```
Structure Dimensions
    Public dblLength As Double
    Public dblWidth As Double
    Public dblDepth As Double
End Structure

Public Function GetGallons(ByVal pool As Dimensions) As Double
    ' calculates and returns the number of gallons

    Const dblGAL_PER_CUBIC_FOOT As Double = 7.48

    Return pool.dblLength * pool.dblWidth *
        pool.dblDepth * dblGAL_PER_CUBIC_FOOT
End Function

Private Sub btnCalc_Click(sender As Object,
e As EventArgs) Handles btnCalc.Click
    ' displays the number of gallons

    Dim poolSize As Dimensions
    Dim dblGallons As Double

    Double.TryParse(txtLength.Text, poolSize.dblLength)
    Double.TryParse(txtWidth.Text, poolSize.dblWidth)
    Double.TryParse(txtDepth.Text, poolSize.dblDepth)

    dblGallons = GetGallons(poolSize)
    lblGallons.Text = dblGallons.ToString("N0")
End Sub
```

form's Declarations section

receives a structure variable *by value*

declares a structure variable to store the input data

assigns the input data to the structure variable

passes the structure variable to the GetGallons function

Figure 21-7 Code for the Bonnette Pool & Spa Depot application (with a structure)

To test the modified code:

1. Save the solution and then start the application. Type **75**, **25**, and **5** in the Length, Width, and Depth boxes, respectively. Click the **Calculate** button. The number 70,125 appears in the Gallons box, as shown earlier in Figure 21-5.
2. Click the **Exit** button. Close the Code Editor window and then close the solution.

Revisiting the West Coast Emporium Application...Again!

As mentioned earlier, another advantage of using a structure is that a structure variable can be stored in an array, even when its members have different data types. The West Coast Emporium application from the previous two chapters can be used to illustrate this concept. As you may remember, the application's interface provides a text box for the user to enter a state ID. The application searches for the ID in an array. If it finds the ID, it displays the corresponding number of stores; otherwise, it displays the number 0. In Chapter 19, you stored the state IDs and numbers of stores in two parallel one-dimensional arrays: a String array for the IDs and an Integer array for the numbers of stores. In Chapter 20, you stored the information in a two-dimensional String array. Recall that in order to do so, you had to treat the store information as strings; this is because all of the elements in an array must have the same data type. The arrays from Chapters 19 and 20 are illustrated in Figure 21-8.

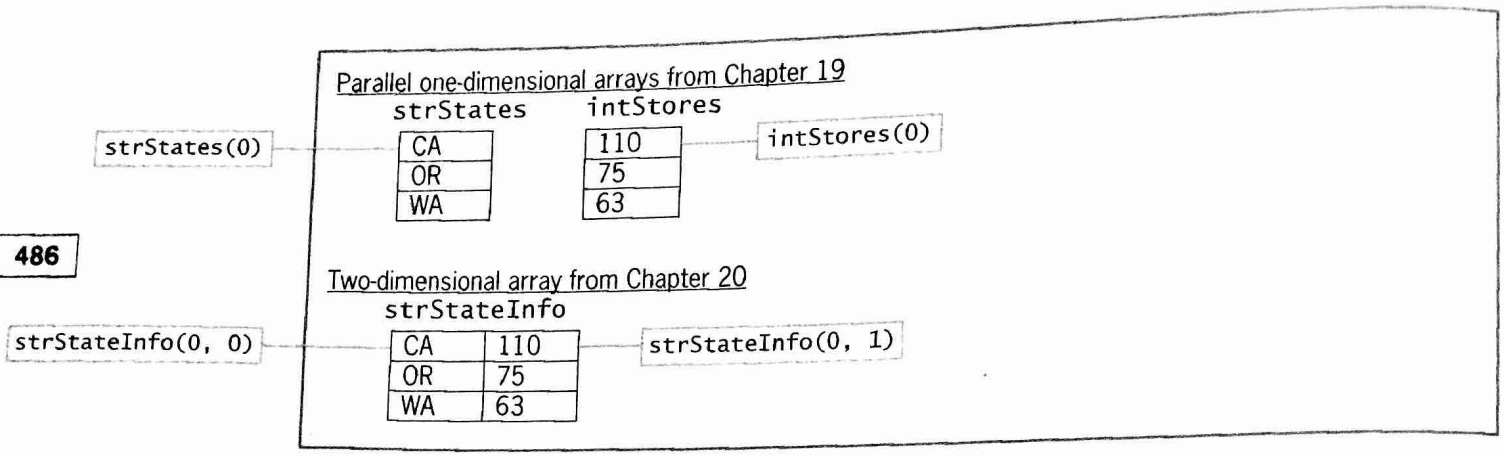

Figure 21-8 Illustration of the arrays from Chapters 19 and 20

Rather than coding the application using either parallel one-dimensional arrays or a two-dimensional array, you can use a one-dimensional array of structure variables. The structure, which you will name StateInfo, will contain two member variables: `strId` and `intStores`. The array, which you will name `states`, will contain three structure variables because there are three states. Each structure variable in the array will store the state's ID and the number of stores in the state, as illustrated in Figure 21-9. You refer to a member variable in an array using the syntax *arrayName*(*subscript*)*.memberVariableName*. For example, `states(0).strId` refers to the `strId` member contained in the first element in the `states` array. Likewise, `states(2).intStores` refers to the `intStores` member contained in the last element in the `states` array.

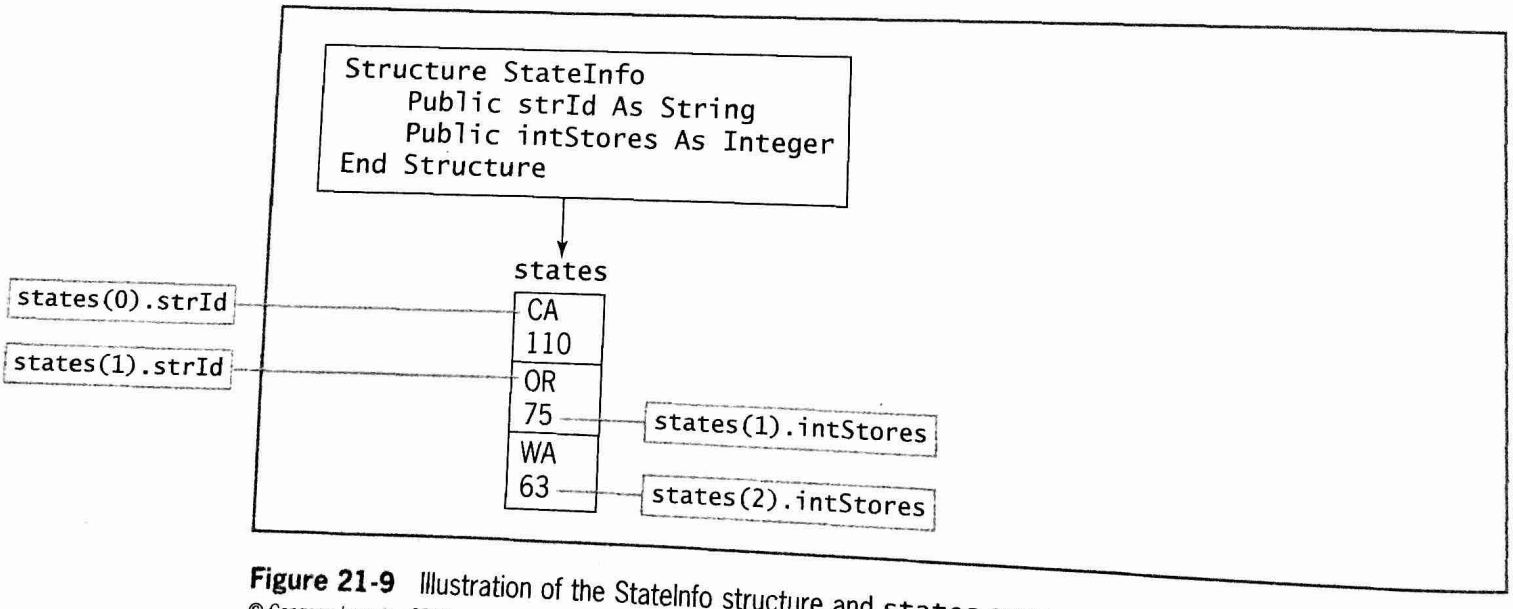

Figure 21-9 Illustration of the StateInfo structure and `states` array

To use a structure in the West Coast Emporium application:

1. Open the **West Coast Solution (West Coast Solution.sln)** file contained in the ClearlyVB2012\Chap21\West Coast Solution folder. If necessary, open the designer window. See Figure 21-10.

Figure 21-10 Interface for the West Coast Emporium application

2. Open the Code Editor window. First, you will create the structure. Click the **blank line** below the `' structure` comment in the form's Declarations section and then enter the following Structure statement:

```
Structure StateInfo
    Public strId As String
    Public intStores As Integer
End Structure
```

3. Next, you will declare a three-element one-dimensional array, using the StateInfo structure as the array's data type. Click the **blank line** below the `' class-level array` comment in the form's Declarations section and then enter the following declaration statement:

```
Dim states(2) As StateInfo
```

4. Now, locate the frmMain_Load procedure. You will use this procedure to fill the array with values. Click the **blank line** above the End Sub clause and then enter the following assignment statements:

```
states(0).strId = "CA"
states(0).intStores = 110
states(1).strId = "OR"
states(1).intStores = 75
states(2).strId = "WA"
states(2).intStores = 63
```

5. 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 array. The `intSub` variable will keep track of the array subscripts during the search, and the `strFound` variable will keep track of whether the ID was found in the 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
```

6. Now you will assign the state ID, which the user enters in the txtState control, to the `strSearchFor` variable. Before doing so, however, you will trim any leading and/or trailing spaces from the ID and then convert it to uppercase. 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

7. The search should begin with the first element in the array. The `intSub` variable in the procedure 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

8. Before the search begins, the procedure will assume that the state ID is not contained in the array. Enter the following assignment statement:

   strFound = "N"

9. Next, you will enter a loop that repeats its instructions until one of two conditions is true: either the state ID has been found in the array or the subscript equals the number of array elements (which indicates there are no more elements to search). Enter the following Do clause:

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

10. Now you will use a selection structure that compares the contents of the `strId` member in the current array element 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 array. Enter the following selection structure:

```
If states(intSub).strId = strSearchFor Then
    strFound = "Y"
Else
    intSub += 1
End If
```

11. The value stored in the `strFound` variable indicates whether the ID was located 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 = states(intSub).intStores
Else
    lblStores.Text = "0"
End If
```

Figure 21-11 shows most of the application's code.

```
' structure
Structure StateInfo
    Public strId As String
    Public intStores As Integer
End Structure

' class-level array
Dim states(2) As StateInfo

Private Sub frmMain_Load(sender As Object,
e As EventArgs) Handles Me.Load
    ' assign values to the array

    states(0).strId = "CA"
    states(0).intStores = 110
    states(1).strId = "OR"
    states(1).intStores = 75
    states(2).strId = "WA"
    states(2).intStores = 63

End Sub

Private Sub btnNumberOfStores_Click(sender As Object,
e As EventArgs) Handles btnNumberOfStores.Click
    ' searches for a state ID in an array of
    ' structure variables and then displays
    ' either the number of stores 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 for the ID in the array
    ' continue searching until the ID is
    ' found or the end of the array
    intSub = 0
    strFound = "N"
    Do Until strFound = "Y" OrElse intSub = states.Length
        If states(intSub).strId = strSearchFor Then
            strFound = "Y"
        Else
            intSub += 1
        End If
    Loop

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

declares an array of structure variables

form's Declarations section

assigns values to the array

compares the value stored in the current element's strId member

displays the value stored in the current element's intStores member

Figure 21-11 Most of the West Coast Emporium application's code

To test the application's code:

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 `states(1).strId` element, and its corresponding number of stores (75) appears in the `states(1).intStores` element. Click the **Number of Stores** button. The correct number of stores appears in the Stores box, as shown in Figure 21-12.

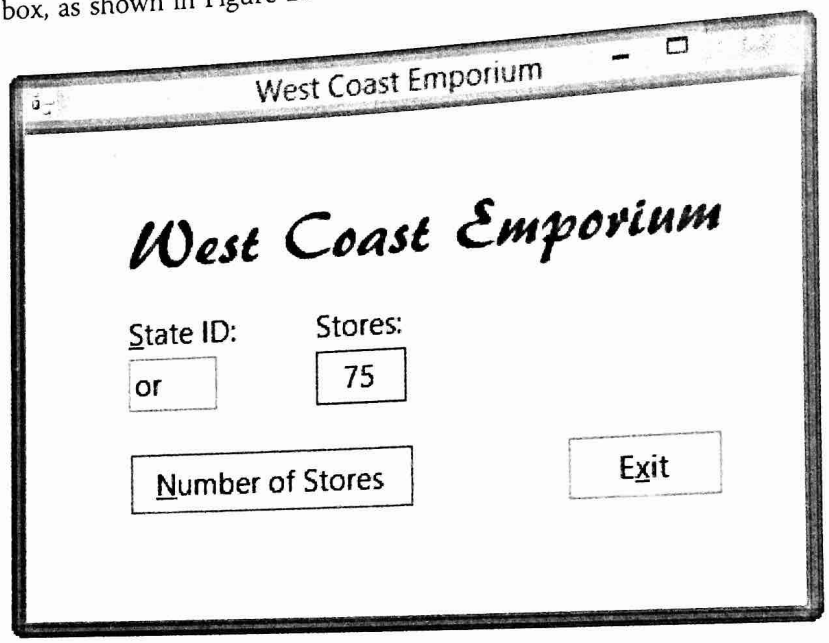

Figure 21-12 Number of stores displayed in the interface

To learn more about structures, see the Structures section in the Ch21WantMore.pdf file.

Ch21-Structures

2. Now you will enter an invalid state ID. Change the state ID to **ky** and then click the **Number of Stores** button. The number 0 appears in the Stores box.
3. Click the **Exit** button. Close the Code Editor window and then close the solution.

As you observed by coding the West Coast Emporium application in this chapter and in the previous two chapters, there are many different ways of solving the same problem. Most times, the "best" way is simply a matter of personal preference.

Mini-Quiz 21-1

See Appendix B for the answers.

1. The Structure statement is usually entered in the form's _____ section in the Code Editor window.
2. Write a Visual Basic statement that assigns the string "Maple" to the `strStreet` member of a structure variable named `address`.
3. An array of structure variables is declared using the statement `Dim inventory(4) As Product`. Write a Visual Basic statement that assigns the number 100 to the `intQuantity` member contained in the last array element.

Exercises

1. Open the Commission Solution (Commission Solution.sln) file contained in the ClearlyVB2012\Chap21\Commission Solution folder. Open the designer window. The interface provides three text boxes for the user to enter the sales for three regions. Open the Code Editor window. Declare a structure named SalesInfo. The structure should contain three Decimal member variables to store the three sales amounts. The btnCalc control's Click event procedure should store the contents of the text boxes in a SalesInfo structure variable. It then should use a function named GetCommission to sum the three sales amounts. The function should also calculate and return a 3% commission. Finally, the Click event procedure should display the commission (formatted with a dollar sign and two decimal places) in the lblComm control. Code the application. 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 Price List Solution (Price List Solution.sln) file contained in the ClearlyVB2012\Chap21\Price List Solution folder. Open the designer window. The interface provides a text box for the user to enter a product ID. When the user clicks the Display Price button, the button's Click event procedure should display either the price associated with the product ID or the "Invalid product ID" message. (Display the message in a message box.) Open the Code Editor window. Declare a structure named Item. The structure should contain two members: a String variable for the product ID and an Integer variable for the price. Declare a class-level five-element array of Item structure variables; name the array `gifts`. Use the form's Load event procedure to assign the following IDs and prices to the `gifts` array: BX35, 13, CR20, 10, FE15, 12, KW10, 24, MM67, and 4. Locate the btnDisplay_Click procedure and then finish coding it. 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

3. In this exercise, you modify the application from TRY THIS Exercise 1. Use Windows to make a copy of the Commission Solution folder. Save the copy in the ClearlyVB2012\Chap21 folder. Rename the copy Modified Commission Solution. Open the Commission Solution (Commission Solution.sln) file contained in the Modified Commission Solution folder. Open the designer and Code Editor windows. The btnCalc_Click procedure should prompt the user to enter the commission rate. It then should pass the rate to the GetCommission function. Make the appropriate modifications to the code. 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 Medical Bills application coded in Chapter 20. Use Windows to copy the Medical Solution folder from the ClearlyVB2012\Chap20 folder to the ClearlyVB2012\Chap21 folder. Open the Medical Solution (Medical Solution.sln) file contained in the ClearlyVB2012\Chap21\Medical Solution folder. Open the designer and Code Editor windows. Instead of using a two-dimensional array, the code should use a structure and an array of structure variables. The structure should contain a String variable and a Decimal variable. Make the appropriate modifications to the code. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

MODIFY THIS

INTRODUCTORY

5. Open the Test Scores Solution (Test Scores Solution.sln) file contained in the ClearlyVB2012\Chap21\Test Scores Solution folder. Open the Code Editor window. Create a structure that contains three member variables; each member variable will represent a test score, which may contain a decimal place. The btnAverage control's Click event procedure should prompt the user to enter the three test scores. Each test score should be stored in a member of a structure variable. The procedure should pass the structure variable to a function that calculates and returns the average score. Finally, the procedure should display the average score (formatted with one decimal place) in the lblResult control. Code the application. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

6. Open the Colors Solution (Colors Solution.sln) file contained in the ClearlyVB2012\Chap21\Colors Solution folder. Open the Code Editor window. Create a structure that contains two member variables named `strEnglish` and `strSpanish`. Declare a class-level array of structure variables. Use the form's Load event procedure to fill the array with the values shown in Figure 21-13. The btnTranslate control's Click event procedure should prompt the user to enter an English word. It then should display the English word and its corresponding Spanish word. If the English word is not in the array, display "N/A". Code the application. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

English	Spanish
Blue	Azul
Brown	Marron
Gray	Gris
Green	Verde
Orange	Anaranjado
Pink	Rosa
Purple	Morado
Red	Rojo
Yellow	Amarillo

Figure 21-13 Information for Exercise 6

INTERMEDIATE

7. In this exercise, you modify the application from Exercise 5. If you did not complete Exercise 5, you will need to do so before completing this exercise. Use Windows to make a copy of the Test Scores Solution folder. Save the copy in the ClearlyVB2012\Chap21 folder. Rename the copy Modified Test Scores Solution.

   a. Open the Test Scores Solution (Test Scores Solution.sln) file contained in the Modified Test Scores Solution folder. Open the designer window. Change the lblResult control's TextAlign property to MiddleLeft.

   b. Open the Code Editor window. The btnAverage_Click procedure should declare a three-element array of structure variables; each element will contain the test scores for one student. The procedure should prompt the user for a student's three test scores and then store the scores in one of the structure variables in the array. It then should use the function to determine the student's average score. Finally, it should display the student's number (1, 2, or 3) along with the average score (formatted with one decimal place) in the lblResult control. Save the solution and then start and test the application. Figure 21-14 shows a sample run of the application when the user enters the following scores: 100, 100, 100, 90, 85, 78, 73, 72, and 67. Close the Code Editor window and then close the solution.

Sequential Access Files

In addition to getting data from the keyboard and sending data to the computer screen, an application also can get data from and send data to a file on a disk. Getting data from a file is referred to as "reading from the file," and sending data to a file is referred to as "writing to the file." Files to which data is written are called **output files** because the files store the output produced by an application. Files that are read by the computer are called **input files** because an application uses the data in these files as input.

Most input and output files are composed of lines of text that are both read and written sequentially. In other words, they are read and written in consecutive order, one line at a time, beginning with the first line in the file and ending with the last line in the file. Such files are referred to as **sequential access files** because of the manner in which the lines of text are accessed. They are also called **text files** because they are composed of lines of text. Examples of text stored in sequential access files include an employee list, a memo, and a sales report.

You will use a sequential access file in the Game Show Contestants application, which you code in the remaining sections of this chapter. As indicated in Figure 22-1, the application's interface provides a text box for entering a contestant's name. The Write to File button will write the name to a sequential access file. The Read from File button will read each name from the sequential access file and display each in the Contestants box. (The txtContestants control's Multiline and ReadOnly properties are set to True, and its ScrollBars property is set to Vertical.) You will code the Write to File button first.

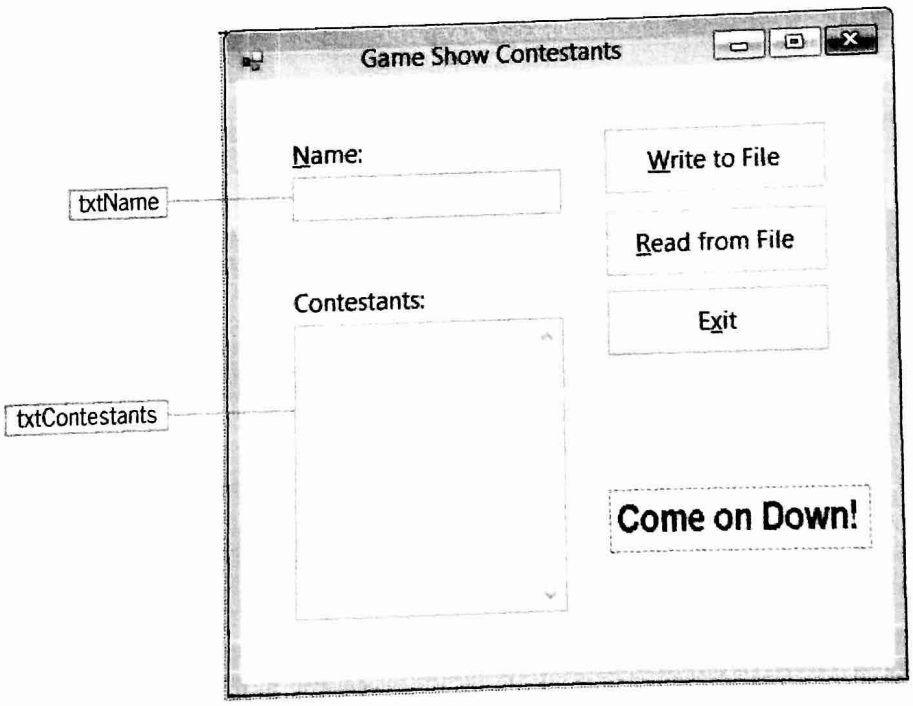

Figure 22-1 Interface for the Game Show Contestants application

Write Those Lines of Text

An item of data—such as the string "Yolanda"—is viewed differently by a human being and a computer. To a human being, the string represents a person's name; to a computer, it is merely a sequence of characters. Programmers refer to a sequence of characters as a **stream of characters**.

In Visual Basic, you use a **StreamWriter object** to write a stream of characters to a sequential access file. Before you create the StreamWriter object, you first declare a variable to store the object in the computer's internal memory. Figure 22-2 shows the syntax and an example of declaring a StreamWriter variable. The IO in the syntax stands for Input/Output.

Declaring a StreamWriter Variable

Syntax
{**Dim** | **Private**} *streamWriterVariableName* **As IO.StreamWriter**

Example
`Dim outFile As IO.StreamWriter`
declares a StreamWriter variable named `outFile`

Figure 22-2 Syntax and an example of declaring a StreamWriter variable

To begin coding the Write to File button's Click event procedure:

1. Start Visual Studio 2012. Open the **Game Show Solution** (**Game Show Solution.sln**) file contained in the ClearlyVB2012\Chap22\Game Show Solution folder. If necessary, open the designer window.
2. Open the Code Editor window. Locate the btnWrite_Click procedure. Click the **blank line** below the `' declare a StreamWriter variable` comment and then enter the following declaration statement:

   Dim outFile As IO.StreamWriter

After declaring a StreamWriter variable, you can use the syntax shown in Figure 22-3 to create a StreamWriter object. As the figure indicates, creating a StreamWriter object involves opening a sequential access file using either the CreateText method or the AppendText method. You use the **CreateText method** to open a sequential access file for output. When you open a file for output, the computer creates a new, empty file to which data can be written. If the file already exists, the computer erases the contents of the file before writing any data to it. You use the **AppendText method** to open a sequential access file for append. When a file is opened for append, new data is written after any existing data in the file. If the file does not exist, the computer creates the file for you. In addition to opening the file, both methods automatically create a StreamWriter object to represent the file in the application. You assign the StreamWriter object to a StreamWriter variable, which you use to refer to the file in code. Figure 22-3 also includes examples of using both methods.

CreateText and AppendText Methods

Syntax
IO.File.*method*(*fileName*)

method	Description
CreateText	opens a sequential access file for output
AppendText	opens a sequential access file for append

Example 1
`outFile = IO.File.CreateText("memo.txt")`
opens the memo.txt file for output; creates a StreamWriter object and assigns it to the `outFile` variable

Example 2
`outFile = IO.File.AppendText("F:\Chap22\pay.txt")`
opens the pay.txt file for append; creates a StreamWriter object and assigns it to the `outFile` variable

Figure 22-3 Syntax and examples of the CreateText and AppendText methods

When processing the statement in Example 1, the computer searches for the memo.txt file in the default folder, which is the current project's bin\Debug folder. If the file exists, its contents are erased and the file is opened for output; otherwise, a new, empty file is created and opened for output. The statement then creates a StreamWriter object and assigns it to the `outFile` variable.

Unlike the *fileName* argument in Example 1, the *fileName* argument in Example 2 contains a folder path. When processing the statement in Example 2, the computer searches for the pay.txt file in the Chap22 folder on the F drive. If the computer locates the file, it opens the file for append. If it does not find the file, it creates a new, empty file and then opens the file for append. Like the statement in Example 1, the statement in Example 2 creates a StreamWriter object and assigns it to the `outFile` variable. When deciding whether to include the folder path in the *fileName* argument, keep in mind that a USB drive may have a different letter designation on another computer. Therefore, you should specify the folder path only when you are sure that it will not change.

When the user clicks the Write to File button in the Game Show Contestants interface, the name entered in the Name box should be added to the end of the existing names in the file. Therefore, you will need to open the sequential access file for append. A descriptive name for a file that stores the names of contestants is contestants.txt. Although it is not a requirement, the "txt" (short for "text") filename extension is commonly used when naming sequential access files; this is because the files contain text.

To continue coding the btnWrite_Click procedure:

1. Click the **blank line** below the `' open the file for append` comment and then enter the following assignment statement:

   outFile = IO.File.AppendText("contestants.txt")

After opening a file for either output or append, you can begin writing data to it using either the **Write method** or the **WriteLine method**. The difference between both methods is that the WriteLine method writes a newline character after the data. Figure 22-4 shows the syntax and an example of both methods. As the figure indicates, when using the Write method, the next character written to the file will appear immediately after the letter o in the string "Hello". When using the WriteLine method, however, the next character written to the file will appear on the line immediately below the string. You do not need to include the file's name in either method's syntax because the data will be written to the file associated with the StreamWriter variable.

Write and WriteLine Methods

Syntax
streamWriterVariableName.**Write**(*data*)
streamWriterVariableName.**WriteLine**(*data*)

Example 1
```
outFile.Write("Hello")
```
Result
Hello|
the next character will appear immediately after the o

Example 2
```
outFile.WriteLine("Hello")
```
Result
Hello
|
the next character will appear on the next line

Figure 22-4 Syntax and examples of the Write and WriteLine methods

Each contestant's name should appear on a separate line in the file, so you will use the WriteLine method to write each name to the file.

To continue coding the btnWrite_Click procedure:

1. Click the blank line below the `' write the name on a separate line in the file` comment and then enter the following statement:

   outFile.WriteLine(txtName.Text)

You should use the **Close method** to close an output sequential access file as soon as you are finished using it. This ensures that the data is saved and it makes the file available for use elsewhere in the application. The syntax to close an output sequential access file is *streamWriterVariableName*`.Close()`. Here again, notice that you use the StreamWriter variable to refer to the file you want to close.

To continue coding the btnWrite_Click procedure:

1. Click the **blank line** below the `' close the file` comment and then enter the following statement:

   outFile.Close()

To make it more convenient for the user to enter the next name, you will clear the current name from the Name box and then send the focus to the box. You can use a text box's **Focus method** to send the focus to the text box. The method's syntax is *textbox*`.Focus()`.

To complete the btnWrite_Click procedure:

1. Click the **blank line** below the `' clear the Name box and then set the focus` comment. Enter the following two statements:

   txtName.Text = String.Empty
   txtName.Focus()

Figure 22-5 shows the code entered in the Write to File button's Click event procedure.

```
Private Sub btnWrite_Click(sender As Object,
e As EventArgs) Handles btnWrite.Click
    ' writes a name to a sequential access file

    ' declare a StreamWriter variable
    Dim outFile As IO.StreamWriter

    ' open the file for append
    outFile = IO.File.AppendText("contestants.txt")

    ' write the name on a separate line in the file
    outFile.WriteLine(txtName.Text)

    ' close the file
    outFile.Close()

    ' clear the Name box and then set the focus
    txtName.Text = String.Empty
    txtName.Focus()

End Sub
```

Figure 22-5 ... File button's Click event procedure

To test the btnWrite_Click procedure:

1. Save the solution and then start the application. Type **Suzanne Barston** in the Name box and then click the **Write to File** button. Use the application to write the following four names to the file:

   Aiden Jones
   Treyson Mills
   Addison Tennison
   Sydney Poulos

2. Click the **Exit** button. Now you will open the contestants.txt file to verify its contents. Click **FILE** on the menu bar and then click **Open File**. Open the project's bin\Debug folder. Click **contestants.txt** in the list of filenames and then click the **Open** button. The contestants.txt window opens and shows the five names contained in the file. See Figure 22-6.

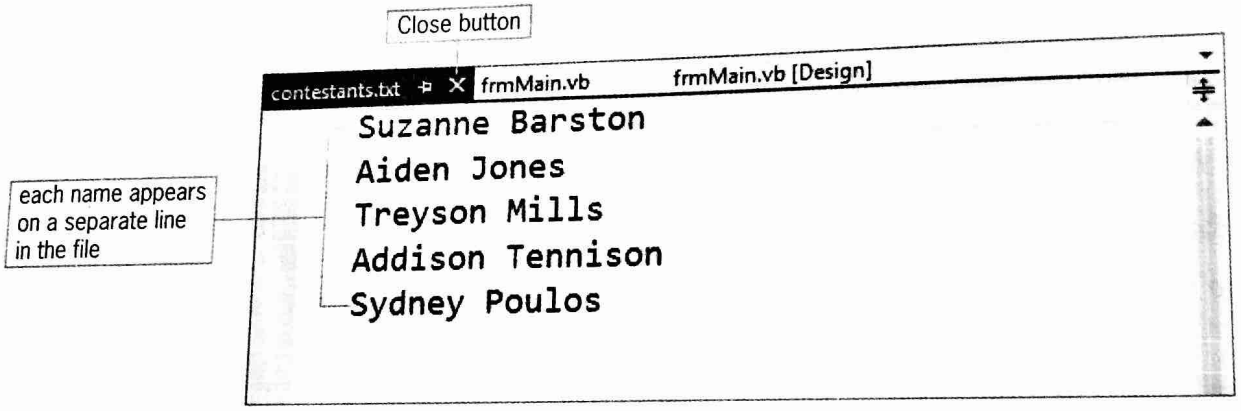

Figure 22-6 Names contained in the contestants.txt file

3. Close the contestants.txt window by clicking its **Close** button.

Mini-Quiz 22-1

See Appendix B for the answers.

1. Write the code to declare a variable that can be used to write data to a sequential access file. Name the variable `outFile`.
2. The AppendText method creates a(n) _____ object.
3. Write a Visual Basic statement that sends the focus to the txtCity control.

Now Read Those Lines of Text

Next, you will code the Read from File button. In Visual Basic, you use a **StreamReader obje** to read data from a sequential access file. Before creating the StreamReader object, you first declare a variable to store the object in the computer's internal memory. Figure 22-7 shows t syntax and an example of declaring a StreamReader variable. As mentioned earlier, the IO in t syntax stands for Input/Output.

Declaring a StreamReader Variable

Syntax
{Dim | Private} *streamReaderVariableName* **As IO.StreamReader**

Example
`Dim inFile As IO.StreamReader`
declares a StreamReader variable named `inFile`

Figure 22-7 Syntax and an example of declaring a StreamReader variable

To begin coding the Read from File button's Click event procedure:

1. Locate the btnRead_Click procedure. Click the **blank line** below the `' declare variables` comment and then enter the following declaration statement:

   Dim inFile As IO.StreamReader

After declaring a StreamReader variable, you can use the **OpenText method** to open a sequential access file for input, which will automatically create a StreamReader object. When a file is opened for input, the computer can read the lines of text stored in the file. Figure 22-8 shows the OpenText method's syntax along with an example of using the method. The *fileName* argument in the example does not include a folder path, so the computer will search for the memo.txt file in the default folder, which is the current project's bin\Debug folder. If the computer finds the file, it opens the file for input. If the computer does not find the file, a run time error occurs. You assign the StreamReader object created by the OpenText method to a StreamReader variable, which you use to refer to the file in code.

OpenText Method

Syntax
IO.File.OpenText(*fileName***)**

Example
`inFile = IO.File.OpenText("memo.txt")`
opens the memo.txt file for input; creates a StreamReader object and assigns it to the `inFile` variable

Figure 22-8 Syntax and an example of the OpenText method

The runtime error that occurs when the computer cannot locate the file you want opened for input will cause the application to end abruptly. You can use the Exists method to avoid this run time error. Figure 22-9 shows the method's syntax and includes an example of using the method. If the *fileName* argument does not include a folder path, the computer searches for the file in the current project's bin\Debug folder. The **Exists method** returns the Boolean value True if the file exists; otherwise, it returns the Boolean value False.

Exists Method

Syntax
IO.File.Exists(*fileName***)**

Example
`If IO.File.Exists("memo.txt") = True Then`
determines whether the memo.txt file exists in the current project's bin\Debug folder; you can also write the If clause as `If IO.File.Exists("memo.txt") Then`

Figure 22-9 Syntax and an example of the Exists method

To continue coding the btnRead_Click procedure:

1. Click the **blank line** below the `' determine whether the file exists` comment and then enter the following If clause:

   If IO.File.Exists("contestants.txt") = True Then

2. If the file exists, you will use the OpenText method to open the file. Enter the following comment and assignment statement. Press **Enter** twice after typing the assignment statement.

   ' open the file for input
   inFile = IO.File.OpenText("contestants.txt")

3. If the file does not exist, you will display an appropriate message. Enter the additional lines of code shown in Figure 22-10.

enter these five lines of code

```
' determine whether the file exists
If IO.File.Exists("contestants.txt") = True Then
    ' open the file for input
    inFile = IO.File.OpenText("contestants.txt")

Else
    MessageBox.Show("Can't find the file",
                    "Game Show Contestants",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information)
End If
```

Figure 22-10 Code entered in the selection structure's false path

After opening a file for input, you can use the **ReadLine method** to read the file's contents, one line at a time. A **line** is defined as a sequence (stream) of characters followed by the newline character. The ReadLine method returns a string that contains only the sequence of characters in the current line. The returned string does not include the newline character at the end of the line. In most cases, you assign the string returned by the ReadLine method to a String variable. Figure 22-11 shows the ReadLine method's syntax and includes an example of using the method. The ReadLine method does not require you to provide the file's name because it uses the file associated with the StreamReader variable.

ReadLine Method

Syntax
streamReaderVariableName.**ReadLine**

Example
```
Dim strMessage As String
strMessage = inFile.ReadLine
```
reads a line of text from the sequential access file associated with the `inFile` variable and assigns the line, excluding the newline character, to the `strMessage` variable

Figure 22-11 Syntax and an example of the ReadLine method

In most cases, an application will need to read each line of text contained in a sequential access file, one line at a time. You can do this using a loop along with the Peek method. The **Peek method** "peeks" into the file to determine whether the file contains another character to read. If the file contains another character, the Peek method returns the character; otherwise, it returns the number −1 (a negative 1). The Peek method's syntax is shown in Figure 22-12 along with an example of using the method. The Do clause in the example tells the computer to process the loop instructions until the Peek method returns the number −1, which indicates that there are no more characters to read. In other words, the Do clause tells the computer to process the loop instructions until the end of the file is reached.

Peek Method

Syntax
streamReaderVariableName.**Peek**

Example
```
Dim strLineOfText As String
Do Until inFile.Peek = -1
    strLineOfText = inFile.ReadLine
    MessageBox.Show(strLineOfText)
Loop
```
reads each line of text from the sequential access file associated with the `inFile` variable, line by line; each line (excluding the newline character) is assigned to the `strLineOfText` variable and is then displayed in a message box

Figure 22-12 Syntax and an example of the Peek method

To continue coding the btnRead_Click procedure:

1. First, you will declare a variable to store the string returned by the ReadLine method. Each line in the contestants.txt file represents a name, so you will call the variable `strName`. Click the **blank line** below the Dim statement and then enter the following declaration statement:

   Dim strName As String

2. The Do clause is next. Click the **blank line** above the Else clause and then enter the following comment and Do clause, being sure to type the minus sign before the number 1:

   ' process loop instructions until end of file
   Do Until inFile.Peek = -1

3. The first instruction in the loop should read a line of text and assign it (excluding the newline character) to the `strName` variable. Enter the following comment and assignment statement:

```
' read a name
strName = inFile.ReadLine
```


4. Now, you will display the name in the Contestants box. Enter the following comment and assignment statement:

```
' display the name
txtContestants.Text = txtContestants.Text &
    strName & ControlChars.NewLine
```

5. If necessary, delete the blank line above the Loop clause.

Just as you do with an output sequential access file, you should use the Close method to close an input sequential access file as soon as you are finished using it. Doing this makes the file available for use elsewhere in the application. The syntax to close an input sequential access file is *streamReaderVariableName*`.Close()`. Notice that you use the StreamReader variable to refer to the file you want to close.

To finish coding the btnRead_Click procedure:

1. Click **after the letter p** in the Loop clause and then press **Enter** to insert a blank line.
2. Enter the following comment and statement:

```
' close the file
inFile.Close()
```

Figure 22-13 shows the code entered in the Read from File button's Click event procedure.

```
Private Sub btnRead_Click(sender As Object,
e As EventArgs) Handles btnRead.Click
    ' reads names from a sequential access file
    ' and displays them in the interface

    ' declare variables
    Dim inFile As IO.StreamReader
    Dim strName As String

    ' clear previous names from the Contestants box
    txtContestants.Text = String.Empty

    ' determine whether the file exists
    If IO.File.Exists("contestants.txt") = True Then
        ' open the file for input
        inFile = IO.File.OpenText("contestants.txt")
        ' process loop instructions until end of file
        Do Until inFile.Peek = -1
            ' read a name
            strName = inFile.ReadLine
            ' display the name
            txtContestants.Text = txtContestants.Text &
                strName & ControlChars.NewLine
        Loop
        ' close the file
        inFile.Close()
```

Figure 22-13 Read from File button's Click event procedure *(continues)*

(continued)

```
    Else
        MessageBox.Show("Can't find the file",
                        "Game Show Contestants",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information)
    End If
End Sub
```

Figure 22-13 Read from File button's Click event procedure

To test the application's code:

1. Save the solution and then start the application. Click the **Read from File** button. The five names contained in the contestants.txt file appear in the Contestants box, as shown in Figure 22-14. (Recall that you can use the Alt key to show/hide the access keys.)

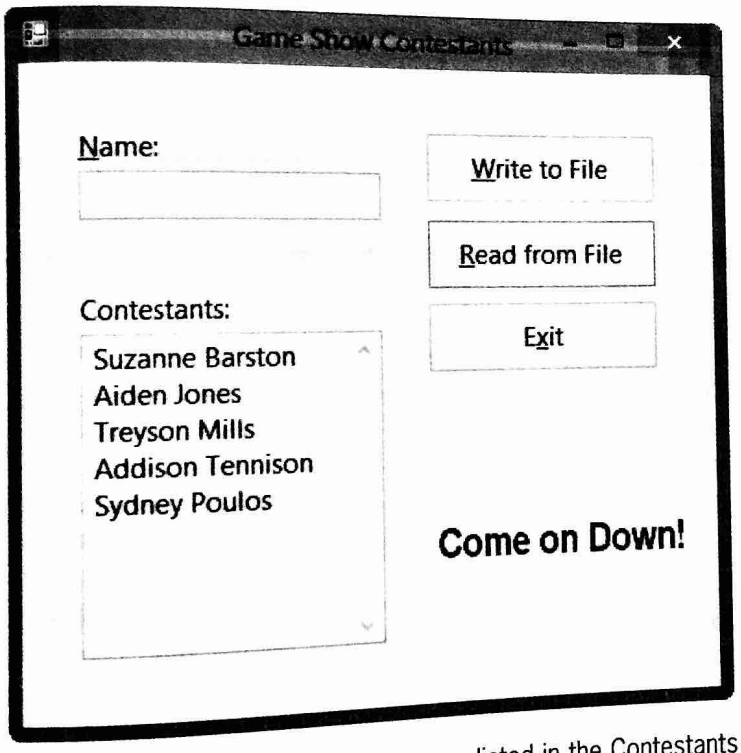

Figure 22-14 Five contestant names listed in the Contestants box

2. On your own, add the following two names to the file:

   Kevin Little
   Darlene Aronson

3. Click the **Read from File** button to display the seven names in the Contestants box, and then click the **Exit** button.
4. Next, you will modify the If clause in the btnRead_Click procedure. More specifically, you will change the Exists method's filename from contestants.txt to contestant.txt. Doing this will allow you to test the code entered in the selection structure's false path. Change `contestants.txt` in the If clause to **contestant.txt**.

5. Save the solution and then start the application. Click the **Read from File** button. Because the contestant.txt file does not exist, the Exists method in the If clause returns the Boolean value False. As a result, the instruction in the selection structure's false path displays the "Can't find the file" message in a message box. Close the message box and then click the **Exit** button.
6. Change `contestant.txt` in the If clause to **contestants.txt**. Save the solution and then start the application. Click the **Read from File** button, which displays the seven names in the Contestants box.
7. Click the **Exit** button. Close the Code Editor window and then close the solution.

To learn more about sequential access files, see the Sequential Access Files section in the Ch22WantMore.pdf file.

Mini-Quiz 22-2

See Appendix B for the answers.

1. Write the code to declare a variable that can be used to read data from a sequential access file. Name the variable `inFile`.
2. The OpenText method creates a(n) _____ object.
3. The string returned by the ReadLine method contains the newline character.

   a. True b. False

To learn how to handle exceptions that occur when using sequential access files, view the Ch22-TryCatch video.

Summary

- An application can write data to a sequential access file. It also can read data from the file. The data in a sequential access file is always accessed in consecutive order (sequentially) from the beginning of the file through the end of the file.
- You use a StreamWriter object to write a sequence (stream) of characters to a sequential access file. The StreamWriter object is created when you open a file for either output or append. You use a StreamReader object to read a sequence (stream) of characters from a sequential access file. The StreamReader object is created when you open a file for input.
- You can use either the Write method or the WriteLine method to write data to a sequential access file. You use the ReadLine method to read a line of text from a sequential access file. The ReadLine method returns a string that includes only the characters in the current line; it does not include the newline character at the end of the line.
- You should use the Close method to close a sequential access file as soon as you are finished using the file.
- You can use a text box's Focus method to send the focus to the text box.
- The Exists method returns a Boolean value that indicates whether a sequential access file exists.
- If a file contains another character to read, the Peek method returns the character; otherwise, it returns the number –1.

Exercises

1. Open the Gross Pay Solution (Gross Pay Solution.sln) file contained in the ClearlyVB2012\Chap22\Gross Pay Solution folder. If necessary, open the designer window. The interface provides a text box for entering a gross pay amount. The Save button should write the gross pay amount to a sequential access file named gross.txt. Save the file in the project's bin\Debug folder. Display an appropriate message if the txtGrossPay control is empty. The Display button should read the gross pay amounts from the gross.txt file and display each (formatted with a dollar sign and two decimal places) in the interface. Open the Code Editor window. Code the Click event procedures for the btnSave and btnDisplay controls. Save the solution and then start the application. Write the following 10 gross pay amounts to the file: 400, 763, 957.75, 235.72, 679.89, 250, 700, 548, 1100, and 1500.67. Click the Display button to display the gross pay amounts in the interface. Close the Code Editor window and then close the solution. (See Appendix B for the answer.) TRY THIS

2. Open the Name Solution (Name Solution.sln) file contained in the ClearlyVB2012\Chap22\Name Solution folder. If necessary, open the designer window. Open the Code Editor window. Open the names.txt file contained in the project's bin\Debug folder. The sequential access file contains five names. Close the names.txt window. The Display button's Click event procedure should read the five names contained in the names.txt file, storing each in a five-element one-dimensional array. The procedure should sort the array in ascending order and then display the contents of the array in the lblFriends control. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. If you need to recreate the names.txt file, open the file in a window in the IDE. Delete the contents of the file and then type the following five names, pressing Enter after typing each name: Jacob, Zachary, Alicia, Benjamin, and Kate. (See Appendix B for the answer.) TRY THIS

3. Open the Salary Solution (Salary Solution.sln) file contained in the ClearlyVB2012\Chap22\Salary Solution folder. If necessary, open the designer window. Open the Code Editor window and study the existing code. The code stores six salary amounts in a one-dimensional Integer array named `intSalaries`. Each salary amount corresponds to a salary code from 1 through 6. Code 1's salary is stored in the `intSalaries(0)` element in the array, code 2's salary is stored in the `intSalaries(1)` element, and so on. The btnDisplay_Click procedure prompts the user to enter a salary code. It then displays the amount associated with the code. Currently, the Private statement assigns the six salary amounts to the array. Modify the code so that the form's Load event procedure reads the salary amounts from the salary.txt file and stores each in the array. The salary.txt file is contained in the project's bin\Debug folder. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. MODIFY THIS

4. Open the CD Solution (CD Solution.sln) file contained in the ClearlyVB2012\Chap22\CD Solution—Introductory folder. The interface provides a text box for entering the name of a CD. The Save CD button's Click event procedure should write the CD name to a sequential access file named cds.txt. Save the file in the project's bin\Debug folder. The Display CDs button's Click event procedure should read the CD names from the cds.txt file and display each in the interface. Open the Code Editor window and code both procedures. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. INTRODUCTORY

5. Open the Test Scores Solution (Test Scores Solution.sln) file contained in the ClearlyVB2012\Chap22\Test Scores Solution folder. If necessary, open the designer window. Open the Code Editor window. The btnSave control's Click event procedure should allow the user to enter an unknown number of test scores, saving each score in a sequential access file. The btnCount control's Click event procedure should display (in a message box) the number of scores stored in the file. Code both procedures. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. INTRODUCTORY