MODIFY THIS

3. In this exercise, you modify the Test Scores application coded in the chapter. Use Windows to make a copy of the Test Scores Solution folder. Save the copy in the ClearlyVB2012\Chap19 folder. Rename the copy Modified Test Scores Solution. Open the Test Scores Solution (Test Scores Solution.sln) file contained in the Modified Test Scores Solution folder. Open the designer and Code Editor windows. Modify the code to allow the user to enter as many test scores as needed. (Hint: It's possible that the user may not enter any test scores.) Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

MODIFY THIS

4. In this exercise, you modify the West Coast Emporium application coded in the chapter. Use Windows to make a copy of the West Coast Solution folder. Save the copy in the ClearlyVB2012\Chap19 folder. Rename the copy Modified West Coast Solution. Open the West Coast Solution (West Coast Solution.sln) file contained in the Modified West Coast Solution folder.

   a. Open the designer window. Change the btnNumberOfStores control's name to btnDisplay, and change its Text property to &Display. Modify the interface so that it also displays the number of employees in the state entered by the user.

   b. Open the Code Editor window. Declare another class-level array to store the number of employees in each state. Use your own values to initialize the array.

   c. Locate the btnNumberOfStores_Click procedure. Change the name to btnDisplay_Click and then modify the code to also display the number of employees.

   d. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

5. In this exercise, you code an application that displays a grade based on the number of points entered by the user. The grading scale is shown in Figure 19-16. Open the Carver Solution (Carver Solution.sln) file contained in the ClearlyVB2012\Chap19\Carver Solution folder. Open the Code Editor window. Store the minimum points in a five-element one-dimensional integer array. Store the grades in a five-element one-dimensional String array. Both arrays should be class-level parallel arrays. Open the code template for the btnDisplay control's Click event procedure. The procedure should display the grade corresponding to the number of points entered by the user. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

Minimum Points	Maximum Points	Grade
0	299	F
300	349	D
350	399	C
400	449	B
450	500	A

Figure 19-16 Grading scale for Exercise 5

Using an Element in a Two-Dimensional Array

Example 1

```
intScores(0, 1) = 95
```

assigns the number 95 to the element located in the first row, second column in the `intScores` array

Example 2

```
intSalaries(3, 0) = intSalaries(3, 0) + 2000
```

adds 2000 to the contents of the element located in the fourth row, first column in the `intSalaries` array and then assigns the result to the element; you can also write this statement as `intSalaries(3, 0) += 2000`

Example 3

```
Decimal.TryParse(txtSales.Text, decSales(2, 1))
lblSales.Text = decSales(2, 1).ToString("C2")
```

assigns the value returned by the TryParse method to the element located in the third row, second column in the `decSales` array and then displays the value (formatted with a dollar sign and two decimal places) in the lblSales control

Example 4

```
For intRow As Integer = 0 To 5
     For intColumn As Integer = 0 To 3
          intNumbers(intRow, intColumn) = 0
     Next intColumn
Next intRow
```

assigns the number 0 to each element in the six-row, four-column `intNumbers` array

Figure 20-3 Examples of using an element in a two-dimensional array

Mini-Quiz 20-1

See Appendix B for the answers.

1. Write a Private statement that declares an Integer array named `intQuantities`. The array should have four rows and two columns.
2. What is the highest row subscript in the `intQuantities` array from Question 1?
3. Write a statement that assigns the number 7 to the element located in the third row, first column in the `intQuantities` array.

Revisiting the West Coast Emporium Application

You coded the West Coast Emporium application in Chapter 19. The application stores three state IDs in a one-dimensional String array named `strStates`, and it stores the corresponding number of stores in a parallel one-dimensional Integer array named `intStores`. The arrays are illustrated in Figure 20-4. As you may remember, the application searches for the ID (which is entered by the user) in the `strStates` array, beginning with the first element in the array. If it finds the ID, it displays the corresponding number of stores from the `intStores` array; otherwise, it displays the number 0.

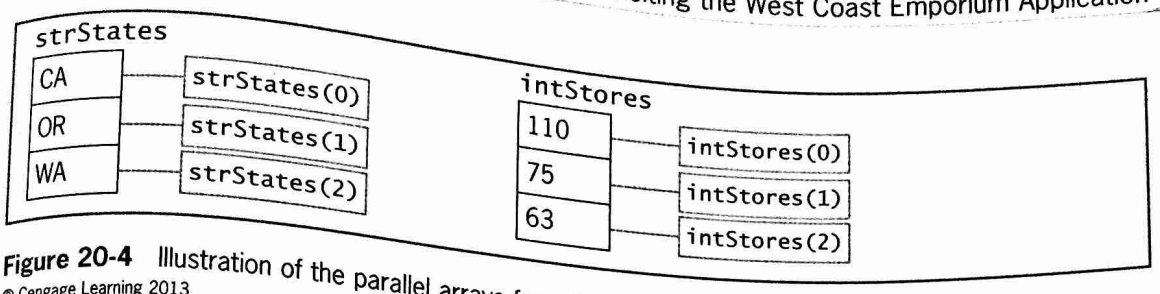

Figure 20-4 Illustration of the parallel arrays from Chapter 19

Instead of storing the state and store information in parallel one-dimensional arrays, you can store the information in a two-dimensional array, with the state IDs in the first column and the corresponding number of stores in the second column. However, to do this, you will need to treat the store information as strings rather than as numbers. This is because the state IDs are strings and all of the elements in an array must have the same data type. The two-dimensional array, named `strStateInfo`, is illustrated in Figure 20-5. Each state ID is stored in the first column of the array, and its associated number of stores is stored (as a string) in the corresponding row in the second column.

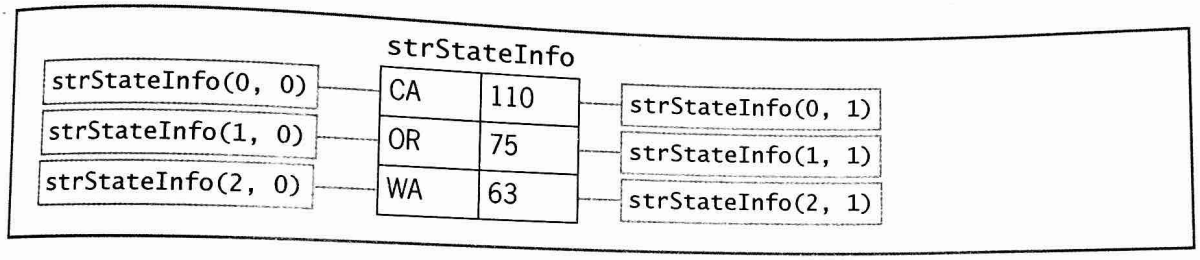

Figure 20-5 Illustration of the two-dimensional `strStateInfo` array

In this case, the application will search for the state ID in the first column of the `strStateInfo` array. It will search the column, row by row, beginning with the first row. If it finds the ID, it will display the corresponding number of stores from the second column of the array. Otherwise, it will display the number 0. Figure 20-6 shows the planning information for the application, using a two-dimensional array.

```
Output:      number of stores or 0

Processing:  two-dimensional (three rows, three columns) store information
             array (class-level)
             row subscript
             number of rows
             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 row in the array by starting the row subscript
   at 0
3. assume that the state ID is not in the array by starting the search results variable at
   "N"
4. determine the number of rows in the array
5. repeat until the search results variable contains "Y" or the row subscript equals
   the number of rows in the array:
      if the state ID in the first column of the current row matches the state ID to
      search for, do this:
            assign "Y" to the search results variable
      otherwise, do this:
            add 1 to the row subscript
      end if
   end repeat
6. if the search results variable contains "Y", do this:
      display the corresponding number of stores from the second column of the
      current row
   otherwise, do this:
      display 0
   end if
```

Figure 20-6 Planning information for the West Coast Emporium application using a two-dimensional array

To use a two-dimensional array to code the West Coast Emporium application:

1. Start Visual Studio 2012. Open the **West Coast Solution (West Coast Solution.sln)** file contained in the ClearlyVB2012\Chap20\West Coast Solution folder. If necessary, open the designer window.

2. Open the Code Editor window. First, you will declare the two-dimensional array. Click the **blank line** below the `' class-level two-dimensional array` comment in the form's Declarations section and then enter the following array declaration statement:

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

3. Next, you will code the btnNumberOfStores_Click procedure. The procedure will use four variables. The `strSearchFor` variable will store the ID to search for in the array. The `intRow` variable will keep track of the row subscripts during the search, and the `intNumRows` variable will store an integer that represents the number of rows in the array. The `strFound` variable will keep track of whether the ID was found in the first column of the array. Locate the btnNumberOfStores_Click procedure. Click the **blank line** below the `' declare variables` comment and then enter the following four declaration statements:

```
Dim strSearchFor As String
Dim intRow As Integer
Dim intNumRows As Integer
Dim strFound As String
```

4. The first step in the algorithm is to enter the state ID to search for in the 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 row subscript at 0. In the btnNumberOfStores_Click procedure, the `intRow` variable keeps track of the row 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:

```
intRow = 0
```

6. Before the search begins, the procedure will assume that the state ID is not contained in the 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"
```

efore coding the fourth step in the algorithm, you will learn about an array's GetUpperBound ethod.

ne GetUpperBound Method

gure 20-7 shows the syntax of an array's **GetUpperBound method**, which returns an integer t indicates the highest subscript in the specified dimension of the array. In the syntax, *ayName* is the name of the array, and *dimension* is an integer that specifies the dimension ose upper bound you want to retrieve. When used with a one-dimensional array, the cified dimension is always 0. When used with a two-dimensional array, on the other hand, specified dimension is either 0 or 1: The 0 represents the row dimension and the 1 resents the column dimension. Also shown in Figure 20-7 are examples of using the UpperBound method with one-dimensional and two-dimensional arrays.

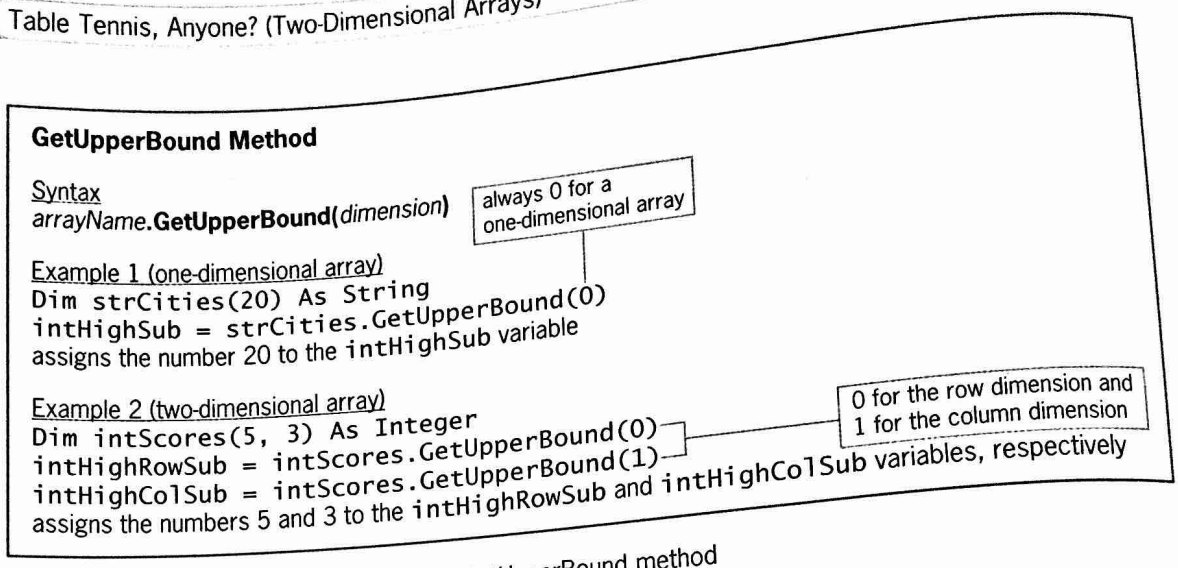

Figure 20-7 Syntax and examples of the GetUpperBound method

Recall that the fourth step in the algorithm from Figure 20-6 determines the number of rows in the `strStateInfo` array. To accomplish this task, you first will use the array's GetUpperBound method to get the highest row subscript. You then will increase that value by 1 because the number of rows in an array is always one number more than the highest row subscript.

To determine the number of rows in the array:

1. The insertion point should be positioned below the `strFound = "N"` statement. Enter the following assignment statement:

   intNumRows = strStateInfo.GetUpperBound(0) + 1

The fifth step in the algorithm is a pretest loop that repeats its instructions until one of two conditions is true: either the state ID has been found in the first column in the array, or the row subscript equals the number of rows in the array (which indicates there are no more rows 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 btnNumberOfStores_Click procedure:

1. Enter the following Do clause:

   Do Until strFound = "Y" OrElse intRow = intNumRows

2. The selection structure within the loop compares the state ID stored in the first colu of the current row in the array with the state ID stored in the `strSearchFor` variabl 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 n match, the selection structure's false path will increment the array subscript by 1 so the loop can search the next row in the array. Enter the following selection structu

   ```
   If strStateInfo(intRow, 0) = strSearchFor Then
       strFound = "Y"
   Else
       intRow += 1
   End If
   ```

3. The value stored in the `strFound` variable indicates whether the ID was located in the first column in the 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 = strStateInfo(intRow, 1)
Else
    lblStores.Text = "0"
End If
```

Figure 20-8 shows the code entered in the form's Declarations section and btnNumberOfStores_Click procedure.

```
' class-level two-dimensional array
Private strStateInfo(,) As String = {{"CA", "110"},
                                     {"OR", "75"},
                                     {"WA", "63"}}

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

    ' declare variables
    Dim strSearchFor As String
    Dim intRow As Integer
    Dim intNumRows As Integer
    Dim strFound As String

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

    ' search the first column for the ID
    ' continue searching until the ID is
    ' found or the end of the array
    intRow = 0
    strFound = "N"
    intNumRows = strStateInfo.GetUpperBound(0) + 1
    Do Until strFound = "Y" OrElse intRow = intNumRows
        If strStateInfo(intRow, 0) = strSearchFor Then
            strFound = "Y"
        Else
            intRow += 1
        End If
    Loop

    ' determine whether the ID was found
    ' and display the appropriate output
    If strFound = "Y" Then
        lblStores.Text = strStateInfo(intRow, 1)
    Else
        lblStores.Text = "0"
    End If
End Sub
```

form's Declarations section

Figure 20-8 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 **wa** in the State ID box. The state ID appears in the `strStateInfo(2, 0)` element, and its corresponding number of stores (63) appears in the `strStateInfo(2, 1)` element. Click the **Number of Stores** button. The correct number of stores appears in the Stores box, as shown in Figure 20-9. (Recall that you can use the Alt key to show/hide the access keys.)

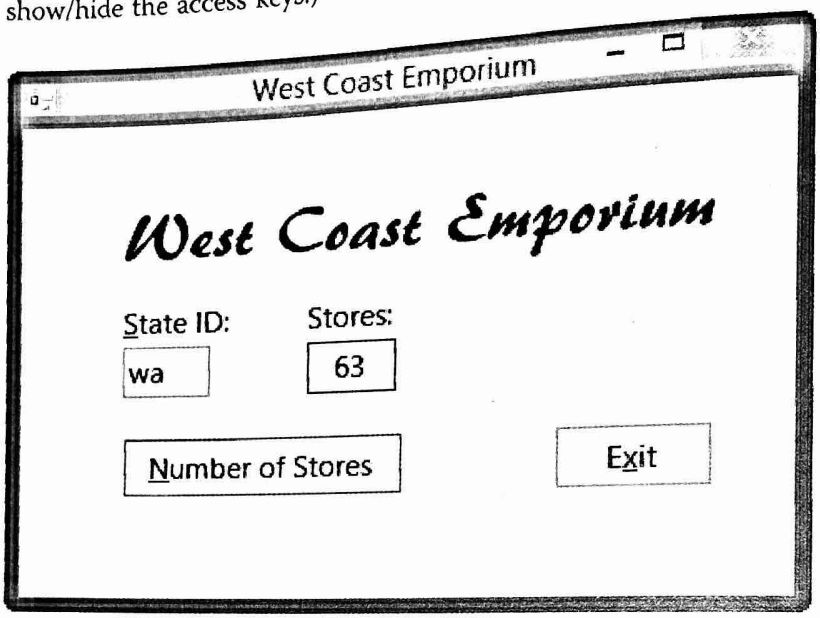

Figure 20-9 Number of stores displayed in the interface

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.

Calendar Sales Application

The Calendar Sales application displays the total sales made by three stores in each of six months. The monthly sales amounts for each store are stored in a two-dimensional array named `intSales`. The array has three rows and six columns: Each row represents a store, and each column represents a month. To display the total sales, you will need to accumulate the sales amounts stored in the array.

To code the Calendar Sales application:

1. Open the **Calendar Solution (Calendar Solution.sln)** file contained in the ClearlyVB2012\Chap20\Calendar Solution folder. If necessary, open the designer window. See Figure 20-10. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.)

Medical Bills Application

The Medical Bills application displays the total amount paid to doctors, pharmacies, and dentists. The amounts paid in each category are stored in a two-dimensional array named `strMedBills`. The array has three rows and two columns, as illustrated in Figure 20-14. To display the total amount paid, you will need to accumulate the amounts stored in the array's second column.

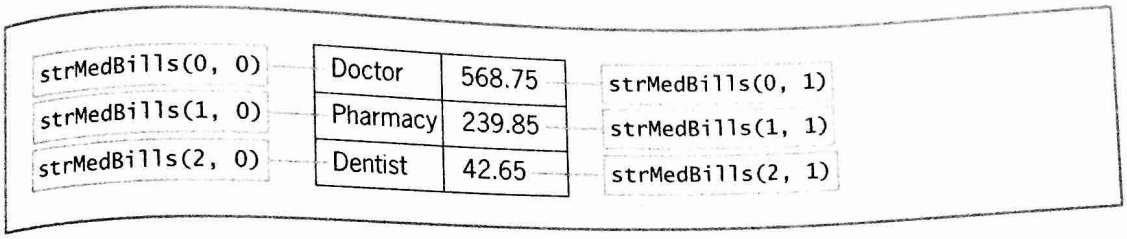

Figure 20-14 Illustration of the two-dimensional `strMedBills` array

To code the Medical Bills application:

1. Open the **Medical Solution** (**Medical Solution.sln**) file contained in the ClearlyVB2012\Chap20\Medical Solution folder. If necessary, open the designer window.
2. Open the Code Editor window. The class-level `strMedBills` array is already declared in the form's Declarations section.
3. Locate the btnDisplay_Click procedure. Most of the procedure has already been coded for you. You just need to enter a loop that will accumulate the values stored in the second column of the array. Click the **blank line** below the `' accumulate the values stored in the second column` comment and then enter the following For clause:

   For intRow As Integer = 0 To strMedBills.GetUpperBound(0)
4. Change the `Next` clause to **Next intRow**.
5. Before you can add the first array value to the `decTotal` accumulator, you need to convert the value from String to Decimal. Click the **blank line** below the For clause and then enter the following comment and TryParse method:

   ' convert array value to Decimal
   Decimal.TryParse(strMedBills(intRow, 1), decAmount)
6. Now you can add the converted value to the accumulator. Type the following assignment statement and then click the **blank line** below the `Next intRow` clause:

   decTotal += decAmount

Figure 20-15 shows the code entered in the form's Declarations section and btnDisplay_Click event procedure.

form's Declarations section

```
' class-level array
Private strMedBills(,) As String = {{"Doctor", "568.75"},
                                    {"Pharmacy", "239.85"},
                                    {"Dentist", "42.65"}}

Private Sub btnDisplay_Click(sender As Object,
e As EventArgs) Handles btnDisplay.Click
    ' calculates and displays the total paid

    Dim decAmount As Decimal
    Dim decTotal As Decimal ' accumulator

    ' accumulate the values stored in the second column
    For intRow As Integer = 0 To strMedBills.GetUpperBound(0)
        ' convert array value to Decimal
        Decimal.TryParse(strMedBills(intRow, 1), decAmount)
        decTotal += decAmount
    Next intRow

    ' display the total paid
    lblTotal.Text = decTotal.ToString("C2")
End Sub
```

you can also use `decTotal = decTotal + decAmount`

Figure 20-15 Form's Declarations section and btnDisplay_Click procedure

To test the Medical Bills application:

1. Save the solution and then start the application. Click the **Display** button. The total amount paid appears in the Total paid box, as shown in Figure 20-16.

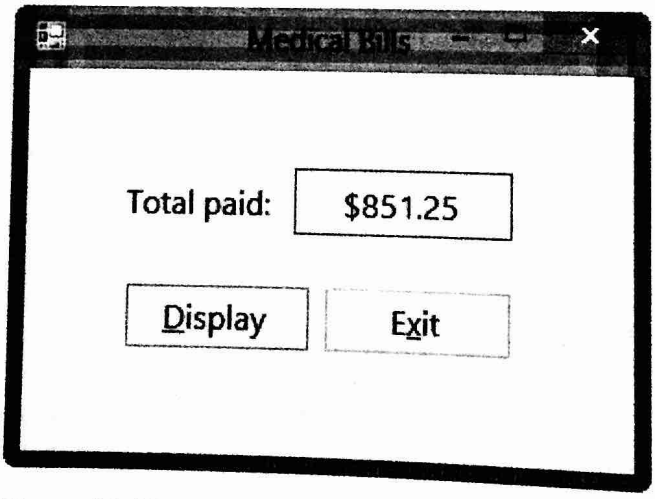

Figure 20-16 Total amount paid displayed in the interface

To learn more about two-dimensional arrays, see e Two-Dimensional rays section in the n20WantMore.pdf file.

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

Mini-Quiz 20-2

See Appendix B for the answers.

1. If the `decSales` array is a two-dimensional array, which of the following assigns the highest column subscript to the `intHighCol` variable?
   a. `intHighCol = decSales.GetHighestSub(0)`
   b. `intHighCol = decSales.GetHighBound(1)`
   c. `intHighCol = decSales.GetUpperBound(0)`
   d. none of the above

2. Which of the following adds the number 20 to the value stored in the first row, second column of the `decSales` array?
   a. `decSales(0, 1) += 20`
   b. `decSales(1, 2) = decSales(1, 2) + 20`
   c. `decSales(1, 2) += 20`
   d. both b and c

3. Which of the following will repeat the loop instructions for each row in a two-dimensional array named `intNum`?
   a. `For intRow As Integer = 0 To intNum.GetUpperBound(0)`
   b. `For intRow As Integer = 0 To intNum.GetUpperBound(1)`
   c. `intRow = 0`
      `Do While intRow < intNum.GetUpperBound(0)`
   d. both b and c

Summary

- A two-dimensional array resembles a table in that the variables (elements) are in rows and columns.
- You can determine the number of elements in a two-dimensional array by multiplying the number of its rows by the number of its columns.
- Each element in a two-dimensional array is identified by a unique combination of two subscripts: a row subscript and a column subscript. The subscripts appear in parentheses after the array's name. You list the row subscript first, followed by a comma and the column subscript.
- The first row subscript in a two-dimensional array is 0. The first column subscript is also 0.
- When declaring a two-dimensional array, you provide either the highest row and column subscripts or the initial values.
- The number of rows in a two-dimensional array is one number more than its highest row subscript. Likewise, the number of columns is one number more than its highest column subscript.
- You refer to an element in a two-dimensional array using the array's name followed by the element's row and column subscripts, which are separated by a comma and enclosed in parentheses.

9. Which of the following will assign the number of elements in the `intSales` array from Review Question 8 to the `intElements` variable?

   a.
   ```
   intElements =
       (intSales.GetUpperBound(0) + 1) *
       (intSales.GetUpperBound(1) + 1)
   ```

   b.
   ```
   intElements =
       intSales.GetUpperBound(0) *
       intSales.GetUpperBound(1)
   ```

   c.
   ```
   intElements =
       intSales.GetNumRows *
       intSales.GetNumColumns
   ```

   d. `intElements = intSales.Length`

10. Which of the following assigns the number of columns in the `strBooks` array to the `intNumCols` variable?

    a. `intNumCols = strBooks.Length(1)`
    b. `intNumCols = strBooks.GetUpperBound(1) + 1`
    c. `intNumCols = strBooks.GetUpperBound(1) - 1`
    d. `intNumCols = strBooks.GetUpperBound(1)`

Exercises

TRY THIS

1. Open the Price List Solution (Price List Solution.sln) file contained in the ClearlyVB2012\Chap20\Price List Solution folder. Open the designer window. The interface provides a text box for the user to enter a product ID. Open the Code Editor window. Declare a class-level, two-dimensional String array that contains the following product IDs and prices: BX35, 13, CR20, 10, FE15, 12, KW10, 24, MM67, and 4. Open the code template for the btnDisplay control's Click event procedure. The procedure should display either the price associated with the product ID or the "Invalid product ID" message. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. (See Appendix B for the answer.)

TRY THIS

2. Open the Inventory Solution (Inventory Solution.sln) file contained in the ClearlyVB2012\Chap20\Inventory Solution folder. Open the designer and Code Editor windows. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.) Locate the btnDisplay_Click procedure. The procedure should add together the values stored in the `intInventory` array and then display the total in the lblTotal control. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. (See Appendix B for the answer.)

MODIFY THIS

3. In this exercise, you modify the Medical Bills application coded in the chapter. Use Windows to make a copy of the Medical Solution folder. Save the copy in the ClearlyVB2012\Chap20 folder. Rename the copy Modified Medical Solution. Open the Medical Solution (Medical Solution.sln) file contained in the Modified Medical Solution folder. Open the designer and Code Editor windows. Modify the array to include a row for the $75 paid to an optometrist. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

MODIFY THIS

4. In this exercise, you modify the West Coast Emporium application coded in the chapter. Use Windows to make a copy of the West Coast Solution folder. Save the copy