Chapter 7 – Arrays

7.1 Creating and Using Arrays
7.2 Using LINQ with Arrays
7.3 Arrays of Structures
7.4 Two-Dimensional Arrays
7.1 Creating and Using Arrays

- Declaring an Array Variable
- The Load Event Procedure
- Implicit Array Sizing and Initialization
- Text Files
- Array Methods
- Calculating an Array Value with a Loop
- The ReDim Statement
7.1 Creating and Accessing Arrays (continued)

- For Each Loops
- Functions that Return Arrays
- Searching for an Element in an Array
- Copying an Array
- Split Method and Join Function
- Passing an Array to a Procedure
Simple and Array Variables

- A **variable** (or simple variable) is a name to which Visual Basic can assign a single value.
- An **array variable** is a collection of simple variables of the same type to which Visual Basic can efficiently assign a list of values.
Example

Suppose you want to evaluate the exam grades for 30 students and to display the names of the students whose scores are above average.

```vbnet
Private Sub btnDisplay_Click(...) _
Handles btnDisplay.Click
    Dim student0 As String, score0 As Double
    Dim student1 As String, score1 As Double
    Dim student2 As String, score2 As Double
    .
    .
```
Using Arrays

Dim students(29) As String
Dim scores(29) As Double

Upper bound of subscripts in the array
Putting Values into an Array

students(0) = "Tom Brown"

Read: "students sub zero equals Tom Brown"
Which means that the string "Tom Brown" is being stored at the first location in the array called students because all arrays begin counting at 0.
Array Terminology

- `Dim arrayName(n) As DataType`
- 0 is the **lower bound** of the array
- `n` is the **upper bound** of the array—the last available subscript in this array
- The number of elements, $n + 1$, is the **size** of the array.
Example 1: Form

![Form Example]

- mtbNumber
- txtWinner
Example 1

Private Sub btnWhoWon_Click(...) _
    Handles btnWhoWon.Click

    Dim teamNames(3) As String
    Dim n As Integer
    teamNames(0) = "Packers"
    teamNames(1) = "Packers"
    teamNames(2) = "Jets"
    teamNames(3) = "Chiefs"
    n = CInt(mtbNumber.Text)
    txtWinner.Text = teamName(n - 1)
End Sub
Example 1: Output

![Early Super Bowls output](image-url)
Load Event Procedure

Occurs as the Form loads in memory

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

The keyword MyBase refers to the form being loaded. This event procedure is a good place to assign values to an array.
Example 2: Code

Dim teamNames(3) As String

Private Sub frmBowl_Load(...) Handles MyBase.Load
    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
Initializing Arrays

Arrays may be initialized when created:

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

declares an array having upper bound \( N \) and assigns \( value0 \) to \( arrayName(0) \), \( value1 \) to \( arrayName(1) \), ..., and \( valueN \) to \( arrayName(N) \).

Example: ```vbnet
Dim teamNames() As String =
    {"Packers", "Packers", "Jets", "Chiefs"}
```
Text Files

• Hold data to be processed by programs.
• Can be created, viewed, and managed by word processors or by the Visual Basic IDE.
• Have the extension txt
• Normally placed in the bin\Debug folder in the Solution Explorer.
A Text File Displayed in the Visual Basic IDE

The file contains the ages of the first 44 U.S. presidents when they assumed office.
Using a Text File to Populate a String Array

• Assume that the previous text file is in the program’s \bin\Debug folder.
• The text file can be used to fill a string array with the statement

```vbnet
Dim strAges() As String = IO.File.ReadAllLines("AgesAtInaugural.txt")
```

• The array `strAges` will have size 44 and upper bound 43.
Dim strAges() As String = IO.File.ReadAllLines("AgesAtInaugural.txt")

Dim ages(43) As Integer

For i As Integer = 0 To 43
    ages(i) = CInt(strAges(i))

Next
# Array Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>arrayName.Count</code></td>
<td>number of elements</td>
</tr>
<tr>
<td><code>arrayName.Max</code></td>
<td>highest value</td>
</tr>
<tr>
<td><code>arrayName.Min</code></td>
<td>lowest value</td>
</tr>
<tr>
<td><code>arrayName.First</code></td>
<td>first element</td>
</tr>
<tr>
<td><code>arrayName.Last</code></td>
<td>last element</td>
</tr>
</tbody>
</table>
Array Methods (continued)

- The upper bound of `arrayName` is
  \[ \text{arrayName.Count} - 1 \]

- `arrayName.First` is the same as
  \[ \text{arrayName(0)} \]
# Methods for Numeric Arrays

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>numArrayName.Average</code></td>
<td>average value of elements</td>
</tr>
<tr>
<td><code>numArrayName.Sum</code></td>
<td>sum of values of elements</td>
</tr>
</tbody>
</table>
Using Loops Instead of Methods

- In Example 4 the greatest value in a numeric array *ages* is determined.
- The value of the variable *max* is set to the first element of the array.
- Then a For…Next loop successively examines each element of the array and resets the value of *max* when appropriate.
Example 4: Code

Dim ages() As Integer = {55, 56, 61, 52, 69, 64, 46, 54, 47}  'last 9 presidents
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 = "Greatest age: " & max

Output: Greatest age: 69
The size of an array may be changed after it has been created. The statement `ReDim arrayName(m)`, where `arrayName` is the name of the already declared array and `m` is an Integer literal, variable, or expression, changes the upper bound of the array to `m`. 
Preserve Keyword

ReDim arrayName(m) resets all values to their default. This can be prevented with the keyword Preserve.

ReDim Preserve arrayName(m) resizes the array and retains as many values as possible.
For Each Loops

For i As Integer = 1 To (ages.Count - 1)
    If ages(i) > max Then
        max = ages(i)
    End If
Next

can be replaced with

For Each age As Integer In ages
    If age > max Then
        max = age
    End If
Next
For Each Loops (continued)

- In the For…Next loop, the counter variable \( i \) can have any name.
- In the For Each loop, the looping variable \( age \) can have any name.
- The primary difference between the two types of loops is that in a For Each loop no changes can be made in the values of elements of the array.
Passing an Array Element

A single element of an array can be passed to a procedure just like any ordinary numeric or string variable.

```vbnet
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim num(20) As Integer
    num(5) = 10
    lstOutput.Items.Add(Triple(num(5)))
End Sub

Function Triple(x As Integer) As Integer
    Return 3 * x
End Function
```
Functions That Return Arrays

Headers have the form

```plaintext
Function FunctionName(var1 As Typel,
                         var2 As Type2, ...) As DataType()
```
A statement of the form

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

assigns to `numVar` the index of the first occurrence of `value` in `arrayName`. Or assigns -1 if the value is not found.
Copying an Array

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

```
arrayOne = arrayTwo
```

makes `arrayOne` an exact duplicate of `arrayTwo`. Actually, they share the same location in memory.
Split Method

- Facilitates working with text files.
- Split can convert a string containing comma-separated data into a string array.
- The 0th element of the array contains the text preceding the first comma, the 1st element contains the text between the first and second commas, ..., and the last element contains the text following the last comma.
Split Example

For instance, suppose the string array `employees` has been declared without an upper bound, and the string variable `line` has the value “Bob,23.50,45”.

```csharp
employees = line.Split("","c")
```

- sets the size of `employees` to 3
- sets `employees(0) = “Bob”`
- sets `employees(1) = “23.50”`
- sets `employees(2) = “45”`
employees = line.Split("","c")

• In this example, the character comma is called the delimiter for the Split method, and the letter c specifies that the comma has data type Character instead of String.

• Any character can be used as a delimiter. If no character is specified, the space character will be used as the delimiter.
Example

Private Sub btnConvert_Click(...) _
Handles btnConvert.Click

Dim stateData(), line As String
line = "California,1850,Sacramento,Eureka"
stateData = line.Split("","c")
For Each entry As String In stateData
    lstOutput.Items.Add(entry)
Next
End Sub
Example Output

California
1850
Sacramento
Eureka
Join Function

The reverse of the Split method is the Join function. Join concatenates the elements of a string array into a string containing the elements separated by a specified delimiter.

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

Output: Huron,Ontario,Michigan,Erie,Superior
Passing Arrays to Procedures

- An array declared in a procedure is local to that procedure.
- An entire array can be passed to a Sub or Function procedure.
- The calling statement uses the name of the array without parentheses.
- The header of the Sub or Function procedure uses the name with an empty set of parentheses.
Variation of Example 4

This example uses a Function procedure to find the largest number in an array.

```vbnet
Private Sub btnCalculate_Click(...) Handles _
    btnCalculate.Click
    Dim ages() As Integer = {55, 56, 61, 52,
                               69, 64, 46, 54, 47}  'last 9 presidents
    txtOutput.Text = "Greatest age: " &
                     Maximum(ages)
End Sub
```
Variation of Example 4 (cont.)

Function Maximum(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
Out of Range Error

The following code references an array element that doesn't exist. This will cause an error.

```vbscript
Dim trees() As String = {"Sequoia", "Redwood", "Spruce"}
txtBox.Text = trees(5)
```
7.2 Using LINQ with Arrays

- LINQ Queries
- The Distinct Operator
- The ToArray Method
- Use of Function Procedures in Queries
- The Let Operator
- The OrderBy Operator
- The DataSource Property
- Binary Search
What is LINQ?

- LINQ stands for **Language INtegrated Query**
- A **query** is a request for information.
- **Note**: Option Infer must be set to ON in order to use LINQ
LINQ Query

Code of the form

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

declares the variable `queryName` and assigns to it a sequence of the values from `arrayName` that satisfy the stated condition.
The values in the sequence can be converted to an array, displayed in a list box, or written to a text file.
Example 1

'States.txt contains names of the 50 states
Dim states() As String = 
    IO/File/ReadAllLines("States.txt")
Dim stateQuery = From state In states
    Where state.Length = 5
    Select state
For Each state As String In stateQuery
    lstStates.Items.Add(state)
Next
Output: Maine, Texas, Idaho
Variation on Example 1

Replace the For Each loop with

lstStates.Items.Add(stateQuery.Count)
lstStates.Items.Add(stateQuery.Min)
lstStates.Items.Add(stateQuery(1))

Output: 3
   Idaho
   Texas
Example 2

Dim nums() As Integer = {5, 12, 8, 7, 11}
Dim numQuery = From num In nums
    Where num > 7
    Select num
For Each num As Integer In numQuery
    lstBox.Items.Add(num)
Next

Output:
12
8
11
Variation on Example 2

Replace the For Each loop with

```
lstBox.Items.Add(numQuery.Min)
lstBox.Items.Add(numQuery.First)
lstBox.Items.Add(numQuery.Sum)
```

Output: 8

12
31
Dim nums() As Integer = {5, 12, 8, 7, 11}
Dim numQuery = From num In nums
    Where num > 7
    Select num * num  ' changed
For Each num As Integer In numQuery
    lstBox.Items.Add(num)
Next

Output: 144
        64
        121
Distinct Operator

Dim nums() As Integer = {5, 12, 5, 7, 12}
Dim numQuery = From num In nums
    Select num
    Distinct
For Each num As Integer In numQuery
    lstBox.Items.Add(num)
Next

Output:  5
        12
        7
ToArray Method

- A query variable is not an array variable.
- The ToArray method converts it to an array variable.

Dim nums() As Integer = {5, 12, 5, 7, 12}
Dim numQuery = From num In nums
               Select num
Dim numArray() As Integer = numQuery.ToArray
Function procedures are commonly used in Where and Select clauses

```vbscript
Dim presQuery = From pres In presidents
    Where FirstName(pres) = txtFirstName.Text
    Select IncludeTitle(pres)
For Each pres In presQuery
    lstPres.Items.Add(pres)
Next
```
Continuation of Program

Function IncludeTitle(pres As String) As String
    Return "President " & pres
End Function
Let Operator

- A Let operator gives a name to an expression and makes queries easier to read.
- In Section 7.3, situations arise that make the use of Let operators essential.
Example of Let Operator

\[
\text{Dim } \text{presQuery} = \text{From } \text{pres } \text{In } \text{presidents}
\]
\[
\text{Let } \text{formalName} = \text{IncludeTitle(pres)}
\]
\[
\text{Select formalName}
\]

can be replaced with

\[
\text{Dim } \text{presQuery} = \text{From } \text{pres } \text{In } \text{presidents}
\]
\[
\text{Let } \text{formalName} = \text{IncludeTitle(pres)}
\]
\[
\text{Select formalName}
\]
Order By Operator

- Sorts string values into alphabetical order (either ascending or descending)
- Sorts numbers into numeric order (either ascending or descending)
Example 4

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

Output:  
1
3
4
6
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

Output: Utah, Ohio, Iowa, Texas, Maine, `...
DataSource Property

- The DataSource property fills a list box with the values returned by a query.

```csharp
lstBox.DataSource = queryName.ToList
```

- The first entry will be highlighted. The highlighting can be eliminated with

```csharp
lstBox.SelectedItem = Nothing
```
Alternative for Example 5

Dim states() As String =
    IO.File.ReadAllLines("States.txt")
Dim stateQuery = From state In states
    Order By state.Length
    Ascending, state Descending
    Select state
lstStates.DataSource = stateQuery.ToList
lstStates.SelectedItem = Nothing ← optional line

Output: Utah, Ohio, Iowa, Texas, Maine, ...
7.3 Arrays of Structures

- Structures
- Arrays of Structures
- The DataGridView Control
- Searching an Array of Structures
- Using General Procedures with Structures
- Displaying and Comparing Structure Values
- Complex Structures (optional)
A structure is a grouping of heterogeneous data. Also called a UDT (User Defined Type)
Sample structure definition:

```plaintext
Structure Nation
    Dim name As String
    Dim continent As String
    Dim population As Double 'in millions
    Dim area As Double 'in square miles
End Structure
```
Structure Definition

Each subvariable in a structure is called a member.

To declare a variable of a structure type:

```dim country As Nation```

Each member is accessed via

```variableName.memberName```

```country.continent = "Europe"```
Example 1

Dim country As Nation
'Assign values to country's member variables
Dim line As String =
    "China,Asia,1332.5,3696100"
Dim data() As String = line.Split("","c")
country.name = data(0)
country.continent = data(1)
country.population = CDbl(data(2))
country.area = CDbl(data(3))
Example 1 (continued)

'Display data in text boxes

txtName.Text = country.name

txtContinent.Text = country.continent

txtPop.Text = (1000000 *
    country.population).ToString("N0")

txtArea.Text = (country.area).ToString("N0") &
    " square miles"

txtDensity.Text = (1000000 * country.population / country.area).ToString("N0") &
    " people per square mile"
Text File: UN.txt

4 fields (name, continent, pop in millions, area in sq mi)
193 records

Sample records
Canada, North America, 32.9, 3855000
France, Europe, 63.5, 211209
New Zealand, Australia/Oceania, 4.18, 103738
Nigeria, Africa, 146.5, 356669
Pakistan, Asia, 164, 310403
Peru, South America, 27.9, 496226
Example 3: Sample Output

Display Countries by Continent and Area

Click on the name of a continent:
- Africa
- Antarctica
- Asia
- Australia/Oceania
- Europe
- North America
- South America

Countries:
- China
- India
- Kazakhstan
- Saudi Arabia
- Indonesia
- Iran
- Mongolia
- Pakistan
- Turkey
- Myanmar
Example 3: Partial Code

Dim nations(192) As Nation  'declare array
Dim line, data() As String   'fill with UN.txt
Dim countries() As String =
    IO.File.ReadAllLines("UN.txt")
For i As Integer = 0 To 192
    line = countries(i)
    data = line.Split("","c")
    nations(i).name = data(0)
    nations(i).continent = data(1)
    nations(i).population = CDbl(data(2))
    nations(i).area = CDbl(data(3))
Next
Example 3: More Partial Code

Dim selectedContinent As String = lstContinents.Text

Dim query = From country In nations
    Where country.continent = selectedContinent
    Order By country.area Descending
    Select country.name

For Each countryName In query
    lstCountries.Items.Add(countryName)
Next
Structure College

Structure College

Dim name As String
Dim state As String  'state abbreviation
Dim yearFounded As Integer

End Structure
U.S. Colleges founded before 1800
3 fields (name, state, year founded)

Sample records

Harvard U., MA, 1636
William and Mary, VA, 1693
Yale U., CT, 1701
U. of Pennsylvania, PA, 1740
DataGridView Control

• Useful when two or more pieces of information are to be displayed.
• Found in the Data group and the All Windows Forms group of the Toolbox.
• Displays a table with column headers.
DataGridView Control (continued)

![DataGridView Control Image]

DataGridview control
DataSource Method

When the Select clause of a query contains two or more items, the pair of statements

```csharp
dgvGrid.DataSource = queryName.ToList
dgvGrid.CurrentCell = Nothing
```

displays the items of data in a DataGridView control. (The second statement, which is optional, keeps all cells unhighlighted.)
DataGridView Headers

- By default the rows have blank headers and the column headers contain the names of the items in the Select clause.
DataGridView Headers (cont.)

- Row headers can be removed by setting the RowHeadersVisible property of the DataGridView control to False.
- A column header can be customized with a statement such as

  ```csharp
dgvGrid.Columns("yearFounded").HeaderText = "Year Founded"
  ```

(see next slide)
DataGridview Headers (cont.)

![Earliest Colleges Window](image)

<table>
<thead>
<tr>
<th>College</th>
<th>Year Founded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dickinson College</td>
<td>1773</td>
</tr>
<tr>
<td>Moravian College</td>
<td>1742</td>
</tr>
<tr>
<td>U. of Pennsylvania</td>
<td>1740</td>
</tr>
<tr>
<td>U. of Pittsburgh</td>
<td>1787</td>
</tr>
<tr>
<td>Wash. &amp; Jefferson</td>
<td>1781</td>
</tr>
</tbody>
</table>
Searching an Array of Structures

The Where clause of a query can be used to locate specific items.

Example:

```vbnet
Dim query = From institution In colleges
             Where institution.name = lstColleges.Text
             Select institution

        txtState.Text = query.First.state
        txtYear.Text = CStr(query.First.yearFounded)
```
Another Structure

Structure Grades

Dim exam1 As Double
Dim exam2 As Double
Dim final As Double

End Structure
Using General Procedures with Structures

A variable having a structure as data type can be passed to a Function or Sub procedure.

Example:

```vba
Function CurveGrades(scores As Grades)
    As Grades
    scores.exam1 += 3
    scores.exam2 += 4
    scores.final += 2
    Return scores
End Function
```
Complex Structures

Member Types

• Integer, String, Double, etc.
• Another User Defined Type
• Array
  • Must not specify range
  • Range must be set using ReDim
Example 7

This example gathers information about a student and determines when the student will be eligible to graduate.
Example 7

Structure **FullName**
   Dim `firstName` As String
   Dim `lastName` As String
End Structure

Structure **Student**
   Dim `name` As **FullName**
   Dim `credits`() As Integer
End Structure

Private Sub `btnGet_Click(...) Handles _
   btnGet.Click`
   Dim `numYears` As Integer
   Dim `person` As **Student**
End Sub
Example 7 (continued)

```vbnet
txtResult.Clear()
person.name.firstName = InputBox("First Name:")
person.name.lastName = InputBox("Second Name:")
numYears = CInt(InputBox("Number of years " & "completed:"))
ReDim person.credits(numYears - 1)
For i As Integer = 0 To (numYears - 1)
    person.credits(i) = CInt(InputBox("Credits in year " & (i + 1)))
Next
DetermineStatus(person)
End Sub
```
Sub DetermineStatus(pupil As Student)
    Dim total As Integer = 0
    For i As Integer = 0 To pupil.credits.Count - 1
        total += pupil.credits(i)
    Next
    If (total >= 120) Then
        txtResult.Text = pupil.name.firstName & " " & pupil.name.lastName & " has enough credits"
    Else
        txtResult.Text = pupil.name.firstName & " " & pupil.name.lastName & " needs " & (120 - total) & " more credits to graduate."
    End If
End Sub
7.4 Two-Dimensional Arrays

- Declaring a Two-Dimensional Array Variable
- Implicit Array Sizing and Initialization
- The ReDim Statement
- Filling a Two-Dimensional Array with a Text File
- Using LINQ with Two-Dimensional Arrays
Declaring a Two-Dimensional Array Variable

- One-dimensional arrays store a list of items of the same type.
- Two-dimensional arrays store a table of items of the same type.
- Consider the rows of the table as numbered 0, 1, 2, ..., m and the columns numbered 0, 1, 2, ..., n. Then the array is declared with
  \[
  \text{Dim } \text{arrayName}(m, n) \text{ As DataType}
  \]

  Item in \(i\)th row, \(j\)th column: \(\text{arrayName}(i,j)\)
Implicit Array Sizing and Initialization

Arrays can be initialized when they are declared.

```dim arrayName(,) as DataType =
    {{ROW0}, {ROW1}, {ROW2}, ..., {ROWN}}
```

declares a two-dimensional array where ROW0 consists of the entries in the top row of the corresponding table delimited by commas, and so on.
# Road-Mileage Table

<table>
<thead>
<tr>
<th></th>
<th>Chicago</th>
<th>LA</th>
<th>NY</th>
<th>Philly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>0</td>
<td>2054</td>
<td>802</td>
<td>738</td>
</tr>
<tr>
<td>LA</td>
<td>2054</td>
<td>0</td>
<td>2786</td>
<td>2706</td>
</tr>
<tr>
<td>NY</td>
<td>802</td>
<td>2786</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Philly</td>
<td>738</td>
<td>2706</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>
Road-Mileage Array

```vbnet
Dim rm(,) As Double = {{0, 2054, 802, 738},
                     {2054, 0, 2786, 2706},
                     {802, 2786, 0, 100},
                     {738, 2706, 100, 0}}
```

declares and initializes an array of road-mileages. Some elements of the array are

\[
rm(0,0)=0, \quad rm(0,1)=2054, \quad rm(1,2)=2786
\]
GetUpperBound Method

After execution of the statement

```vbnet
Dim arrayName(r, s) As varType
```

the value of `arrayName.GetUpperBound(0)` is \( r \), and the value of `arrayName.GetUpperBound(1)` is \( s \).
Notes on Two-Dimensional Arrays

An unsized two-dimensional array can be declared with a statement of the form

```
Dim arrayName(,) As varType
```
ReDim and Two-Dimensional Arrays

• An already-created array can be resized with
  \[
  \text{ReDim } \text{arrayName}(r, s)
  \]
  which loses the current contents, or with
  \[
  \text{ReDim Preserve } \text{arrayName}(r, s)
  \]
• When \textit{Preserve} is used, only the columns can be resized.
• ReDim cannot change the number of dimensions in the array.
Filling a Two-Dimensional Array with a Text File

Distances.txt
0,2054,802,738
2054,0,2786,2706
802,2786,0,100
738,2706,100,0
Filling a Two-Dimensional Array with a Text File (cont.)

Dim rm(3, 3) As Double    'road mileage
Dim rowOfNums() As String =
    IO.File.ReadAllLines("Distances.txt")

Dim line, data() As String
For i As Integer = 0 To 3
    line = rowOfNums(i)
    data = line.Split(""
    For j As Integer = 0 To 3
        rm(i, j) = CDbll(data(j))
    Next
Next
LINQ with 2-Dimensional Array

Let $nums()$ be a 2 dimensional array of type Double having $m$ rows and $n$ columns.

```csharp
Dim query = From num In nums.Cast(Of Double)()
            Select num
```

produces a sequence of the $m \cdot n$ numbers in the array.

Can apply array methods to the sequence and can display it in a list box with the DataSource property.