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

state *what* you want to achieve and let Visual Basic do the heavy lifting. ***Important:*** Option Infer must be set to "On" in order to use LINQ. (See Comment 1 at the end of this section.)

VideoNote
LINQ

LINQ Queries

A LINQ *query* for an array is declarative (that is, self-evident) code that describes *what* you want to retrieve from the array. A statement of the form

```
Dim queryName = From var In arrayName Where [condition on var] Select var
```

is called a LINQ query. The variable *var* takes on the values of elements in the array much like the looping variable in a For Each loop. The statement declares a variable *queryName* and assigns it a sequence consisting of the elements of the array that satisfy the condition on *var*. The phrases "`From var In arrayName`", "`Where [condition on var]`", and "`Select var`" are called **query clauses**. The keywords From, Where, and Select are called **query operators,** *var* is called a **range variable,** and *arrayName* is called the **source data**. The entire expression to the right of the equal sign is called a **query expression**.

The LINQ query above is usually written in the style

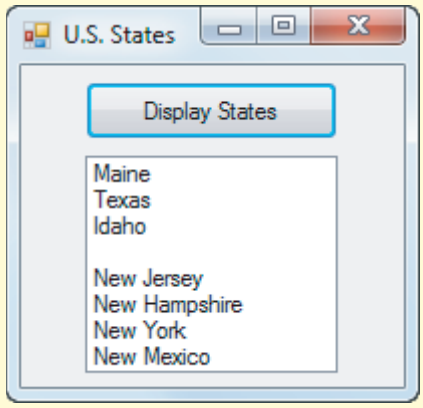
```
Dim queryName = From var In arrayName
                Where [condition on var]
                Select var
```

As soon as you type the first line, the Code Editor will know that you are declaring a query and will treat each press of the Enter key as signaling a line continuation. However, after you type the last clause of the query, you can press Ctrl + Shift + Enter to tell the Code Editor that the query declaration is complete. (Alternately, you can press the Enter key twice to complete entry of the query.)

Example 1 The file States.txt contains the 50 U.S. states in the order in which they joined the union. The following program first displays the states with five-letter names and then displays the states beginning with "New". Each query expression returns a sequence of states that is displayed with a For Each loop.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim states() As String = IO.File.ReadAllLines("States.txt")
  Dim stateQuery1 = From state In states
                    Where state.Length = 5
                    Select state
  For Each state As String In stateQuery1
    lstStates.Items.Add(state)
  Next
  lstStates.Items.Add("")
  Dim stateQuery2 = From state In states
                    Where state.StartsWith("New")
                    Select state
  For Each state As String In stateQuery2
    lstStates.Items.Add(state)
  Next
End Sub
```

[Run, and click on the button.]

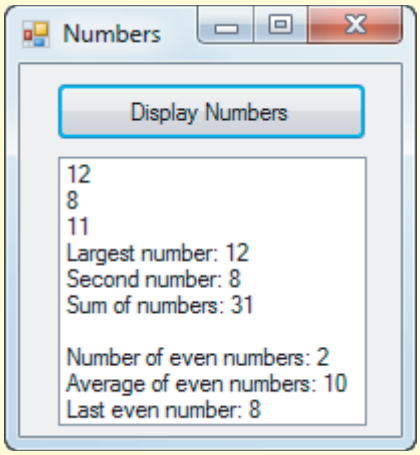


The array methods Count, Max, Min, First, and Last apply to all sequences returned by LINQ queries, and the array methods Average and Sum apply to numeric sequences. Also, the successive elements in the sequence can be referred to by indices ranging from 0 to *queryName*.Count − 1. For instance, in Example 1, the values of `stateQuery1(0)`, `stateQuery1(1)`, and `stateQuery1(2)` are Maine, Texas, and Idaho.

Example 2 The following program displays values associated with numeric sequences returned by LINQ queries. ***Note:*** The integer *n* is even if *n* Mod 2 is 0.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {5, 12, 8, 7, 11}
  Dim numQuery1 = From num In nums
                  Where num > 7
                  Select num
  For Each num As Integer In numQuery1
    lstBox.Items.Add(num)
  Next
  lstBox.Items.Add("Largest number: " & numQuery1.Max)
  lstBox.Items.Add("Second number: " & numQuery1(1))
  lstBox.Items.Add("Sum of numbers: " & numQuery1.Sum)
  lstBox.Items.Add("")
  Dim numQuery2 = From num In nums
                  Where num Mod 2 = 0
                  Select num
  lstBox.Items.Add("Number of even numbers: " & numQuery2.Count)
  lstBox.Items.Add("Average of even numbers: " & numQuery2.Average)
  lstBox.Items.Add("Last even number: " & numQuery2.Last)
End Sub
```

[Run, and click on the button.]



The variable in the Select clause can be replaced by an expression involving the variable. For instance, if the clause `Select num` in *numQuery1* of Example 2 were replaced by

```
Select num * num
```

then the first three lines in the list box would be 144, 64, and 121. Also, both Where clauses and Select clauses are optional. When the Where clause is missing, all values in the source data are included. A missing Select clause produces the same effect as the clause `Select var`. In this textbook we always include a Select clause.

The Distinct Operator

The sequence created with a LINQ query might contain duplicate elements. Duplicates can be eliminated by adding the Distinct operator to the query. For instance, using the array *teamNames* from Example 5 of the previous section, the lines of code

```
Dim teamQuery = From team In teamNames
                Select team
                Distinct
For Each team As String In teamQuery
  lstGamesWon.Items.Add(team)
Next
```

display the names of the teams that have won a Super Bowl, with each team listed once.

The ToArray Method

The sequence returned by a LINQ query has many of the features of an array. Its main limitation is that its values cannot be altered with assignment statements. However, the sequence can be converted to an array with the ToArray method. For instance, using the array *teamNames* from Example 5 of the previous section, the lines of code

```
Dim teamQuery = From team In teamNames
                Select team
                Distinct
Dim uniqueWinners() As String = teamQuery.ToArray
```

create the array named *uniqueWinners* containing the names of the teams that have won the Super Bowl, with each team appearing just once as an element of the array.

Use of Function Procedures in Queries

The Where and Select clauses of a LINQ query can use Function procedures, as illustrated in the next example.

Example 3 The file USPres.txt contains the names of the 44 U.S. presidents. The first two lines of the file are George Washington and John Adams. The following program asks the user to enter a first name and then displays the names of all the presidents having that first name:

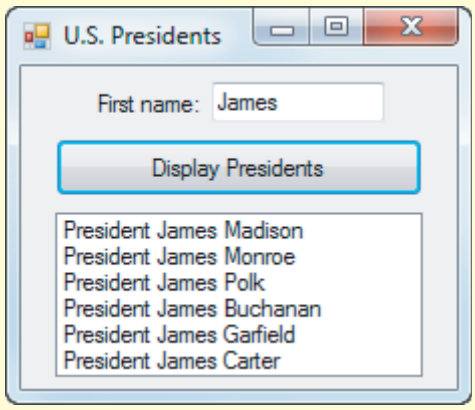
```
Dim presidents() As String = IO.File.ReadAllLines("USPres.txt")

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim presQuery = From pres In presidents
                  Where FirstName(pres) = txtFirstName.Text
                  Select IncludeTitle(pres)
  lstPres.Items.Clear()
  For Each pres In presQuery
    lstPres.Items.Add(pres)
  Next
End Sub

Function FirstName(ByVal name As String) As String
  'Extract the first name from a full name.
  Dim parsedName() As String = name.Split(" "c)
  Return parsedName.First
End Function

Function IncludeTitle(ByVal pres As String) As String
  Return "President " & pres
End Function
```

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



The Let Operator

The Let operator, which gives a name to an expression, makes queries easier to read. For instance, the query in Example 3 can be written as

```
Dim presQuery = From pres In presidents
                Where FirstName(pres) = txtFirstName.Text
                Let formalName = IncludeTitle(pres)
                Select formalName
```

Let operators also can significantly improve the readability of queries.

The Order By Operator

An array or query result is said to be **ordered** if its values are in either ascending or descending order. With ascending order, the value of each element is less than or equal to the value of the next element. That is,

[each element] ≤ [next element].

For string values, the ANSI table is used to evaluate the "less than or equal to" condition.

Putting elements in alphabetical or numeric order (either ascending or descending) is referred to as **sorting**. There are many algorithms for sorting arrays. The most efficient ones use complex nested loops and are tricky to program. However, LINQ provides the Order By query operator that spares us from having to code complicated sorting algorithms. The simplest form of an Order By clause is

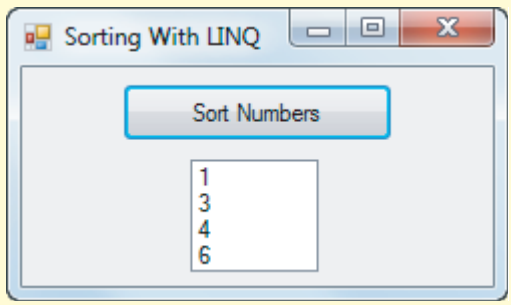
```
Order By [expression] Direction
```

where *Direction* is one of the keywords Ascending or Descending, and the expression involves range and/or Let variables.

Example 4 The following program sorts an array of numbers in ascending order. ***Note:*** If the word Ascending is replaced by Descending, the array will be sorted in descending order.

```
Private Sub btnSort_Click(...) Handles btnSort.Click
  Dim nums() As Integer = {3, 6, 4, 1}
  Dim numQuery = From num In nums
                 Order By num Ascending
                 Select num
  For Each n As Integer In numQuery
    lstOutput.Items.Add(n)
  Next
End Sub
```

[Run, and click on the button.]



The Order By operator is quite flexible and can order arrays in ways other than just alphabetical or numeric order. Secondary criteria for ordering can be specified by listing two or more criteria separated by commas. In general, an Order By clause of the form

```
Order By expression1 Direction1, expression2 Direction2, ...
```

primarily sorts by expression1 and Direction1, secondarily by expression2 and Direction 2, and so on. For instance, an Order By clause such as

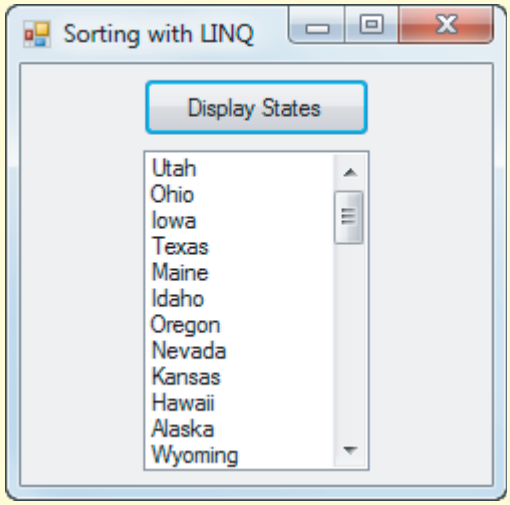
```
Order By lastName Ascending, firstName Ascending
```

can be used to alphabetize a sequence of full names. When two people have the same last name, their first names will be used to determine whose full name comes first.

Example 5 The following program uses the file States.txt considered in Example 1 and sorts the states by the length of their names in ascending order. States with names of the same length are sorted by their names in reverse alphabetical order.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim states() As String = IO.File.ReadAllLines("States.txt")
  Dim stateQuery = From state In states
                   Order By state.Length Ascending, state Descending
                   Select state
  For Each state As String In stateQuery
    lstStates.Items.Add(state)
  Next
End Sub
```

[Run, and click on the button.]



Note: In an Order By clause, the default direction is *Ascending*. For instance, the fourth line of Example 5 could have been written

```
Order By state.Length, state Descending
```

The DataSource Property

In Example 5, a For Each loop was used to place the values returned by the query into a list box. The task also can be accomplished with a DataSource property. To use the DataSource property, replace the For Each loop with the following pair of statements. ***Note:*** The second statement is optional. It prevents having the first item in the list box selected at startup.

```
lstStates.DataSource = stateQuery.ToList
lstStates.SelectedItem = Nothing
```

The DataSource property also can be used to display the contents of an array directly into a list box with a statement of the form

```
lstBox.DataSource = arrayName
```

Note: If a SelectedIndexChanged event procedure has been defined for a list box, then execution of the DataSource property will raise the SelectedIndexChanged event.

Binary Search

A large array in ascending order is most efficiently searched with a **binary search**. The BinarySearch method looks for a value in the array by first determining in which half of the array it lies. The other half is then ignored, and the search is narrowed to the retained half. The process is repeated until the item is found or the entire list has been considered. A statement of the form

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

assigns to *numVar* the index of an occurrence of the requested value in *arrayName*. If the value is not found, then a negative number is assigned to *numVar*.

Comments

1. The Option Infer setting can be made the default setting for all of your Visual Basic programs or can be made the setting for a single program.

   (a) If a value for Option Infer is set in the Option default project setting window shown in Fig. 3.1 of Section 3.2, then that setting will be the default setting for all new programs.

   (b) To override the default setting for an individual program, enter the statement `Option Infer On` or `Option Infer Off` at the top of the Code Window. Alternately, right-click on the program name at the top of Solution Explorer, click on Properties, click on the Compile tab, and change the value for "Option infer".

2. LINQ Where operators are said to *filter* data, and Select operators are said to *project* data.
3. Visual Basic has a built-in routine for sorting arrays. A statement of the form

   ```
   Array.Sort(arrayName)
   ```

   sorts the array in ascending order. This method is useful for simple sorting where the more advanced capabilities of LINQ are not needed.

Practice Problems 7.2

1. Write a program that uses a LINQ query to calculate the sum of the numbers in the file Numbers.txt.
2. The file USPres.txt contains the full names of the 44 U.S. presidents. The following program finds the full name of the president whose last name is "Eisenhower." Since there is only one such president, a text box (rather than a list box) is sufficient to display the output.

```
Private Sub btnFind_Click(...) Handles btnFind.Click
  Dim presidents() As String = IO.File.ReadAllLines("USPres.txt")
  Dim query = From pres In presidents
              Where pres.EndsWith("Eisenhower")
              Select pres
  txtFullName.Text = query.First
End Sub
```

(a) Since the value of query consists of just one name, why can't the sixth line be replaced with `txtFullName.Text = query`?
(b) What expressions, other than `query.First`, can be used for the right side of the sixth line that would yield the same result?

EXERCISES 7.2

In Exercises 1 through 18, determine the output displayed when the button is clicked.

1.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {5, 7, 2, 3}
  Dim numQuery = From num In nums
                 Where num > 4
                 Select num
  For Each num As Integer In numQuery
    lstOutput.Items.Add(num)
  Next
End Sub
```

2.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"Houston", "we", "have", "a", "problem"}
  Dim wordQuery = From word In words
                  Where word.ToUpper.StartsWith("H")
                  Select word
  For Each word As String In wordQuery
    lstOutput.Items.Add(word)
  Next
End Sub
```

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim line As String = "I'm going to make him an offer he can't refuse"
  Dim words() As String = line.Split(" "c)
  Dim wordQuery = From word In words
                  Where word.Length = 5
                  Select word
  lstOutput.DataSource = wordQuery.ToList
  lstOutput.SelectedItem = Nothing
End Sub
```

4.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim line As String = "1492,1776,1812,1929,1941"
  Dim dates() As String = line.Split(","c)
  Dim dateQuery = From yr In dates
                  Where CInt(yr) < 1800
                  Select yr
  lstOutput.DataSource = dateQuery.ToList
  lstOutput.SelectedItem = Nothing
End Sub
```

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim line As String = "If,you,fail,to,plan,then,you,plan,to,fail"
  Dim words() As String = line.Split(","c)
```

```
  Dim wordQuery = From word In words
                  Select word
                  Distinct
  txtOutput.Text = CStr(wordQuery.Count)
End Sub
```

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {2, 3, 4, 3, 2}
  Dim numQuery = From num In nums
                 Select num
                 Distinct
  txtOutput.Text = CStr(numQuery.Sum)
End Sub
```

7.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {2, 3, 4, 3, 2}
  Dim numQuery = From num In nums
                 Select num + 100
                 Distinct
  txtOutput.Text = CStr(numQuery.Average)
End Sub
```

8.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"racecar", "motor", "kayak", "civics"}
  Dim wordQuery = From word In words
                  Where IsPalindrome(word)
                  Select word.ToUpper
  For Each word As String In wordQuery
    lstOutput.Items.Add(word)
  Next
End Sub

Function IsPalindrome(ByVal word As String) As Boolean
  'A palindrome is a word that reads the same forwards and backwards.
  Dim n As Integer = word.Length
  For i As Integer = 0 To (n - 1) \ 2
    If word.Substring(i, 1) <> word.Substring(n - i - 1, 1) Then
      Return False
    End If
  Next
  Return True
End Function
```

9.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'The first four lines of Numbers.txt contain the numbers 2, 6, 7, and 8.
  Dim numbers() as String = IO.File.ReadAllLines("Numbers.txt")
  Dim query = From num In numbers
              Select CInt(num)
  lstOutput.Items.Add(query(0) + query(1))
End Sub
```

10.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'The first four lines of Words.txt contain scale, top, up, and low.
  Dim words() As String = IO.File.ReadAllLines("Words.txt")
```

```
  Dim query = From word In words
              Select word
  txtOutput.Text = query(2) & query(0)
End Sub
```

11.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim grades() As Integer = {66, 68, 72, 76, 90, 92, 93, 94, 95}
  Dim query = From grade In grades
              Let newGrade = CurveGrade(grade)
              Where newGrade = 100
              Select newGrade
  txtOutput.Text = query.Count & " students have a grade of 100"
End Sub

Function CurveGrade(ByVal grade As Integer) As Integer
  grade += 7
  If grade > 100 Then
    grade = 100
  End If
  Return grade
End Function
```

12.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"rated", "savory", "able", "just"}
  Dim query = From word In words
              Let opposite = ("un" & word).ToUpper
              Select opposite
  txtOutput.Text = query.Max
End Sub
```

13.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nums() As Integer = {12, 5, 7, 10, 3, 15, 4}
  Dim query = From num In nums
              Where num > 10
              Order By num Descending
              Select num
  lstOutput.DataSource = query.ToList
  lstOutput.SelectedItem = Nothing
End Sub
```

14.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim words() As String = {"When", "in", "the", "course",
                           "of", "human", "events"}
  Dim query = From word In words
              Order By word.Length
              Select word.Length
  Dim greatestLength As Integer = query.Last
  Dim query2 = From word In words
               Where word.Length = greatestLength
               Order By word Descending
               Select word
  lstOutput.DataSource = query2.ToList
  lstOutput.SelectedItem = Nothing
End Sub
```

15.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim grades() As Integer = {60, 70, 90, 80}
  Dim query = From grade In grades
              Order By grade Descending
              Select grade
  grades = query.ToArray
  ReDim Preserve grades(grades.Count - 2)   'drop lowest grade
  Dim str As String = "The average after dropping the lowest grade is "
  txtOutput.Text = str & grades.Average
End Sub
```

16.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim golfers(2) As String  'top 3 golfers in tournament
  golfers(0) = "Funk,65,69,69,75"     'total = 278
  golfers(1) = "Ramaro,67,69,65,73"   'total = 274
  golfers(2) = "McNulty,68,70,73,68"  'total = 279
  Dim query = From golfer In golfers
              Let data = golfer.Split(","c)
              Let name = data(0)
              Let score = CInt(data(1)) + CInt(data(2)) +
                          CInt(data(3)) + CInt(data(4))
              Let result = score & "  " & name
              Order By score Ascending
              Select result
  For Each result As String In query
    lstOutput.Items.Add(result)
  Next
End Sub
```

17.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim smallPrimes() As Integer = {2, 3, 5, 7, 11, 13, 17, 19, 23,
                                  29, 31, 37, 41, 43, 47, 53, 59,
                                  61, 67, 71, 73, 79, 83, 89, 97}
  Dim n As Integer = CInt(InputBox("Enter a number less than 100:"))
  If Array.BinarySearch(smallPrimes, n) < 0 Then
    txtOutput.Text = n & " is not a prime number"
  Else
    txtOutput.Text = n & " is a prime number"
  End If
End Sub
```

(Assume the response is 37.)

18.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim statesNE() As String = {"Connecticut", "Maine", "Massachusetts",
                              "New Hampshire", "Rhode Island", "Vermont"}
  Dim state As String = InputBox("Enter a state:")
  If Array.BinarySearch(statesNE, state) < 0 Then
    txtOutput.Text = state & " is not in New England."
  Else
    txtOutput.Text = state & " is in New England."
  End If
End Sub
```

(Assume the response is *New York*.)

Use LINQ to carry out the primary tasks of the programs in the remaining exercises of this section. In Exercises 19 through 24, redo the exercises from Section 7.1 using LINQ queries.

19. Exercise 17 of Section 7.1

20. Exercise 18 of Section 7.1

21. Exercise 21 of Section 7.1

22. Exercise 22 of Section 7.1

23. Exercise 25 of Section 7.1

24. Exercise 26 of Section 7.1

In Exercises 25 through 28, use the file SBWinners.txt that lists the winners of the first 44 Super Bowls.

25. Write a program that displays the teams (in alphabetical order) who have won a Super Bowl. Each team should appear only once.

26. Write a program that displays the teams (in alphabetical order) who have won a Super Bowl and whose name begins with the letter *B*. Each team should appear only once.

27. Write a program that displays in a text box the number of games won by the team specified. See Fig. 7.9.

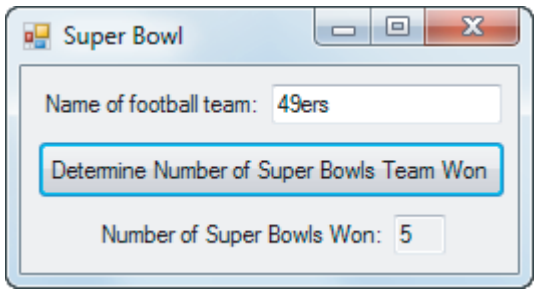


FIGURE 7.9 **Possible outcome of Exercise 27.**

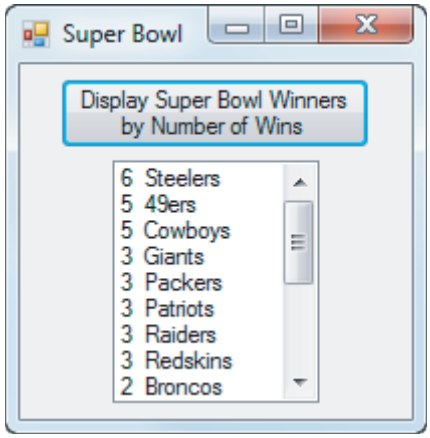


FIGURE 7.10 **Outcome of Exercise 28.**

28. Write a program that displays a list of Super Bowl winners ordered by the number of games won. See Fig. 7.10.

29. The file Final.txt contains student grades on a final exam. Write a program using LINQ that displays the average grade on the exam and the percentage of grades that are above average.

30. Write a program that requests five grades as input and then calculates the average after dropping the two lowest grades.

31. The file States.txt contains the 50 U.S. states in the order in which they joined the union. Write a program to display the original 13 states in alphabetical order.

32. An *anagram* of a word or phrase is another word or phrase that uses the same letters with the same frequency. Punctuation marks, case, and spaces are ignored. Write a program that requests two words (no punctuation) as input and determines if they are anagrams of each other. (Test the program with the words *Elvis* and *lives*.)

33. The file USPres.txt contains the names of the 44 presidents in the order in which they served. The first two lines contain the names George Washington and John Adams. Write a program that displays the presidents ordered by their last name.

34. The file Words.txt contains a list of words. Write a program that displays the words in a list box sorted by the number of different vowels (A, E, I, O and U) in the word. When two words have the same number of different vowels, they should be ordered first by their length (descending) and then alphabetically. The display should show both the word and the number of different vowels in the word. See Fig. 7.11.

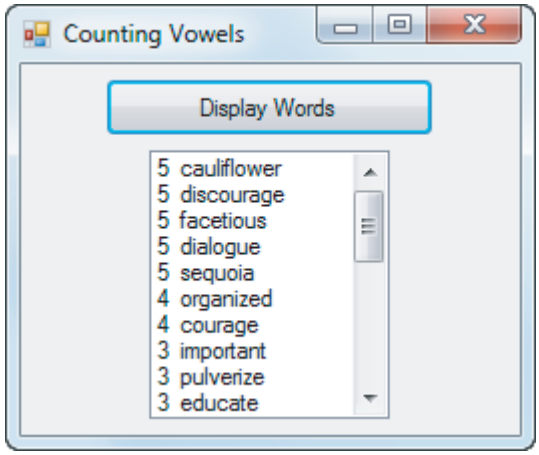


FIGURE 7.11 Output of Exercise 34.

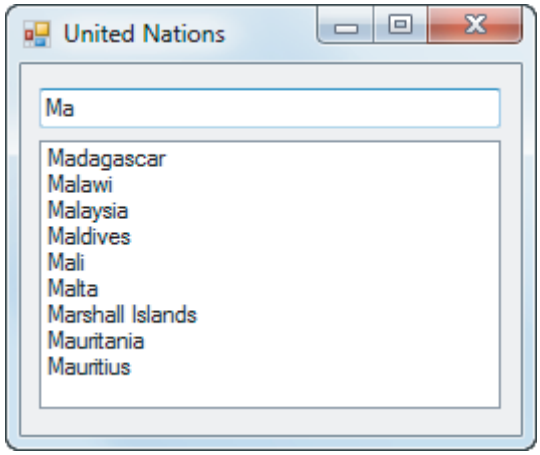


FIGURE 7.12 Output of Exercise 35.

VideoNote
Presidents (Homework)

35. The file Nations.txt contains the names of the 192 member nations of the United Nations. Write a program that initially displays all the nations in a list box. Each time a letter is typed into a text box, the program should reduce the displayed nations to those beginning with the letters in the text box. Figure 7.12 shows the status after the letters "Ma" are typed into the text box. At any time, the user should be able to click on the name of a nation to have it appear in the text box.

36. The **median** of an ordered set of measurements is a number separating the lower half from the upper half. If the number of measurements is odd, the median is the middle measurement. If the number of measurements is even, the median is the average of the two middle measurements. Write a program that requests a number *n* and a set of *n* measurements as input and then displays the median of the measurements.

Solutions to Practice Problems 7.2

1.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim numbers() As String = IO.File.ReadAllLines("Numbers.txt")
    Dim query = From num In numbers
                Select CDbl(num)
    MessageBox.Show(CStr(query.Sum), "Total")
End Sub
```

2. (a) A text box can be filled only with a string. The value returned by a query is a *sequence* type that contains one string element. Only that element, not the sequence itself can be assigned to the text property of the text box.

   (b) `query(0)`, `query.Last`, `query.Max`, `query.Min`

7.3 Arrays of Structures

Often we work with several pieces of related data. For instance, four related pieces of data about a country are *name*, *continent*, *population*, and *area*. Suppose we are considering this information for the 192 countries in the United Nations. In the early days of programming, the way to work