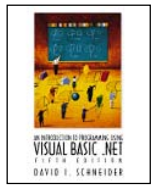
Chapter 7: Arrays

- Creating and Accessing Arrays (7.1)
- Using Arrays (7.2)
- Some Additional Types of Arrays (7.3)
- Sorting and Searching (7.4)
- Two-Dimensional Arrays (7.5)



David I. Schneider, *An Introduction to Programming using Visual Basic.NET, 5th Edition*, Prentice Hall, 2002.

Creating and Accessing Arrays (7.1)

- ReDim Statement
- Using an Array as a Frequency Table

Creating and Accessing Arrays (7.1) (cont.)

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

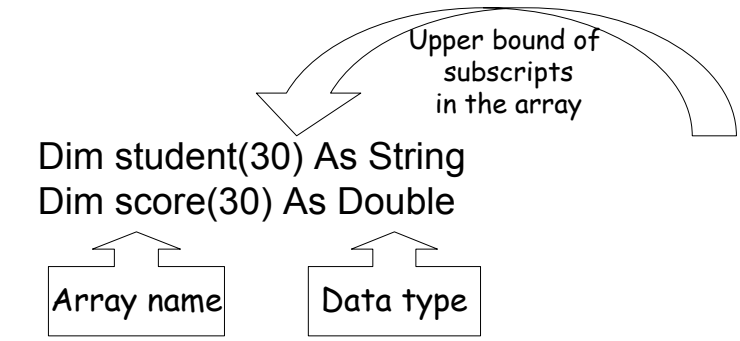
Creating and Accessing Arrays (7.1) (cont.)

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

```
Private Sub btnDisplay_Click(...)
    Handles btnDisplay.Click
  Dim student1 As String, score1 As Double
  Dim student2 As String, score2 As Double
  Dim student3 As String, score3 As Double
```

Creating and Accessing Arrays (7.1) (cont.)

- Using Arrays



Creating and Accessing Arrays (7.1) (cont.)

- Putting Values into an Array

```
student(1) = "Tom Brown"
```

Read: "student sub one equals Tom Brown"

Which means that the string "Tom Brown" is being stored at the second location in the array called student… because all arrays begin counting at 0.

Creating and Accessing Arrays (7.1) (cont.)

- 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 is the *size* of the array

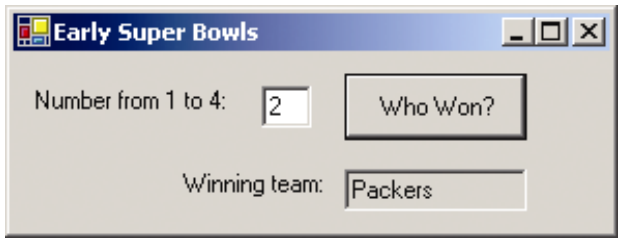
Creating and Accessing Arrays (7.1) (cont.)

- Example 1

```
Private Sub btnWhoWon_Click(...) _
    Handles btnWhoWon.Click
  Dim teamName(4) As String
  Dim n As Integer
  'Place Super Bowl Winners into the array
  teamName(1) = "Packers"
  teamName(2) = "Packers"
  teamName(3) = "Jets"
  teamName(4) = "Chiefs"
  'Access array
  n = CInt(txtNumber.Text)
  txtWinner.Text = teamName(n)
End Sub
```

Creating and Accessing Arrays (7.1) (cont.)

- Output Example 1



Creating and Accessing Arrays (7.1) (cont.)

- Example 2

```
Dim teamName(4) As String
Private Sub btnWhoWon_Click(...)Handles btnWhoWon.Click
  Dim n As Integer
  n = CInt(txtNumber.Text)
  txtWinner.Text = teamName(n)
End Sub

Private Sub Form1_Load(...) Handles MyBase.Load
  'Place Super Bowl Winners into the array
  teamName(1) = "Packers"
  teamName(2) = "Packers"
  teamName(3) = "Jets"
  teamName(4) = "Chiefs"
End Sub
```

Creating and Accessing Arrays (7.1) (cont.)

- Initializing Arrays
  - Arrays may be initialized when they are created:

```
Dim arrayName() As varType = {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*).

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

Creating and Accessing Arrays (7.1) (cont.)

- Using an Array as a Frequency Table

```
Private Sub btnAnalyze_Click(...) Handles btnAnalyze.Click
'Count occurrences of the various letters in a sentence
  Dim index, letterNum As Integer
  Dim sentence, letter As String
  Dim charCount(26) As Integer
  'Examine and tally each letter of the sentence
  sentence = (txtSentence.Text).ToUpper
  For letterNum = 1 To sentence.Length
    letter = sentence.Substring(letterNum - 1, 1)
    If (letter >= "A") And (letter <= "Z") Then
      index = Asc(letter) - 64 'The ANSI value of "A" is 65
      charCount(index) += 1
    End If
  Next
```

Cell 0 of the array not being used

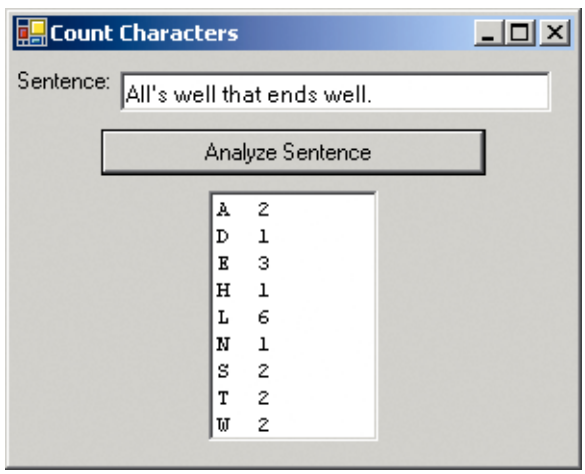
Creating and Accessing Arrays (7.1) (cont.)

- Example 4 (continued)

```
'List the tally for each letter of alphabet
  lstCount.Items.Clear()
  For index = 1 To 26
    letter = Chr(index + 64)
    If charCount(index) > 0 Then
      lstCount.Items.Add(letter & " " & _
                charCount(index))
    End If
  Next
End Sub
```

Creating and Accessing Arrays (7.1) (cont.)

- Example 4 Output



Creating and Accessing Arrays (7.1) (cont.)

- ReDim Statement
  - The size of an array may be changed after it is created:

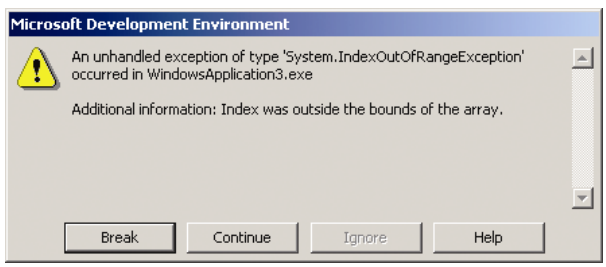
```
ReDim arrayName(m)
```

  - where *arrayName* is the name of the already declared array and *m* is an Integer literal,variable, or expression.

Creating and Accessing Arrays (7.1) (cont.)

- Out of Bounds Error
  - The following sets up an array, and then references an element that doesn't exist. This will cause an error.

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



Creating and Accessing Arrays (7.1) (cont.)

- Preserve
  - To keep any data that has already been stored in the array when resizing it, use

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

Creating and Accessing Arrays (7.1) (cont.)

- Copying Arrays
  - If *arrayOne*() and *arrayTwo*() have been declared with the same data type, then the statement

```
arrayOne = arrayTwo
```

  - makes *arrayOne*() an exact duplicate of *arrayTwo*(). It will have the same size and contain the same information.

Using Arrays (7.2)

- Ordered Arrays
- Using Part of an Array
- Merging Two Ordered Arrays
- Passing Arrays to Procedures

Using Arrays (7.2) (cont.)

- Ordered Arrays
  - An array is ordered if the elements are in ascending or descending order.

Using Arrays (7.2) (cont.)

- Example 1

```
Dim nom() As String = {"", "AL", "BOB", "CARL", "DON","ERIC", _
    "FRED", "GREG", "HERB", "IRA", "JACK"}

Private Sub btnSearch_Click(...) Handles btnSearch.Click
  Dim name, name2Find As String
  Dim n As Integer 'Subscript of the array
  name2Find = txtName.Text.ToUpper
  Do
    n += 1 'Add 1 to n
  Loop Until (nom(n) >= name2Find) Or (n = 10)
  If nom(n) = name2Find Then
    txtResult.Text = "Found."
  Else
    txtResult.Text = "Not found."
  End If
End Sub
```

Using Arrays (7.2) (cont.)

- Output Example 1

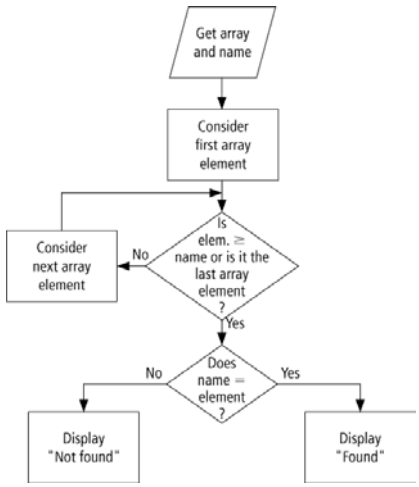
Search for the string “Don” in the array

Searching successive elements of an ordered list beginning with the first element is called a **sequential search**

Half of the array will be searched in average !

Using Arrays (7.2) (cont.)

- Flow Chart Search of Ordered Array



Using Arrays (7.2) (cont.)

- Using Part of an Array

```
'Demonstrate using part of an array
Dim stock(100) As String
Dim counter As Integer
Private Sub btnRecord_Click(...) Handles btnRecord.Click
  If (counter < 100) Then
    counter += 1 'Increment counter by 1
    stock(counter) = txtCompany.Text
    txtCompany.Clear()
    txtCompany.Focus()
    txtNumber.Text = CStr(counter)
  Else
    MsgBox("No space to record additional companies.", , "")
    txtCompany.Clear()
    btnsummarize.Focus()
  End If
End Sub
```

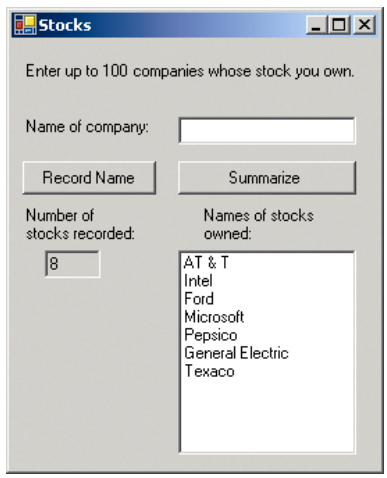
Using Arrays (7.2) (cont.)

- Example 2 (continued)

```
Private Sub btnsummarize_Click(...)
    Handles btnsummarize.Click
  'List stock companies that have been recorded
  Dim i As Integer
  lstStocks.Items.Clear()
  For i = 1 To counter
    lstStocks.Items.Add(stock(i))
  Next
End Sub
```

Using Arrays (7.2) (cont.)

- Example 2 output



Using Arrays (7.2) (cont.)

- Merging Two Ordered Arrays

To consolidate two lists into a single ordered list:

1. Compare the two names at the top of the first and second lists.
   a) If one name alphabetically precedes the other, copy it onto the third list and cross it off its original list.
   b) If the names are the same, copy the name onto the third list and cross out the name from the first and second lists.
2. Repeat Step 1 with the current top names until you reach the end of either list.
3. Copy the names from the remaining list onto the third list.

Using Arrays (7.2) (cont.)

- 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

Using Arrays (7.2) (cont.)

- Example 4

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim score() As Integer = {0, 85, 92, 75, 68, 84, 86, _
                            94, 74, 79, 88}
  txtAverage.Text = CStr(Sum(score) / 10)
End Sub

Function Sum(ByVal s() As Integer) As Integer
  Dim total, index As Integer
  total = 0
  For index = 1 To s.GetUpperBound(0) 'The upper bound of an
   'array = 10
    total += s(index)
  Next
  Return total
End Function
```

Using Arrays (7.2) (cont.)

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

```
Private Sub btnDisplay_Click(...) Handles
  btnDisplay.Click
  Dim num(20) As Integer
  num(5) = 10
  lstOutput.Items.Add(Triple(num(5)))
End Sub
Private Function Triple(ByVal x As Integer) As Integer
  Return 3 * x
End Function
```

Some Additional Types of Arrays (7.3)

- Control Arrays
- Array of Structures

Some Additional Types of Arrays (7.3)

- Control Arrays
  - Control arrays are arrays of controls, such as labels, text boxes, etc.
  - They are created in much the same way as any other array:

```
Dim arrayName(n) As ControlType
```

or

```
Dim arrayName() As ControlType
```

Some Additional Types of Arrays (7.3)

- Control Arrays continued
  - The following statements declare control arrays.

```
Dim lblTitle(10) As Label
Dim txtNumber(8) As TextBox
Dim btnAmount() As Button
```

Some Additional Types of Arrays (7.3)

- Example 1

```
Dim lblDept(5) As Label
Dim txtDept(5) As TextBox
Private Sub Form1_Load(...) Handles MyBase.Load
  Dim depNum As Integer
  lblDept(1) = Label1
  lblDept(2) = Label2
  lblDept(3) = Label3
  lblDept(4) = Label4
  lblDept(5) = Label5
  txtDept(1) = TextBox1
  txtDept(2) = TextBox2
  txtDept(3) = TextBox3
  txtDept(4) = TextBox4
  txtDept(5) = TextBox5
```

Array of controls

Placing controls Into arrays

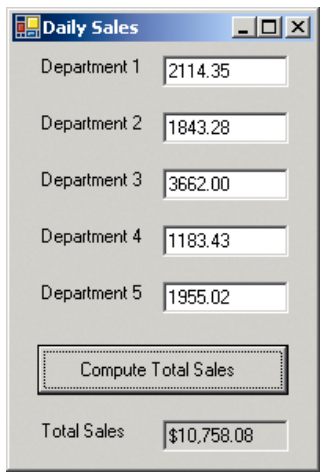
Some Additional Types of Arrays (7.3)

- Example 1 continued

```
  For depNum = 1 To 5
    lblDept(depNum).Text = "Department " & depNum
    txtDept(depNum).Clear()
  Next
End Sub
Private Sub btnCompute_Click(...)
    Handles btnCompute.Click
  Dim totalSales As Double = 0
  Dim depNum As Integer
  For depNum = 1 To 5
    totalSales += CDbl(txtDept(depNum).Text)
  Next
  txtTotal.Text = FormatCurrency(totalSales)
End Sub
```

Some Additional Types of Arrays (7.3)

- Example 1 Output



Some Additional Types of Arrays (7.3)

- Structures
  - A way of grouping heterogeneous data together
  - Also called a UDT (User Defined Type)
  - Sample structure definition:

```
Structure College
  Dim name As String
  Dim state As String
  Dim yearFounded As Integer
End Structure
```

Some Additional Types of Arrays (7.3)

- Structure Definition
  - Each sub-variable in a structure is called a **member or field**
  - To declare a variable of a structure type:

```
Dim college1 As College
```

  - Each member is accessed via the variable name dot member name

```
college1.name = "Harvard"
```

Some Additional Types of Arrays (7.3)

- Example 2

```
Structure College
  Dim name As String
  Dim state As String
  Dim yearFounded As Integer
End Structure
Dim college1, college2, collegeOlder As College
Private Sub btnFirst_Click(...) Handles btnFirst.Click
  Dim prompt As String
  college1.name = InputBox("Enter name of first
  college.", "Name")
  college1.state = InputBox("Enter state of first
  college.", "State")
  prompt = "Enter the year the first college was
  founded."
  college1.yearFounded = CInt(InputBox(prompt, "Year"))
End Sub
```

Example 2 (cont'd)

```
Private Sub btnFirst_Click(...) Handles
  btnFirst.Click
  Dim prompt As String
  college2.name = InputBox("Enter name of second
  college.", "Name")
  college2.state = InputBox("Enter state of
  second college.", "State")
  prompt = "Enter the year the second college
  was founded."
  college2.yearFounded = CInt(InputBox(prompt,
  "Year"))
End Sub
```

- Example 2 (cont'd)

```
Private Sub btnOlder_Click(…) Handles
  btnOlder.Click
If (college1.yearFounded < college2.yearFounded
  Then
collegeOlder = college1
Else
collegeOlder = college2
End If
txtResult.Text = collegeOlder.name & " was founded
  in " & _
College.Older.state & " in " &
  collegeOlder.yearFounded
End Sub
```

Some Additional Types of Arrays (7.3)

- Structure Members
  - Integer, String, Double, etc.
  - Another User Defined Type
  - Arrays
    - ❖Must not specify range
    - ❖Range must be set using ReDim

Some Additional Types of Arrays (7.3)

- Example 4

```
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, i As Integer
  Dim person As Student
```

Structure "FullName" contained, or nested, inside Student

Some Additional Types of Arrays (7.3)

- Example 4 continued

```
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)
  For i = 1 To numYears
    person.credits(i) = CInt(InputBox("Credits in
  yr "_
                               & i))
  Next
  DetermineStatus(person)
End Sub
```

Some Additional Types of Arrays (7.3)

- Example 4 continued

```
Sub DetermineStatus(ByVal person As Student)
  Dim i, total As Integer
  For i = 1 To person.credits.GetUpperBound(0)
    total += person.credits(i)
  Next
  If (total >= 120) Then
    txtResult.Text = person.name.firstName & " " & _
       person.name.lastName & " has enough credits" & _
       " to graduate."
  Else
    txtResult.Text = person.name.firstName & " " & _
       person.name.lastName & " needs " & _
       (120 - total) & " more credits to graduate."
  End If
End Sub
```

Sorting and Searching (7.4)

- Bubble Sort
- Shell Sort
- Searching

Sorting and Searching (7.4) (cont.)

- Sorting
  - Sorting is an algorithm for ordering an array.
  - We discuss two sorting algorithms:
    - **bubble sort**
    - **Shell sort**
  - Both use the swap algorithm:

```
temp = var1
var1 = var2
var2 = temp
```

Sorting and Searching (7.4) (cont.)

- Example 1 Swap Algorithm

```
Private Sub btnAlphabetize_Click(...) _
    Handles btnAlphabetize.Click
  Dim firstWord, secondWord, temp As String
  firstWord = txtFirstWord.Text
  secondWord = txtSecondWord.Text
  If (firstWord > secondWord) Then
    temp = firstWord
    firstWord = secondWord
    secondWord = temp
  End If
  txtResult.Text = firstWord & " before " & _
                   secondWord
End Sub
```

Sorting and Searching (7.4) (cont.)

- Bubble Sort Algorithm

1. Compare the first and second items. If they are out of order, swap them.
2. Compare the second and third items. If they are out of order, swap them.
3. Repeat this pattern for all remaining pairs. The final comparison and possible swap are between the next-to-last and last elements.

Class exercise for sorting numbers

Sorting and Searching (7.4) (cont.)

- Shell Sort Algorithm

1. Begin with a gap of g = Int(n/2)
2. Compare items 1 and 1 + g, 2 and 2 + g, . . ., n and n - g. Swap any pairs that are out of order.
3. Repeat Step 2 until no swaps are made for gap g.
4. Halve the value of g.
5. Repeat Steps 2, 3, and 4 until the value of g is 0.

Class exercise for sorting numbers

Sorting and Searching (7.4) (cont.)

- Searching
  - Sequential search starts at the beginning of a list and keeps looking one by one until the item is found or the end of the list is reached.
  - For a sequential search, the list need not be sorted.

Sorting and Searching (7.4) (cont.)

- Binary Search
  - Usually more efficient than sequential search
  - List must be sorted

Sorting and Searching (7.4) (cont.)

- Algorithm for Binary Search

Assume the item sought is "quarry"

1. Denote the subscript of the first item in the list by first and the subscript of the last item by last. Initially, the value of first is 1, the value of last is the number of items in the list, and the value of flag is False.
2. Look at the middle item of the current list, the item having the subscript middle = Int((first + last) / 2)
3. If the middle item is quarry (what you are looking for), then flag is set to True and the search is over.
4. If the middle item is greater than quarry, then quarry should be in the first half of the list. So the subscript of quarry must lie between first and middle - 1. That is, the new value of last is middle - 1.

Sorting and Searching (7.4) (cont.)

- Algorithm for Binary Search continued

5. If the middle item is less than quarry, then quarry should be in the second half of the list of possible items. So the subscript of quarry must lie between middle +1 and last. That is, the new value of first is middle +1.
6. Repeat Steps 2 through 5 until quarry is found or until the halving process uses up the entire list. (When the entire list has been used up, first > last.) In the second case, quarry was not in the original list.

Class exercise to search a name in a sorted array of names

Two dimensional arrays (7.5)

- One-dimensional arrays require one subscript to access each element in the array.
- Two-dimensional arrays require two subscripts to access each element.

```
Dim rm(4, 4) As Double
```

Two dimensional arrays (7.5) (cont.)

- Populating a Two-Dimensional Array

```
Dim rm(4, 4) As Double
Private Sub Form1_Load(...) Handles MyBase.Load
'Fill two-dimensional array with intercity mileages
  Dim sr As IO.StreamReader =
            IO.File.OpenText("DISTANCE.TXT")
  Dim row, col As Integer
  For row = 1 To 4
    For col = 1 To 4
      rm(row, col) = CDbl(sr.ReadLine)
    Next
  Next
  sr.Close()
End Sub
```

Two dimensional arrays (7.5) (cont.)

- Displaying the road mileages between cities

```
Private Sub btnShow_Click(…) Handles btnShow.Click
Dim row, col As Integer
row = CInt(txtOrig.Text)
col = CInt(txtDest.Text)
 If (row >= 1 And row <=4) And (col >=1 And col
  <=4) Then
 txtMiles.Text = CStr(rm(row, col))
 Else
 MsgBox(Origin and Destination must be numbers from
  1 to 4", , "Error")
 End If
End Sub
```

Two dimensional arrays (7.5) (cont.)

- Notes on Two-Dimensional Arrays

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

Dim *arrayName*(,) As *varType*

and a two-dimensional array can be declared and initialized at the same time with a statement of the form

Dim *arrayName*(,) As *varType* = {{ROW$_0$}, {ROW$_1$},... {ROW$_m$}}

Two dimensional arrays (7.5) (cont.)

- ReDim and Two-Dimensional Arrays
  - An already-created array can be resized
    ```
    ReDim arrayName(r, s)
    ```
  - which loses the current contents, or with
    ```
    ReDim Preserve arrayName(m, s)
    ```
  - When **Preserve** is used, only the column can be resized.
  - ReDim cannot change the number of dimensions in an array.