7

Arrays

7.1 Creating and Accessing Arrays

A variable (or simple variable) is a name to which Visual Basic can assign a single value. An **array variable** is an indexed list of simple variables of the same type, to and from which Visual Basic can efficiently assign and access a list of values.

Consider the following situation: Suppose you want to evaluate the exam grades for 30 students. Not only do you want to compute the average score, but you also want to display the names of the students whose grades are above average. You might run the program outlined below. ***Note:*** Visual Basic prefers zero-based numberings—that is, numberings beginning with 0 instead of 1. (We saw this preference in Chapter 3, where the numbering of the positions in a string began with 0.) Therefore we will number the 30 student names and grades from 0 through 29 instead of from 1 through 30.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim student0 As String, grade0 As Double
  Dim student1 As String, grade1 As Double
  .
  .
  Dim student29 As String, grade29 As Double
  'Analyze exam grades
  Dim promptName As String = "Enter name of student #"
  Dim promptGrade As String = "Enter grade for student #"
  student0 = InputBox(promptName & 0, "Name")
  grade0 = CDbl(InputBox(promptGrade & 0, "Grade"))
  student1 = InputBox(promptName & 1, "Name")
  grade1 = CDbl(InputBox(promptGrade & 1, "Grade"))
  .
  .
  student29 = InputBox(promptName & 29, "Name")
  grade29 = CDbl(InputBox(promptGrade & 29, "Grade"))
  'Compute the average grade
  .
  .
  'Display the names of students with above-average grades
  .
  .
End Sub
```

This program is going to be uncomfortably long. What's most frustrating is that the 30 Dim statements and 30 pairs of statements obtaining input are very similar and look as if they should be condensed into a loop. A shorthand notation for the many related variables would be welcome. It would be nice if we could just write

```
For i As Integer = 0 To 29
  studenti = InputBox(promptName & i, "Name")
  gradei = CDbl(InputBox(promptGrade & i, "Grade"))
Next
```

Of course, this will not work. Visual Basic will treat *studenti* and *gradei* as two variables and keep reassigning new values to them. At the end of the loop, they will have the values of the thirtieth student.

Declaring an Array Variable

Visual Basic provides a data structure called an **array** that lets us do what we tried to accomplish in the loop above. The variable names, similar to those in the loop, will be

```
students(0), students(1), students(2), students(3), ..., students(29)
```

and

```
grades(0), grades(1), grades(2), grades(3), ..., grades(29)
```

We refer to these collections of variables as the array variables *students* and *grades*. The numbers inside the parentheses of the individual variables are called **subscripts** or **indexes**, and each individual variable is called a **subscripted variable** or **element**. For instance, *students*(3) is the fourth element of the array *students*, and *grades*(20) is the twenty-first element of the array *grades*. The elements of an array are located in successive memory locations. Figure 7.1 shows the memory locations for the array *grades*.

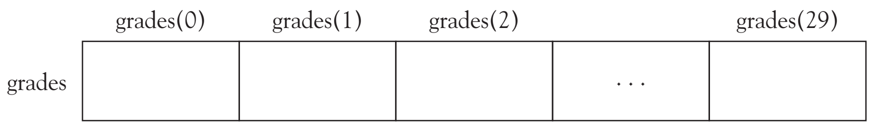


FIGURE 7.1 **The array *grades*.**

Names of array variables follow the same naming conventions as simple variables. If *arrayName* is the name of an array variable and *n* is a literal, variable, or expression of type Integer, then the declaration statement

```
Dim arrayName(n) As DataType
```

reserves space in memory to hold the values of the subscripted variables *arrayName*(0), *arrayName*(1), *arrayName*(2), . . . , *arrayName*(*n*). The value of *n* is called the **upper bound** of the array. The number of elements in the array, *n* + 1, is called the **size** of the array. The subscripted variables will all have the same data type—namely, the type specified by *DataType*. For instance, they could all be variables of type String or all be variables of type Double. In particular, the statements

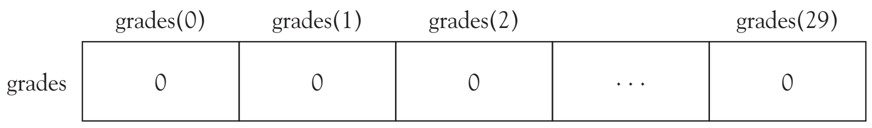
```
Dim students(29) As String
Dim grades(29) As Double
```

declare the 30-element arrays needed for the preceding program.

Values can be assigned to individual subscripted variables with assignment statements and displayed in text boxes and list boxes just like values of ordinary variables. The default initial value of each subscripted variable is the same as with an ordinary variable—that is, the keyword Nothing for String types and 0 for numeric types. The statement

```
Dim grades(29) As Double
```

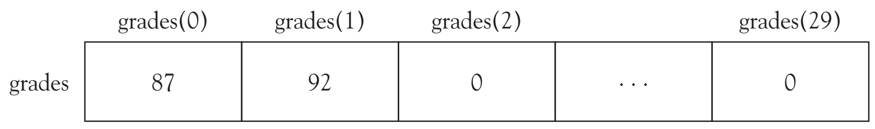
sets aside a portion of memory for the array *grades* and assigns the default value 0 to each element.



The statements

```
grades(0) = 87
grades(1) = 92
```

assign values to the first two elements of the array.



The statements

```
For i As Integer = 0 To 2
  lstBox.Items.Add(grades(i))
Next
```

then produce the following output in the list box:

```
87
92
0
```

As with an ordinary variable, an array declared in the Declarations section of the Code Editor is class-level. That is, it will be visible to all procedures in the form, and any values assigned to it in a procedure will persist after the procedure terminates. Array variables declared inside a procedure are local to that procedure and cease to exist when the procedure is exited.

Example 1 The following program creates a string array consisting of the names of the first four Super Bowl winners. Figure 7.2 shows the array created by the program.

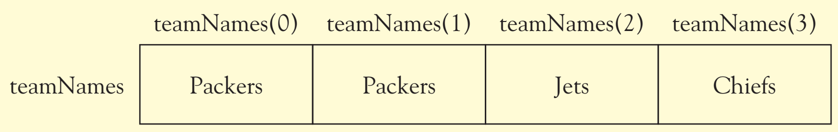
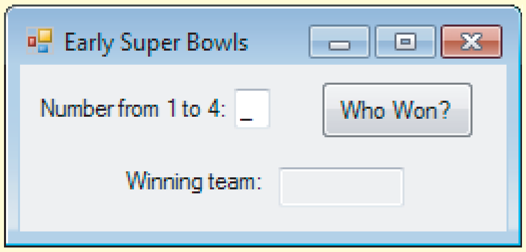


FIGURE 7.2 **The array *teamNames* of Example 1.**

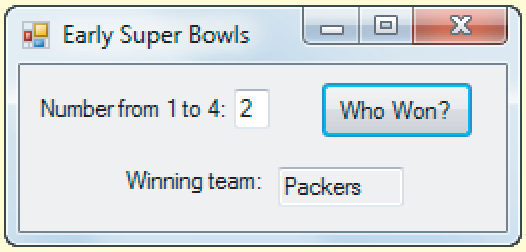


OBJECT	PROPERTY	SETTING
frmBowl	Text	Early Super Bowls
lblNumber	Text	Number from 1 to 4:
mtbNumber	Mask	0
btnWhoWon	Text	Who Won?
lblWinner	Text	Winning team:
txtWinner	ReadOnly	True

```
Private Sub btnWhoWon_Click(...) Handles btnWhoWon.Click
  Dim teamNames(3) As String
  Dim n As Integer
  'Place Super Bowl Winners into the array
```

```
    teamNames(0) = "Packers"
    teamNames(1) = "Packers"
    teamNames(2) = "Jets"
    teamNames(3) = "Chiefs"
    'Access array
    n = CInt(mtbNumber.Text)
    txtWinner.Text = teamNames(n - 1)
  End Sub
```

[Run, type 2 into the masked text box, and click on the button.]



The Load Event Procedure

VideoNote
Filling arrays

In Example 1, the array *teamNames* was assigned values in the btnWhoWon_Click event procedure. Every time the button is clicked, the values are reassigned to the array. This approach can be very inefficient, especially in programs with large arrays, where the task of the program (in Example 1, looking up a fact) may be repeated numerous times for different user input. When, as in Example 1, the data to be placed in an array are known at the time the program begins to run, a more efficient location for the statements that fill the array is in the form's Load event procedure. A form's Load event occurs just before the form is displayed to the user. It is the default event for the form. The header for the Load event procedure is

```
Private Sub frmName_Load(...) Handles MyBase.Load
```

The keyword MyBase is similar to the Me keyword and refers to the form. Example 2 uses the frmBowl_Load procedure to improve Example 1.

Example 2 The following variation of Example 1 makes *teamNames* a class-level array and assigns values to the elements of the array in the event procedure frmBowl_Load.

```
Dim teamNames(3) As String

Private Sub frmBowl_Load(...) Handles MyBase.Load
  'Place Super Bowl Winners into the array
  teamNames(0) = "Packers"
  teamNames(1) = "Packers"
  teamNames(2) = "Jets"
  teamNames(3) = "Chiefs"
End Sub

Private Sub btnWhoWon_Click(...) Handles btnWhoWon.Click
  Dim n As Integer
  n = CInt(mtbNumber.Text)
  txtWinner.Text = teamNames(n - 1)
End Sub
```

Implicit Array Sizing and Initialization

Like ordinary variables, array variables can be assigned initial values when they are declared. A statement of the form

```
Dim arrayName() As DataType = {value0, value1, value2, ..., valueN}
```

declares an array having upper bound *N* and assigns *value0* to *arrayName*(0), *value1* to *arrayName*(1), *value2* to *arrayName*(2), . . . , and *valueN* to *arrayName*(*N*). For instance, in Example 2, the Dim statement and frmBowl_Load event procedure can be replaced by the single line

```
Dim teamNames() As String = {"Packers", "Packers", "Jets", "Chiefs"}
```

Note: You cannot use a list of values in braces to fill an array if an upper bound has been specified for the array. For instance, the following line of code is not valid:

```
Dim teamNames(3) As String = {"Packers", "Packers", "Jets", "Chiefs"}
```

Text Files

The two methods we have used to fill an array are fine for small arrays. However, in practice arrays can be quite large. One way to fill a large array is to use a simple data file known as a **text file**. Text files can be created, viewed, and modified with sophisticated word processors such as Word, or with elementary word processors such as the Windows accessories WordPad and Notepad. They differ from files normally created with Word in that they have no formatting (such as line spacing and font style). They are pure text and nothing else—hence the name *text file*. For instance, a text file that could be used to fill the array in Example 1 would look as follows:

```
Packers
Packers
Jets
Chiefs
```

The text files needed for exercises in this book have been created for you and are in the material you downloaded from the companion website. They are contained in a subfolder (named Text_Files_for_Exercises) of the appropriate chapter folder. Each text file ends with the extension ".txt".

The Visual Basic IDE provides simple ways to create and edit text files. The details can be found in Appendix B. Chapter 8 shows how to create text files programmatically.

A statement of the form

```
Dim strArrayName() As String = IO.File.ReadAllLines(filespec)
```

where *filespec* refers to a text file, declares a string array whose size equals the number of lines in the file, and fills it with the contents of the file. A numeric array can be filled with a text file by first filling a temporary string array with the file and then using a loop, along with CInt or CDbl, to transfer the numbers into the numeric array. (See Example 3.) **Note:** In Section 7.2, we present a way to fill a numeric array with the contents of a numeric text file without using a loop.

The Solution Explorer window has the name of the program as its first line. If only a few entries appear in the Solution Explorer, you can click on the *Show All Files* button (📄) at the top of the Solution Explorer window to display all the files and subfolders. One subfolder is named *bin*. The folder *bin* has a subfolder named *Debug*. If the *filespec* above consists only of a filename (that is, if no path is given), Visual Basic will look for the file in the *Debug* subfolder of the program's *bin* folder. **Throughout this book, we assume that every text file accessed by a program is located in the program's *bin*\*Debug* folder.** Every program downloaded from the companion website has this feature. When you write a program that uses one of the text files from a Text_Files_for_Exercises folder, you should use Windows Explorer to place a copy of the text file into the program's *bin**Debug* folder.

Array Methods

VideoNote Array methods

Both numeric and string arrays have the Count, Max, Min, First, and Last methods.[1] The value of *arrayName*.Count is the size of the array, *arrayName*.Max is the highest value (alphabetically or numerically), *arrayName*.Min is the lowest value, *arrayName*.First is the first element of the array, and *arrayName*.Last is the last element. ***Note:*** The upper bound of the array is *arrayName*.Count – 1. Table 7.1 shows some values with the array from Example 1.

TABLE 7.1 Some values from Example 1.

Expression	Value
`teamNames.Count`	4
`teamNames.Max`	Packers
`teamNames.Min`	Chiefs
`teamNames.First`	Packers
`teamNames.Last`	Chiefs

When working with numeric arrays, we often also want to compute the average and total values for the elements. The average value is given by *arrayName*.Average and the total value by *arrayName*.Sum. The program that follows illustrates the use of array methods for a numeric array.

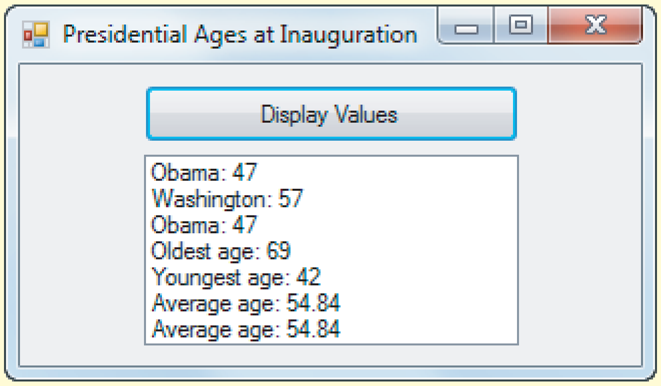
Example 3 The file AgesAtInaugural.txt gives the ages at inauguration of the 44 U.S. presidents. The first four lines contain the data 57, 61, 57, 57—the ages of Washington, Adams, Jefferson, and Madison at their inaugurations. (To see the contents of the file in a text editor, locate the file in the *bin**Debug* folder of the Solution Explorer and double-click on the file. You can remove the text editor by clicking the × symbol on its tab.)

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim ages(43) As Integer
  Dim temp() As String = IO.File.ReadAllLines("AgesAtInaugural.txt")
  For i As Integer = 0 To 43
    ages(i) = CInt(temp(i))
  Next
  lstValues.Items.Add("Obama: " & ages(ages.Count – 1))
  lstValues.Items.Add("Washington: " & ages.First)
  lstValues.Items.Add("Obama: " & ages.Last)
  lstValues.Items.Add("Oldest age: " & ages.Max)
  lstValues.Items.Add("Youngest age: " & ages.Min)
  lstValues.Items.Add("Average age: " & FormatNumber(ages.Average))
```

[1]The property Length can be used instead of the Count method. We favor Count, since it also can be used with LINQ queries.

```
  lstValues.Items.Add("Average age: " & FormatNumber(ages.Sum / ages.Count))
End Sub
```

[Run, and click on the button.]



```
Obama: 47
Washington: 57
Obama: 47
Oldest age: 69
Youngest age: 42
Average age: 54.84
Average age: 54.84
```

Calculating an Array Value with a Loop

Some of the values discussed above also can be calculated with loops. The following example shows how the value returned by the Max method can be calculated with a For . . . Next loop. Some other values returned by methods are calculated with loops in the exercise set.

Example 4 Consider the array consisting of the ages at inauguration of the last nine presidents. To find the maximum age, we temporarily take the age of the first of the nine presidents as the maximum and then adjust the maximum, if required, after looking at each successive age.

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  'Calculate the maximum age at inauguration for the last 9 presidents
  Dim ages() As Integer = {55, 56, 61, 52, 69, 64, 46, 54, 47}
  Dim max As Integer = ages(0)
  For i As Integer = 1 To ages.Count – 1
    If ages(i) > max Then
      max = ages(i)
    End If
  Next
  txtOutput.Text = "The greatest age is " & max & "."
End Sub
```

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

```
The greatest age is 69.
```

The ReDim Statement

After an array has been declared, its size (but not its type) can be changed with a statement of the form

```
ReDim arrayName(m)
```

where *arrayName* is the name of the already declared array and *m* is an Integer literal, variable, or expression. ***Note:*** Since the type cannot be changed, there is no need for an "As *DataType*" clause at the end of a ReDim statement.

Visual Basic allows you to declare an array without specifying an upper bound with a statement of the form

```
Dim arrayName() As DataType
```

Later, the size of the array can be specified with a ReDim statement. (No values can be assigned to the elements of the array until a size is specified.)

The ReDim statement has one shortcoming: It causes the array to lose its current contents. That is, it resets all string values to Nothing and resets all numeric values to 0. This situation can be remedied by following ReDim with the keyword Preserve. The general form of a ReDim Preserve statement is

```
ReDim Preserve arrayName(m)
```

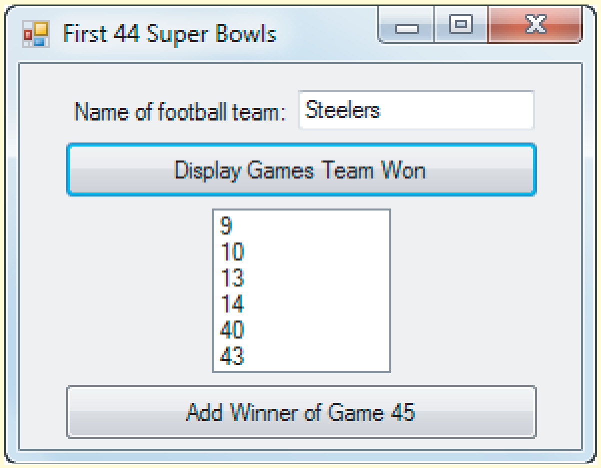
Of course, if you make an array smaller than it was, data at the end of the array will be lost.

Example 5 The following program reads the names of the winners of the first 44 Super Bowl games from a text file and places them into an array. The user can type a team's name into a text box and then display the numbers of the Super Bowl games won by that team. The user has the option of adding winners of subsequent games to the array of winners. The program uses the file SBWinners.txt, whose lines contain the names of the winners in order. That is, the first four lines of the file contain the names Packers, Packers, Jets, and Chiefs.

```
Dim teamNames() As String
Dim numGames As Integer

Private Sub frmBowl_Load(...) Handles MyBase.Load
  teamNames = IO.File.ReadAllLines("SBWinners.txt")
  numGames = teamNames.Count
  'Note: "Me" refers to the form
  Me.Text = "First " & numGames & " Super Bowls"
  'Specify the caption of the Add Winner button
  btnAddWinner.Text = "Add Winner of Game " & (numGames + 1)
End Sub

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Display the numbers of the games won by the team in the text box
  Dim noWins As Boolean = True   'Flag to detect if any wins
  lstGamesWon.Items.Clear()
  For i As Integer = 0 To numGames - 1
    If teamNames(i).ToUpper = txtName.Text.ToUpper Then
      lstGamesWon.Items.Add(i + 1)
      noWins = False
    End If
  Next
  If noWins Then
    lstGamesWon.Items.Add("No Games Won")
  End If
End Sub

Private Sub btnAddWinner_Click(...) Handles btnAddWinner.Click
  'Add winner of next Super Bowl to the array
  Dim prompt As String
  'Add one more element to the array
```

```
    ReDim Preserve teamNames(numGames)
    numGames += 1
    'Request the name of the next winner
    prompt = "Enter winner of game #" & numGames & "."
    teamNames(numGames - 1) = InputBox(prompt, "Super Bowl")
    'Update the title bar of the form and the caption of the button
    Me.Text = "First" & numGames & "Super Bowls"
    btnAddWinner.Text = "Add Winner of Game " & (numGames + 1)
  End Sub
```

[Run, type "Steelers" into the text box, and press the *Display* button. Then feel free to add subsequent winners. Your additions will be taken into account when you next press the *Display* button.]



Steelers

```
9
10
13
14
40
43
```

Flag Variables

The Boolean variable *noWins* in the btnDisplay_Click procedure of Example 5 keeps track of whether a certain situation has occurred. Such a variable is called a **flag**. Flags are used within loops to provide information that will be utilized after the loop terminates. Flags also provide an alternative method of terminating a loop.

For Each Loops

Consider Example 3. The entire sequence of ages can be displayed in the list box with the following statements:

```
For i As Integer = 0 To 43
  lstValues.Items.Add(ages(i))
Next
```

The first line of the For . . . Next loop also could have been written as

```
For i As Integer = 0 To ages.Count - 1
```

In the two For statements, the lower bound and upper bound are given. Another type of loop, called a **For Each loop**, cycles through all the elements of the array in order with no mention whatsoever of the two bounds. The following block of code has the same output as the For . . . Next loop above.

```
For Each age As Integer In ages
  lstValues.Items.Add(age)
Next
```

In general, a block of the form

```
For Each variableName As DataType In arrayName
  statement(s)
Next
```

where *DataType* is the data type of the array, declares the looping variable *variableName* to be of that type, and executes the statement(s) once for each element of the array. That is, at each iteration of the loop, Visual Basic sets the variable to an element in the array and executes the statement(s). When all the elements in the array have been assigned to the variable, the For Each loop terminates and the statement following the Next statement is executed. ***Note:*** When you use local type inference (allowed when Option Infer is set to On), you can omit the *As DataType* clause.

Although For Each loops are less complicated to write than For . . . Next loops, they have a major limitation. They cannot alter the values of elements of the array.

Passing an Array to a Procedure

An array declared in a procedure is local to that procedure and unknown to all other procedures. However, a local array can be passed to another procedure. The argument in the calling statement consists of the name of the array. The corresponding parameter in the header for the procedure must consist of an array name followed by an empty set of parentheses. Like all other parameters, array parameters are preceded with ByVal or ByRef and are followed with "As DataType" clauses. However, any changes to elements of an array passed by value persist after the procedure terminates.

Example 6 The following variation of Example 4 calculates the maximum value with a Function procedure. Notice that the parameter in the function header is written `ByVal ages() As Integer`, not `ByVal ages As Integer`, and the function call is written `Maximum(ages)`, not `Maximum(ages())`.

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  'Calculate the greatest age at inauguration for the last 9 presidents
  Dim ages() As Integer = {55, 56, 61, 52, 69, 64, 46, 54, 47}
  txtOutput.Text = "The greatest age is " & Maximum(ages) & "."
End Sub

Function Maximum(ByVal ages() As Integer) As Integer
  Dim max As Integer = ages(0)
  For i As Integer = 1 To ages.Count - 1
    If ages(i) > max Then
      max = ages(i)
    End If
  Next
  Return max
End Function
```

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

```
The maximum age is 69.
```

User-Defined Array-Valued Functions

A Function procedure with a header of the form

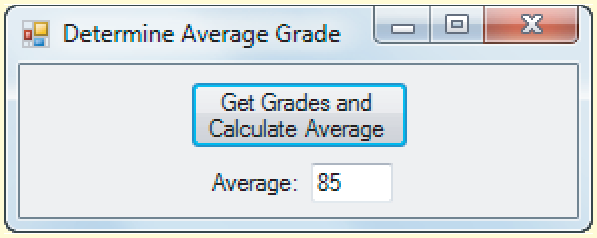
```
Function FunctionName(ByVal var1 As Type1,
                      ByVal var2 As Type2, ...) As DataType()
```

returns an array of type *DataType* as its value. The empty set of parentheses following *DataType* tells us that the Function procedure will return an array instead of just a single value.

Example 7 The following program calculates the average of several grades. The number of grades and the grades themselves are input by the user via input dialog boxes. The Function procedure GetGrades returns an array containing the grades.

```
Private Sub btnGet_Click(...) Handles btnGet.Click
  Dim numGrades As Integer = CInt(InputBox("Number of grades: ", "Grades"))
  txtAverage.Text = CStr(GetGrades(numGrades).Average)
End Sub

Function GetGrades(ByVal numGrades As Integer) As Double()
  Dim grades(numGrades - 1) As Double
  For i As Integer = 1 To numGrades
    grades(i - 1) = CDbl(InputBox("Grade #" & i & ": ", "Get Grade"))
  Next
  Return grades
End Function
```

[Run, enter 3 as the number of grades, and then enter the grades 80, 85, and 90.]



Searching for an Element in an Array

In Example 5, a loop is used to find the indices of the elements having a value specified by the user. Visual Basic has a method for locating elements that is especially efficient with large arrays. Let *numVar* be an integer variable and *value* be a literal or expression of the same type as the elements of *arrayName*. Then a statement of the form

```
numVar = Array.IndexOf(arrayName, value)
```

assigns to *numVar* the index of the first occurrence of the requested value in *arrayName*. If the value is not found, then −1 is assigned to *numVar*.

Example 8 The file States.txt contains the 50 U.S. states in the order in which they joined the union. The first four lines of the file are as follows:

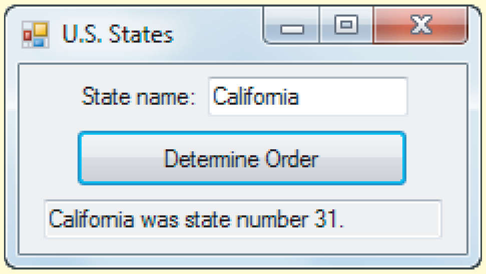
```
Delaware
Pennsylvania
New Jersey
Georgia
```

The following program requests the name of a state and then tells the order in which it joined the union:

```
Dim states() As String = IO.File.ReadAllLines("States.txt")
```

```
Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  Dim n As Integer, state As String
  state = txtState.Text
  n = Array.IndexOf(states, state)
  If n <> -1 Then
    txtOutput.Text = state & " was state number " & n + 1 & "."
  Else
    MessageBox.Show("Re-enter a state name.", "Error")
    txtState.Clear()
    txtState.Focus()
  End If
End Sub
```

[Run, type a state into the top text box, and click on the button.]



California was state number 31.

If a value might occur more than once in an array, an extension of the method above will locate subsequent occurrences. A statement of the form

```
numVar = Array.IndexOf(arrayName, value, startIndex)
```

where *startIndex* is an integer literal or expression, looks only at elements having index *startIndex* or greater, and assigns to *numVar* the index of the first occurrence of the requested value. If the value is not found, then –1 is assigned to *numVar*.

Copying an Array

If *arrayOne* and *arrayTwo* have been declared with the same data type, then the statement

```
arrayTwo = arrayOne
```

makes *arrayTwo* reference the same array as *arrayOne*. It will have the same size and contain the same data. This statement must be used with care, however, since after it is executed, *arrayOne* and *arrayTwo* will share the same portion of memory. Therefore, a change in the value of an element in one of the arrays will affect the other array.

One way to make a copy of an array that does not share the same memory location is illustrated by the following code:

```
'Assume arrayOne and arrayTwo have the same data type and size
For i As Integer = 0 To arrayOne.Count - 1
  arrayTwo(i) = arrayOne(i)
Next
```

Split Method and Join Function

The **Split method** provides another way to assign values to an array. The following code creates the array of Example 1:

```
Dim teamNames() As String
Dim line As String = "Packers,Packers,Jets,Chiefs"
teamNames = line.Split(","c)
```

In general, if *strArray* is a string array and the string variable *strVar* has been assigned a string of the form

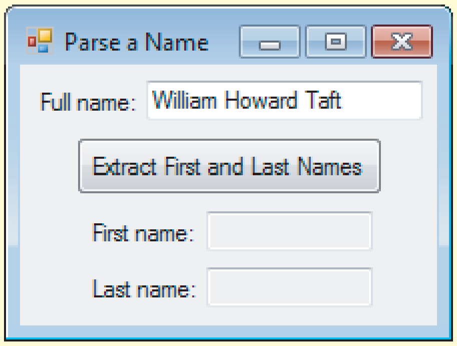
"*value0*,*value1*,*value2*, . . . ,*valueN*"

then a statement of the form

```
strArray = strVar.Split(","c)
```

resizes *strArray* to an array with upper bound *N* having *strArray*(0) = *value*0, *strArray*(1) = *value*1, . . . , *strArray*(*N*) = *valueN*. That is, the first element of the array contains the text preceding the first comma, the second element the text between the first and second commas, . . . , and the last element the text following the last comma. The comma character is called the **delimiter** for the statement above, and the letter *c* specifies that the comma should have data type Character instead of String. Any character can be used as a delimiter. (The two most common delimiters are the comma character and the space character.) The Split method will play a vital role in Section 7.3.

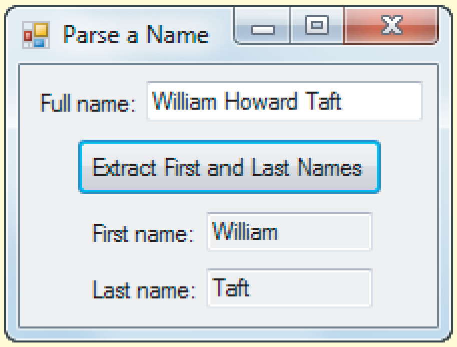
Example 9 The following program determines a person's first and last names. The space character is used as the delimiter for the Split method. Each element of the array contains one part of the person's full name.



OBJECT	PROPERTY	SETTING
frmName	Text	Parse a Name
lblFull	Text	Full name:
txtFull		
btnExtract	Text	Extract First and Last Names
lblFirst	Text	First name:
txtFirst	ReadOnly	True
lblLast	Text	Last name:
txtLast	ReadOnly	True

```
Private Sub btnExtract_Click(...) Handles btnExtract.Click
  Dim fullName As String = txtFull.Text
  Dim parsedName() As String = fullName.Split(" "c)
  txtFirst.Text = parsedName.First
  txtLast.Text = parsedName.Last
End Sub
```

[Run, enter a full name, and click on the button.]



Full name: William Howard Taft

First name: William

Last name: Taft

The reverse of the Split method is the **Join function**, which returns a string value consisting of the elements of an array concatenated together and separated by a specified delimiter. For instance, the code

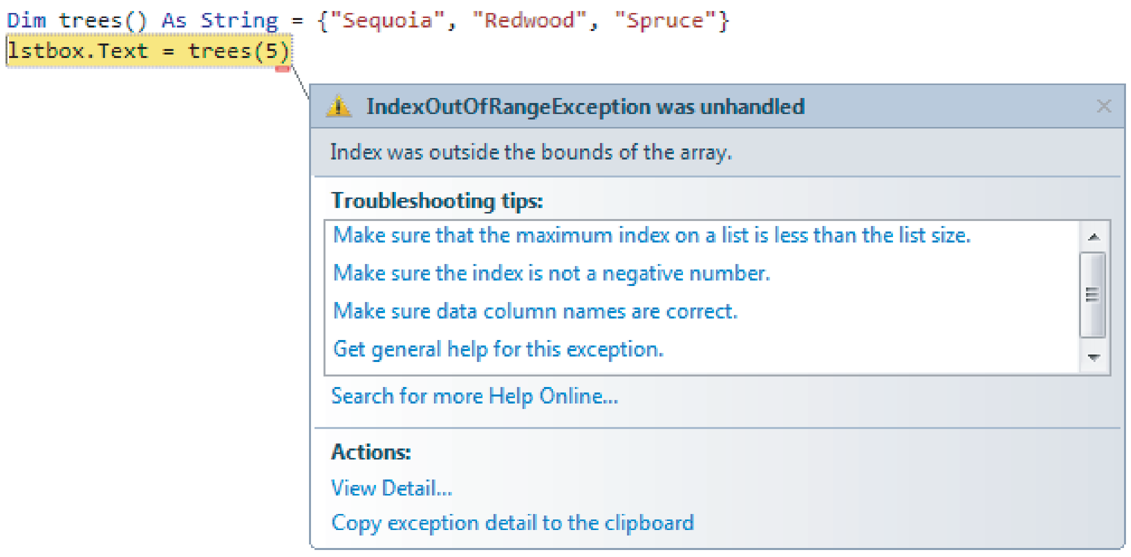
```
Dim greatLakes() As String =
             {"Huron", "Ontario", "Michigan", "Erie", "Superior"}
Dim lakes As String
lakes = Join(greatLakes, ","c)
txtOutput.Text = lakes
```

produces the output

```
Huron,Ontario,Michigan,Erie,Superior
```

Comments

1. Using a subscript greater than the upper bound of an array is not allowed. For instance, at run time the two lines of code in Fig. 7.3 produce an exception dialog box.



```
Dim trees() As String = {"Sequoia", "Redwood", "Spruce"}
lstbox.Text = trees(5)
```

IndexOutOfRangeException was unhandled

Index was outside the bounds of the array.

Troubleshooting tips:

Make sure that the maximum index on a list is less than the list size.
Make sure the index is not a negative number.
Make sure data column names are correct.
Get general help for this exception.

Search for more Help Online...

Actions:

View Detail...
Copy exception detail to the clipboard

FIGURE 7.3 **Exception dialog box.**

2. The statements **Continue For** and **Exit For** can be used in For Each loops in much the same way they are used in For . . . Next loops. Also, a variable declared inside a For Each loop has block-level scope; that is, the variable cannot be referred to by code outside the loop.
3. After you double-click on the name of a text file and place it into the text editor, you can alter the file's contents and save the altered file. To save the altered file, right-click on the file name in the text editor's tab and click on "Save bin\Debug\fileName."

Practice Problems 7.1

1. Give four ways to fill an array with the names of the three musketeers—Athos, Porthos, and Aramis.
2. Write two lines of code that add the name of the fourth musketeer, D'Artagnan, to the array filled in Problem 1.

3. Determine the output displayed when the button is clicked.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim numWords As Integer
  Dim line As String = "This sentence contains five words."
  Dim words() As String = line.Split(" "c)
  numWords = words.Count
  txtOutput.Text = CStr(numWords)
End Sub
```

EXERCISES 7.1

1. What is the size of an array whose upper bound is 100?

2. What is the upper bound of an array whose size is 100?

In Exercises 3 through 26, determine the output displayed when the button is clicked.

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim n As Integer = 2
  Dim spoons(n) As String
  spoons(0) = "soup"
  spoons(1) = "dessert"
  spoons(2) = "coffee"
  txtOutput.Text = "Have a " & spoons(n – 1) & " spoon."
End Sub
```

4.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'I'm looking over a four leaf clover.
  Dim leaves(3) As String
  leaves(0) = "sunshine"
  leaves(1) = "rain"
  leaves(2) = "the roses that bloom in the lane"
  leaves(3) = "somebody I adore"
  For i As Integer = 0 To 3
    lstOutput.Items.Add("Leaf " & (i + 1) & ": " & leaves(i))
  Next
End Sub
```

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim colors(120) As String
  colors(0) = "Atomic Tangerine"
  colors(100) = "Tan"
  If colors(0).IndexOf(colors(100)) = –1 Then
    txtOutput.Text = "No"
  Else
    txtOutput.Text = "Yes"
  End If
End Sub
```

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim years(1) As Integer
  years(0) = 1776
  years(1) = Now.Year   'current year as Integer
  txtOutput.Text = "Age of United States: " & (years(1) - years(0))
End Sub
```

7.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim primes() As Integer = {2, 3, 5, 7, 11}
  lstOutput.Items.Add(primes(2) + primes(3))
End Sub
```

8.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim pres() As String = {"Grant", "Lincoln", "Adams", "Kennedy"}
  txtOutput.Text = pres(3).Substring(0, 3)
End Sub
```

9.
```
Dim bands() As String = {"soloist", "duet", "trio", "quartet"}

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim num As Integer
  ReDim Preserve bands(9)
  bands(4) = "quintet"
  bands(5) = "sextet"
  bands(6) = InputBox("What do you call a group of 7 musicians?")
  num = CInt(InputBox("How many musicians are in your group?"))
  txtOutput.Text = "You have a " & bands(num - 1) & "."
End Sub
```

(Assume the first response is *septet* and the second response is 3.)

10.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Compare the values of two chess pieces
  Dim chess() As String = {"king", "queen", ""}
  chess(2) = "rook"
  ReDim Preserve chess(6)
  chess(3) = "bishop"
  txtOutput.Text = "A " & chess(2) & " is worth more than a " & chess(3)
End Sub
```

11.
```
Dim grades(3) As Double

Private Sub frmGrades_Load(...) Handles MyBase.Load
  grades(0) = 80
  grades(1) = 90
End Sub

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim average As Double
  grades(2) = 70
  grades(3) = 80
  average = (grades(0) + grades(1) + grades(2) + grades(3)) / 4
  txtOutput.Text = "Your average is " & average
End Sub
```

12.
```
Dim names(3) As String

Private Sub frmNames_Load(...) Handles MyBase.Load
```

```
  names(0) = "Al"
  names(1) = "Gore"
  names(2) = "Vidal"
  names(3) = "Sassoon"
End Sub

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  For i As Integer = 0 To 2
    lstOutput.Items.Add(names(i) & " " & names(i + 1))
  Next
End Sub
```

13.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim line As String = "2009,Millionaire,Slumdog"
  Dim films() As String = line.Split(","c)
  txtOutput.Text = films(2) & " " & films(1) & " won in " & films(0)
End Sub
```

14.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim line As String = "2,7,11,13,3"
  Dim nums() As String = line.Split(","c)
  txtOutput.Text = CStr(CInt(nums(4)) * CInt(nums(2)))
End Sub
```

15.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"one", "two", "three"}
  txtOutput.Text = Join(words, ","c)
End Sub
```

16.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {1, 2, 3}
  Dim temp(2) As String
  For i As Integer = 0 To 2
    temp(i) = CStr(nums(i))
  Next
  txtOutput.Text = Join(temp, ","c)
End Sub
```

17.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {3, 5, 8, 10, 21}
  Dim total As Integer = 0
  For Each num As Integer In nums
    If (num Mod 2 = 0) Then   '(num Mod 2) = 0 when num is even
      total += 1
    End If
  Next
  txtOutput.Text = total & " even numbers"
End Sub
```

18.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"When", "in", "the", "course",
                           "of", "human", "events"}
  Dim flag As Boolean = False
  For Each word As String In words
```

```
      If (word.Length = 5) Then
        flag = True
      End If
    Next
    If flag Then
      txtOutput.Text = "at least one five-letter word"
    Else
      txtOutput.Text = "no five-letter word"
    End If
  End Sub
```

In Exercises 19 through 22, assume the five lines of the file Dates.txt contain the numbers 1492, 1776, 1812, 1929, and 1941 and the file is in the program's *bin**Debug* folder.

19.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim dates() As String = IO.File.ReadAllLines("Dates.txt")
  txtOutput.Text = "Pearl Harbor: " & dates(4)
End Sub
```

20.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim dates() As String = IO.File.ReadAllLines("Dates.txt")
  txtOutput.Text = "Bicentennial Year: " & (CInt(dates(1)) + 200)
End Sub
```

21.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim dates() As String = IO.File.ReadAllLines("Dates.txt")
  Dim flag As Boolean = False
  For Each yr As String In dates
    If (CInt(yr) >= 1800) And (CInt(yr) <= 1899) Then
      flag = True
    End If
  Next
  If flag Then
    txtOutput.Text = "contains a 19th-century date"
  Else
    txtOutput.Text = "does not contain a 19th-century date"
  End If
End Sub
```

22.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim dates() As String = IO.File.ReadAllLines("Dates.txt")
  Dim total As Integer = 0
  For Each yr As String In dates
    If (CInt(yr) >= 1900) Then
      total += 1
    End If
  Next
  txtOutput.Text = total & " 20th-century dates"
End Sub
```

23.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"We", "the", "People", "of", "the",
              "United", "States", "in", "Order", "to", "form",
              "a", "more", "perfect", "Union"}
  txtOutput.Text = BeginWithVowel(words) & " words begin with a vowel"
End Sub
```

```
Function BeginWithVowel(ByVal words() As String) As Integer
  Dim total As Integer = 0
  For Each word As String In words
    word = word.ToUpper
    If word.StartsWith("A") Or word.StartsWith("E") Or
       word.StartsWith("I") Or word.StartsWith("O") Or
       word.StartsWith("U") Then
      total += 1
    End If
  Next
  Return total
End Function
```

24.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim grades() As Integer = {85, 95, 90}
  grades = CurveGrades(grades)
  For Each grade As Integer In grades
    lstOutput.Items.Add(grade)
  Next
End Sub

Function CurveGrades(ByVal scores() As Integer) As Integer()
  For i As Integer = 0 To scores.Count - 1
    scores(i) = scores(i) + 7
    If scores(i) > 100 Then
      scores(i) = 100
    End If
  Next
  Return scores
End Function
```

25.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {2, 6, 4}
  nums = Reverse(nums)
  For Each num As Integer In nums
    lstOutput.Items.Add(num)
  Next
End Sub

Function Reverse(ByVal nums() As Integer) As Integer()
  Dim n = nums.Count - 1
  Dim temp(n) As Integer
  For i As Integer = 0 To n
    temp(i) = nums(n - i)
  Next
  Return temp
End Function
```

26.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim speech() As String = {"Four", "score", "and",
                            "seven", "years", "ago"}
  speech = UpperCase(speech)
  txtOutput.Text = speech(3)
End Sub
```

```
Function UpperCase(ByVal words() As String) As String()
  Dim n As Integer = words.Count – 1
  Dim temp(n) As String
  For i As Integer = 0 To n
    temp(i) = words(i).ToUpper
  Next
  Return temp
End Function
```

27. The array declared with the statement

```
Dim lakes() As String = {"Huron", "Ontario", "Michigan", "Erie",
                          "Superior"}
```

contains the names of the five Great Lakes. Evaluate and interpret each of the following:

(a) `lakes.Max` (b) `lakes.Min`
(c) `lakes.First` (d) `lakes.Last`
(e) `lakes.Count` (f) `lakes(1)`
(g) `Array.IndexOf(lakes, "Erie")`

28. The array declared with the statement

```
Dim lakeAreas() As Integer = {23000, 8000, 22000, 10000, 32000}
```

contains the surface areas (in square miles) of the five Great Lakes. Evaluate and interpret each of the following:

(a) `lakeAreas.Max` (b) `lakeAreas.Min`
(c) `lakeAreas.First` (d) `lakeAreas.Last`
(e) `lakeAreas.Count` (f) `lakeAreas.Sum`
(g) `lakeAreas.Average` (h) `lakeAreas(2)`
(i) `Array.IndexOf(lakeAreas, 8000)`

29. The array declared with the statement

```
Dim statePops() As Double = {3.5, 6.5, 1.3, 1.1, 0.7, 1.3}
```

contains the populations (in millions) of the six New England states. Evaluate and interpret each of the following:

(a) `statePops.Max` (b) `statePops.Min`
(c) `statePops.First` (d) `statePops.Last`
(e) `statePops.Count` (f) `statePops(3)`
(g) `Array.IndexOf(statePops, 1.1)`

30. The array declared with the statement

```
Dim statesNE() As String = {"Connecticut", "Massachusetts",
         "New Hampshire", "Rhode Island", "Vermont", "Maine"}
```

contains the names of the six New England states listed in the order in which they became part of the United States. Evaluate and interpret each of the following:

(a) `statesNE.Max` (b) `statesNE.Min`
(c) `statesNE.First` (d) `statesNE.Last`
(e) `statesNE.Count` (f) `statesNE(0)`
(g) `Array.IndexOf(statesNE, "Maine")`

31. Suppose the array *states* has been filled with the names of the fifty states in the order in which they became part of the United States. Write code to display each of the following states in a list box.

    (a) the first state to join the union
    (b) the original thirteen states
    (c) the most recent state to join
    (d) the order number for Ohio
    (e) the second state to join the union
    (f) the twentieth state to join the union
    (g) the last ten states to join the union

32. Suppose the array *pres* has been filled with the names of the 44 U.S. presidents in the order in which they served. Write code to display each of the following in a list box.

    (a) the first president
    (b) the first six presidents
    (c) the most recent president
    (d) the number for "James Monroe"
    (e) the second president
    (f) the tenth president
    (g) the last five presidents

Assume the array *nums* contains a list of positive integers. In Exercises 33 through 38, write a Function procedure that calculates the stated value with a For Each loop.

33. the sum of the numbers in the array
34. the average of the numbers in the array
35. the largest even number in the array (If there are no even numbers in the array, the Function procedure should return 0.)
36. the smallest number in the array
37. the number of two-digit numbers in the array
38. the number of even numbers in the array

In Exercises 39 through 42, identify the errors.

39.
```
Dim nums(3) As Integer = {1, 2, 3, 4}
```
40.
```
Dim nums(10) As Integer
nums(nums.Count) = 7
```
41.
```
Dim nums() As Integer = {1, 2, 3}
'Display 101 + 102 + 103
For Each num As Integer In nums
  num += 100
Next
MessageBox.Show(CStr(nums.Sum))
```
42.
```
Dim nums() As Integer = IO.File.ReadAllLines("Numbers.txt")
```
43. Write a single line of code that displays the number of words in a sentence, where the string variable *line* holds the sentence.
44. Write a single line of code that displays the number of names in the file Names.txt.
45. The file Numbers.txt contains a list of integers. Write a program that displays the number of integers in the file and their sum.
46. The file SomeStates.txt contains a list of some U.S. states. Write a program to determine if the states are in alphabetical order.
47. The file Names2.txt contains a list of names in alphabetical order. Write a program to find and display those entries that are repeated in the file. When a name is found to be repeated, display it only once.

48. Suppose the file Final.txt contains student grades on a final exam. Write a program that displays the average grade on the exam and the percentage of grades that are above average.

49. The file Digits.txt contains a list of digits, all between 0 and 9. Write a program that displays the frequency of each digit.

50. Write a Boolean-valued Function procedure AreSame to compare two integer arrays and determine whether they have the same size and hold identical values—that is, whether $a(i) = b(i)$ for all i.

51. Write a Function procedure to calculate the sum of the entries with odd subscripts in an integer array.

52. The file States.txt contains the names of the 50 U.S. states. Write a program that creates an array consisting of the states beginning with “New”. The program also should display the names of these states in a list box.

53. Table 7.2 shows the different grades of eggs and the minimum weight required for each classification. Write a program that processes the text file Eggs.txt containing a list of the weights of a sample of eggs. The program should report the number of eggs in each grade and the weight of the lightest and heaviest egg in the sample. Figure 7.4 shows the output of the program. ***Note:*** Eggs weighing less than 1.5 ounces cannot be sold in supermarkets and therefore will not be counted.

TABLE 7.2 Grades of eggs.

Grade	Minimum Weight (in ounces)
Jumbo	2.5
Extra Large	2.25
Large	2
Medium	1.75
Small	1.5

```
57 Jumbo eggs
95 Extra Large eggs
76 Large eggs
96 Medium eggs
77 Small eggs
Lightest egg: 1 ounces
Heaviest egg: 2.69 ounces
```

FIGURE 7.4 Output for Exercise 53.

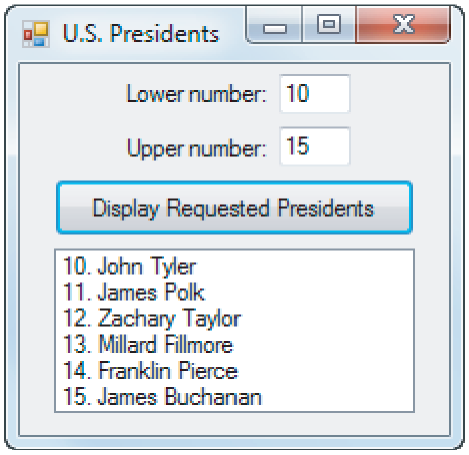
54. The file USPres.txt contains the names of the 44 U.S. presidents in the order in which they served. Write a program that places the names in an array and displays all presidents for a requested range of numbers. Figure 7.5 on the next page shows one possible outcome. (John Tyler was the tenth president. James Polk was the eleventh president. And so on.)

Exercises 55 through 58 should use the file Colors.txt that contains the names of the colors of Crayola® crayons in alphabetical order.

55. Write a program to read the colors into an array and then display the colors beginning with a specified letter. One possible outcome is shown in Fig. 7.6 on the next page.

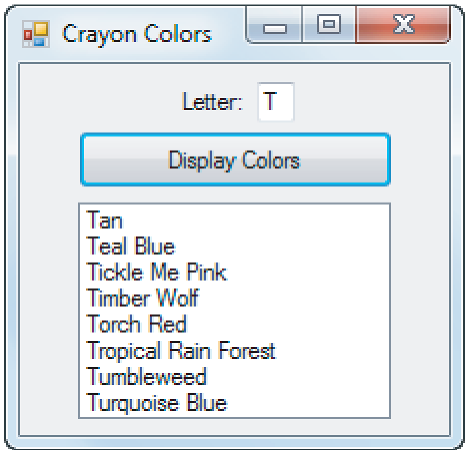
56. Write a program that requests a color as input in a text box and then determines whether or not the color is in the text file. The program should use the Boolean-valued Function procedure IsCrayola that returns the value True if the color in the text box is a Crayola color.

57. Redo Exercise 55 with the letter passed to a Function procedure that returns a smaller array containing just the colors beginning with the specified letter.



```
10. John Tyler
11. James Polk
12. Zachary Taylor
13. Millard Fillmore
14. Franklin Pierce
15. James Buchanan
```

FIGURE 7.5 **Possible outcome of Exercise 54.**



```
Tan
Teal Blue
Tickle Me Pink
Timber Wolf
Torch Red
Tropical Rain Forest
Tumbleweed
Turquoise Blue
```

FIGURE 7.6 **Possible outcome of Exercise 55.**

58. Write a program that displays the colors in reverse alphabetical order.

59. The file Sonnet.txt contains Shakespeare's Sonnet 18. Each entry in the file contains a line of the sonnet. Write a program that reports the average number of words in a line and the total number of words in the sonnet.

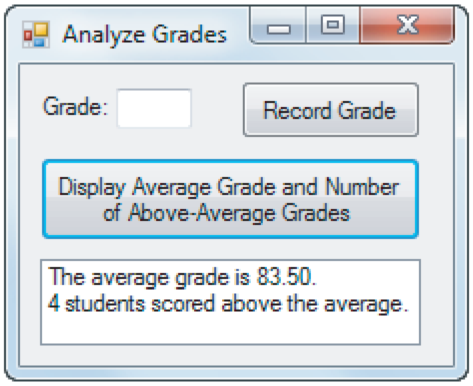
60. Statisticians use the concepts of range, mean, and standard deviation to describe a collection of numerical data. The **range** is the difference between the largest and smallest numbers in the collection. The **mean** is the average of the numbers, and the **standard deviation** measures the spread or dispersal of the numbers about the mean. Formally, if $x_1, x_2, x_3, \ldots, x_n$ is a collection of numbers, then

$$\text{mean} = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n} \qquad \text{(denote the mean by } m\text{)}$$

$$\text{standard deviation} = \sqrt{\frac{(x_1 - m)^2 + (x_2 - m)^2 + (x_3 - m)^2 + \cdots + (x_n - m)^2}{n}}$$

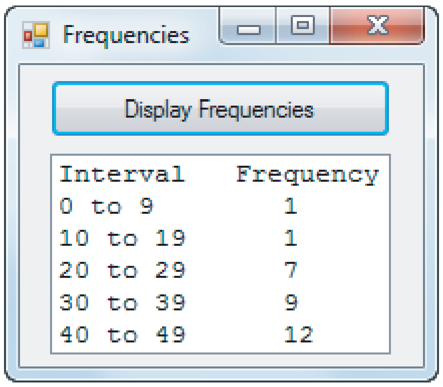
Write a program to calculate the range, mean, and standard deviation for the numbers in the file Data.txt.

61. Write a program to display the average grade and the number of above-average grades on an exam. Each time the user clicks a *Record Grade* button, a grade should be read from a text box. The current average grade and the number of above-average grades should be displayed in a list box whenever the user clicks on a *Display Average* button. (Assume that the class has at most 100 students.) Use a Function procedure to count the number of above-average grades. See Fig. 7.7.



```
The average grade is 83.50.
4 students scored above the average.
```

FIGURE 7.7 **Possible outcome of Exercise 61.**



```
Interval    Frequency
0 to 9          1
10 to 19        1
20 to 29        7
30 to 39        9
40 to 49        12
```

FIGURE 7.8 **Outcome of Exercise 62.**

62. The file Scores.txt contains scores between 1 and 49. Write a program that uses these scores to create an array *frequencies* as follows:

frequencies(0) = # of scores < 10
frequencies(1) = # of scores with 10 ≤ score < 20
frequencies(2) = # of scores with 20 ≤ score < 30
frequencies(3) = # of scores with 30 ≤ score < 40
frequencies(4) = # of scores with 40 ≤ score < 50

The program should then display the results in tabular form, as shown in Fig. 7.8.

In Exercises 63 and 64, execute the statement `sentence = sentence.Replace(",", "")` to remove all commas in *sentence*, and then remove other punctuation marks similarly. After that, use the space character as a delimiter for the Split method.

63. A sentence is called a *chain-link* sentence if the last two letters of each word are the same as the first two letters of the next word—for instance, "The head administrator organized education on online networks." Write a program that accepts a sentence as input and determines whether it is a chain-link sentence. Test the program with the sentence "Broadcast station, once certified, educates estimable legions."

64. A *word palindrome* is a sentence that reads the same, word by word, backward and forward (ignoring punctuation and capitalization). An example is "You can cage a swallow, can't you, but you can't swallow a cage, can you?" Write a program that requests a sentence and then determines whether the sentence is a word palindrome. The program should place the words of the sentence in an array and use a Function procedure to determine whether the sentence is a word palindrome. (Test the program with the sentences "Monkey see, monkey do." and "I am; therefore, am I?")

Solutions to Practice Problems 7.1

1. *First*:
```
Dim names(2) As String
names(0) = "Athos"
names(1) = "Porthos"
names(2) = "Aramis"
```
Second:
```
Dim names() As String = {"Athos", "Porthos", "Aramis"}
```
Third:
```
Dim line As String = "Athos,Porthos,Aramis"
Dim names() As String = line.Split(","c)
```
Fourth: Assume the text file Names.txt has the three names in three lines and is located in the *bin\Debug* folder of the program. Then execute the following line of code:
```
Dim names() As String = IO.File.ReadAllLines("Names.txt")
```
2.
```
ReDim Preserve names(3)      'resize the array
names(3) = "D'Artagnan"      'assign value to last element
```
3. 5

7.2 Using LINQ with Arrays

LINQ (Language-INtegrated Query), a recent exciting and powerful innovation in Visual Basic, provides a standardized way to retrieve information from data sources. In this book we use LINQ with arrays, text files, XML documents, and databases. Before LINQ you often had to write complex loops that specified *how* to retrieve information from a data source. With LINQ you simply