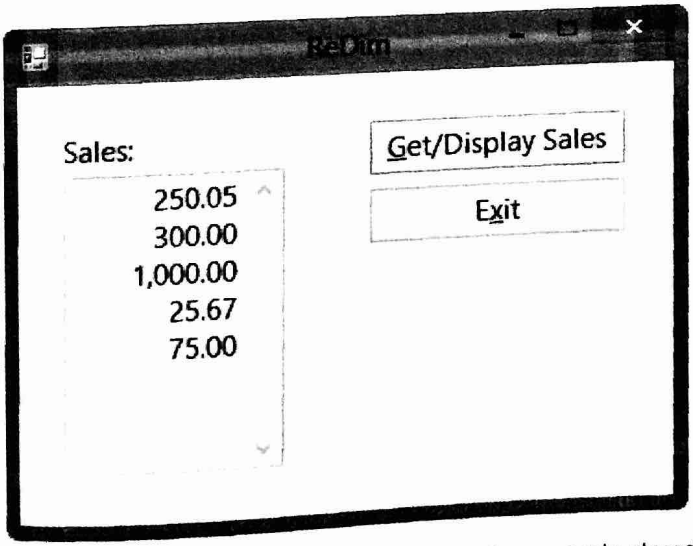

Figure 19-15 Interface showing the five sales amounts stored in the array

3. Click the **Exit** button. Close the Code Editor window and then close the solution.