Administrative

- Midterm

ADMINISTRATIVE

- Assignment #3
  - Console Application

- Writing to the console
  - Console.WriteLine("Writing to the console")
  - UserInput = Console.ReadLine()
    - Using Console.ReadLine will PAUSE the program until the user enters something.

Example Code

- Console.WriteLine(">Welcome _
  to the Gradebook program!")
- number = CInt(Console.ReadLine)

Chapter 7 - Arrays

Chapter 7 – Arrays

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

Simple and Array Variables

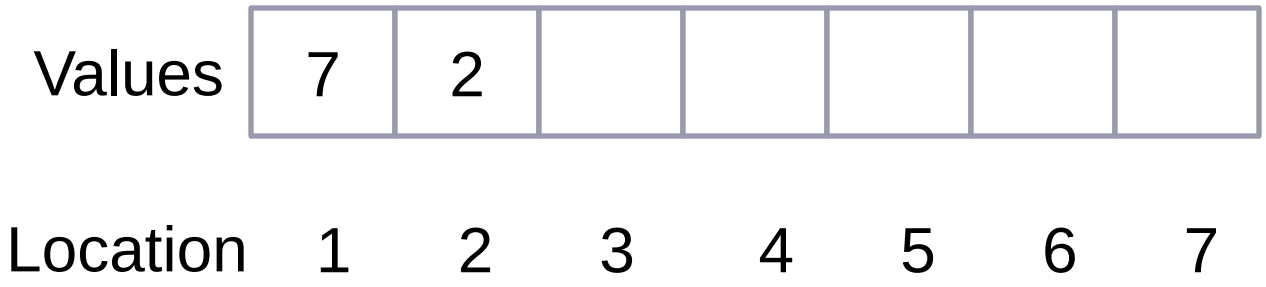
- A **variable** (or simple variable) is a name to which Visual Basic can assign a single value.

- An **array variable** is a *collection* of simple variables of the *same type* to which Visual Basic can efficiently assign a list of values.

Simple and Array Variables

- Many **variables**

- An **array variable**



Simple and Array Variables

Many **variables**

Vs.

An **array variable**

EXAMPLE

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

COULD DO

```
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
```

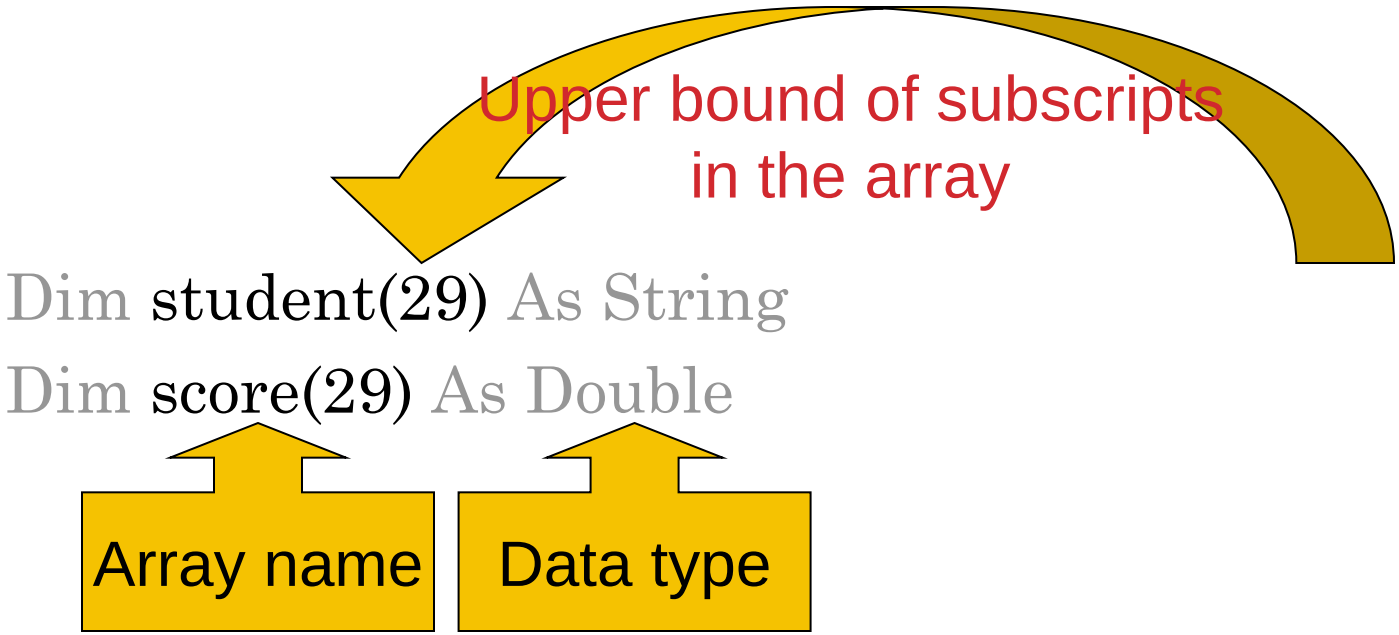
EXAMPLE

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

BETTER

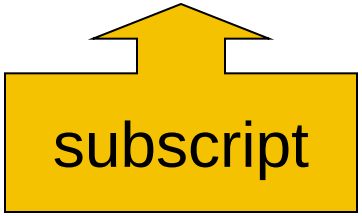
```
Private Sub btnDisplay_Click(...) _
                                Handles btnDisplay.Click
  Dim student(29) As String, score(29) As Double
```

Using Arrays



Putting Values into an Array

student(0) = "Tom Brown"



Read: "student sub zero equals Tom Brown"

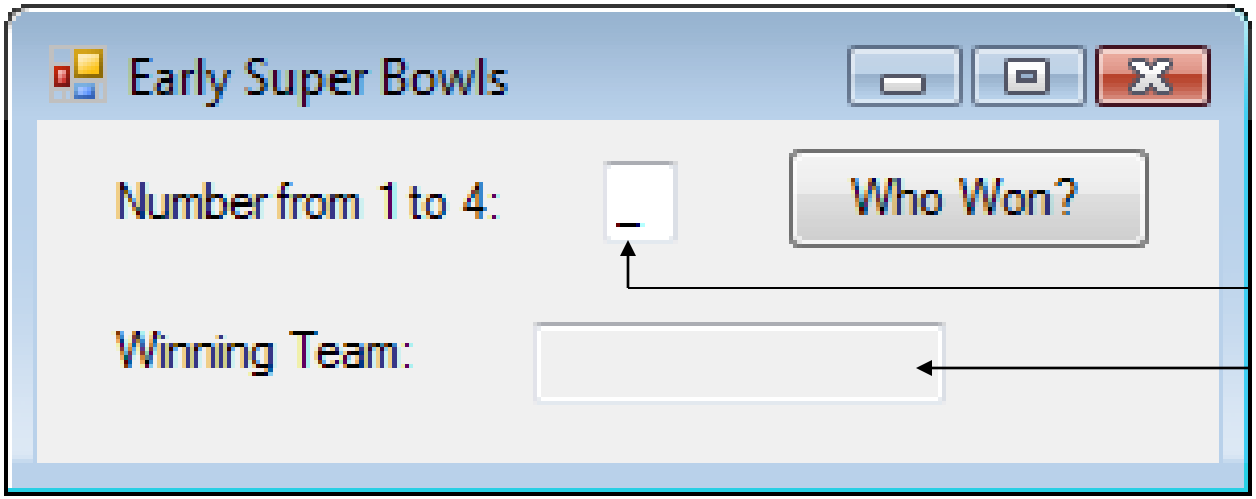
Which means that the string "Tom Brown" is being stored at the first location in the array called student… because all arrays begin counting at 0.

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

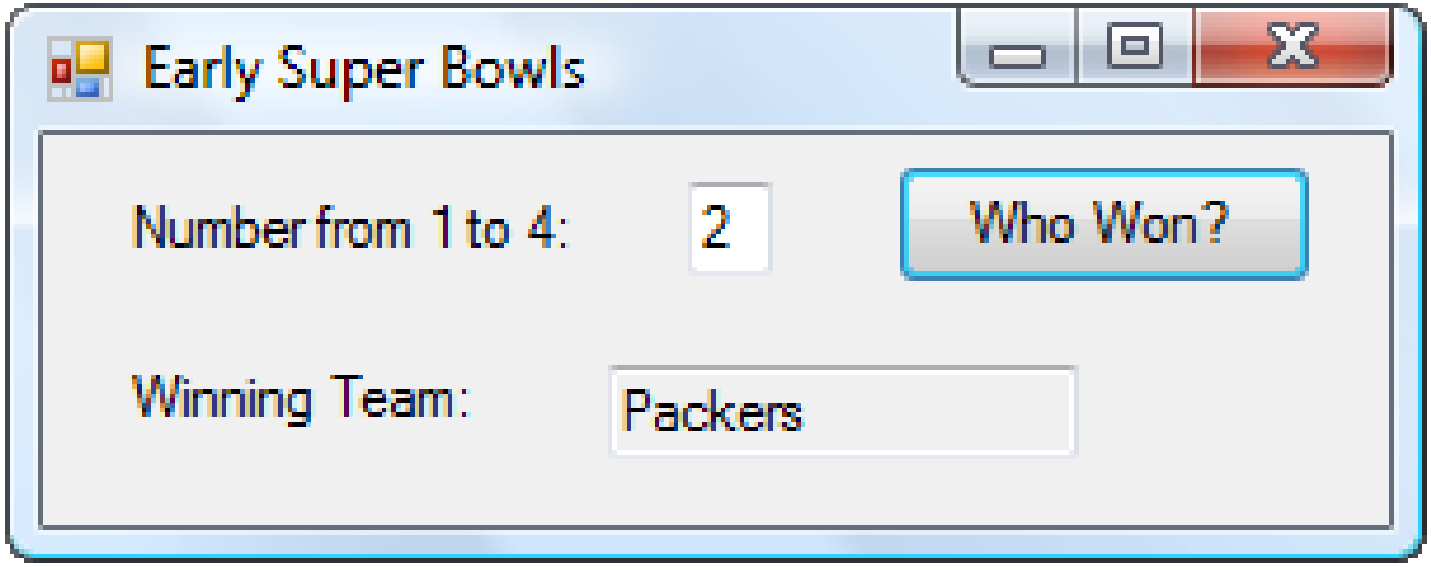
Example 1: Form



EXAMPLE 1

```
Private Sub btnWhoWon_Click(...) _
                                Handles btnWhoWon.Click
  Dim teamName(3) As String
  Dim n As Integer
  'Place Super Bowl Winners into the array
  teamName(0) = "Packers"
  teamName(1) = "Packers"
  teamName(2) = "Jets"
  teamName(3) = "Chiefs"
  'Access array
  n = CInt(txtNumber.Text)
  txtWinner.Text = teamName(n - 1)
End Sub
```

Example 1: Output



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.

- This is the FIRST thing that loads
- Loads BEFORE you see the form

Load Event Procedure

Occurs as the Form loads in memory

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

- Errors here are handled differently!!!
  - http://blog.paulbetts.org/index.php/2010/07/20/the-case-of-the-disappearing-onload-exception-user-mode-callback-exceptions-in-x64/
  - http://stackoverflow.com/questions/4933958/vs2010-does-not-show-unhandled-exception-message-in-a-winforms-application-on-a

- Write the code for a button
- When debugged, put it into MyBase.Load

EXAMPLE 2

```
Dim teamName(3) As String
Private Sub btnWhoWon_Click(...) Handles btnWhoWon.Click
  Dim n As Integer
  n = CInt(txtNumber.Text)
  txtWinner.Text = teamName(n - 1)
End Sub

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

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), ..., and *valueN* to *arrayName*(*N*).

Initializing Arrays

- Arrays may be initialized when they are created:

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

- For Ex:

```
Dim Students() As String = {"Jack",
     "John", "Julie", "Jimmy", "Janet"}
```

Initializing Arrays

- Arrays may be initialized when they are created:

```
Dim arrayName() As varType =
  IO.File.ReadAllLines(filespec)
```

- Opens *filespec*, reads all lines from it, and stores it in *arrayName*
- Each line in *filespec* is stored in one location of *arrayName*

GetUpperBound Method

The value of

***arrayName*.GetUpperBound(0)**

is the upper bound of *arrayName*().

EXAMPLE

```
Dim teamName() As String = {"Packers", _
                            "Packers", "Jets", "Chiefs"}
txtBox.Text = CStr(teamName.GetUpperBound(0))
```

Output: 3

EXAMPLE

```
Dim Grades() As integer = {70, 75, 80, 85, 90}
```

Grades.Average → 80

Grades.Count → 5

Grades.Min → 70

Grades.Max → 90

Grades.Sum → 400

ReDim Statement

The size of an array may be changed after it has been created.

```
ReDim arrayName(m)
```

changes the upper bound of the array to *m*.

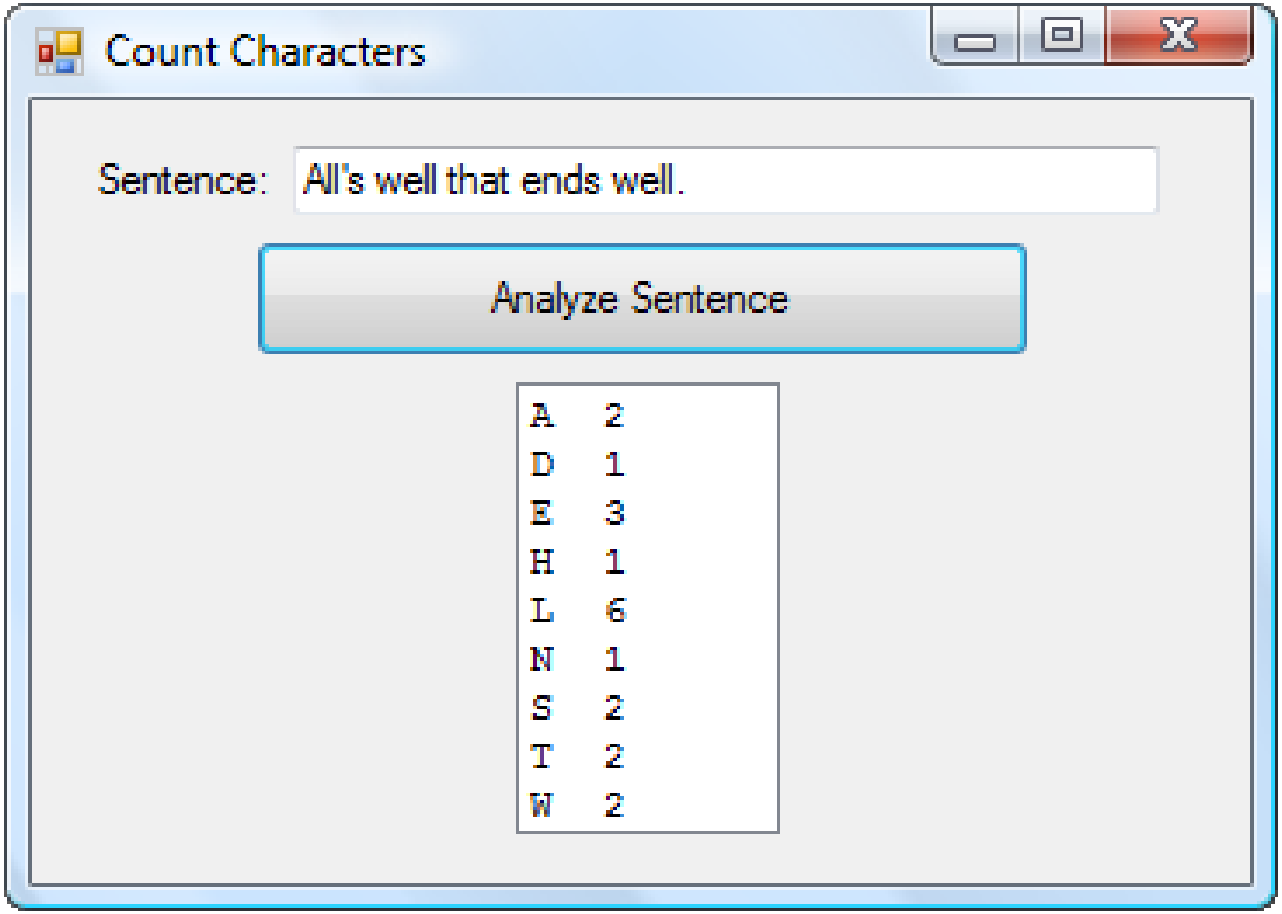
PRESERVE KEYWORD

`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.

Example 4: Using an Array as a Frequency Table



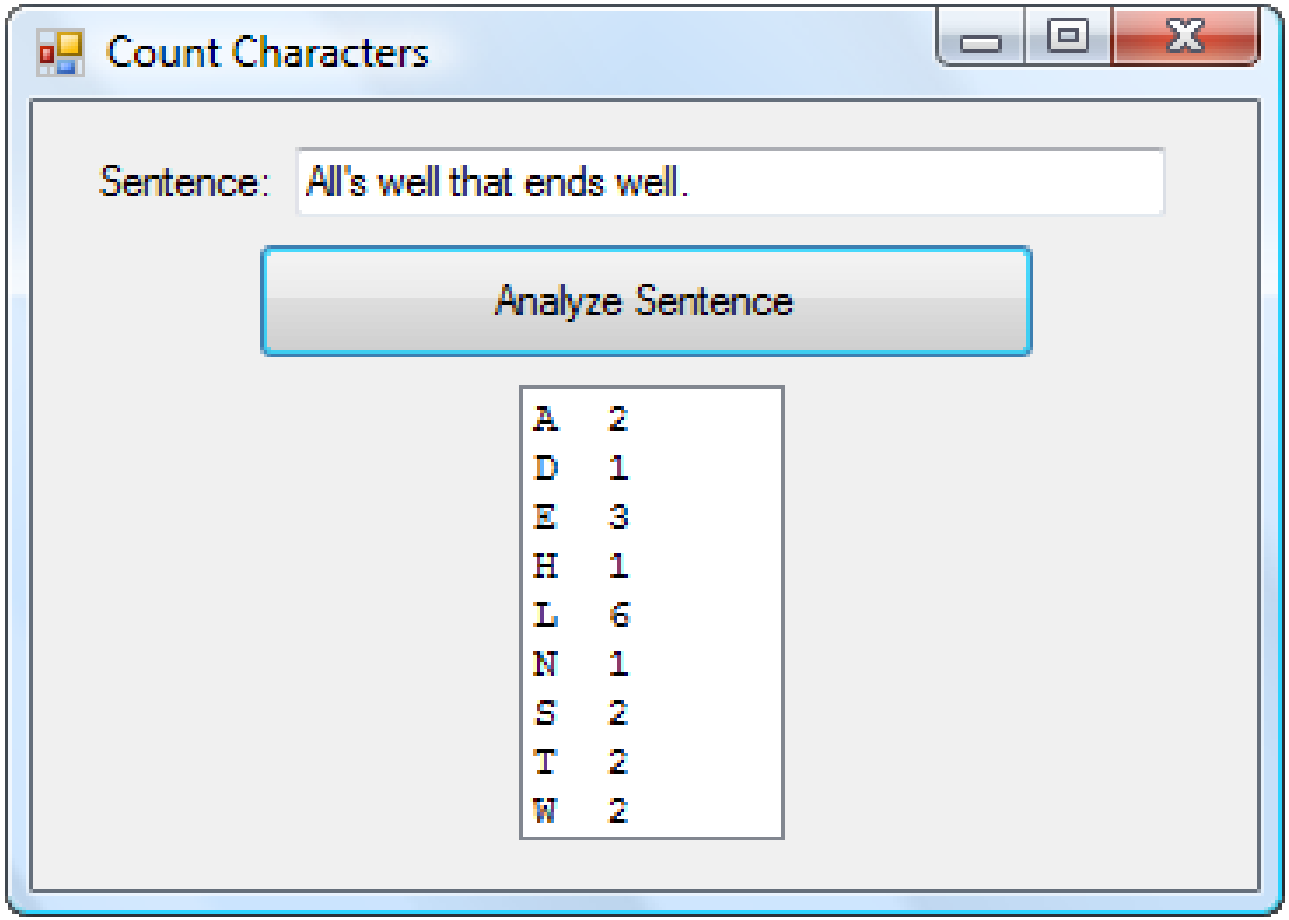
EXAMPLE 4: CODE

```
Private Sub btnAnalyze_Click(...) Handles btnAnalyze.Click
  'Count occurrences of the various letters in a sentence

  Dim sentence, letter As String
  Dim index, charCount(25) As Integer

  'Examine and tally each letter of the sentence
  sentence = (txtSentence.Text).ToUpper
  For letterNum As Integer = 1 To sentence.Length
   'Get a specific letter from the sentence
    letter = sentence.Substring(letterNum - 1, 1)
    If (letter >= "A") And (letter <= "Z") Then
      index = Asc(letter) - 65 'The ANSI value of "A" is 65
      charCount(index) += 1
    End If
  Next
```

Example 4: Code Continued

```
    'List the tally for each letter of alphabet
    lstCount.Items.Clear()
    For i As Integer = 0 To 25
      letter = Chr(i + 65)
      If charCount(i) > 0 Then
        lstCount.Items.Add(letter & " " & _
                                    charCount(i))
      End If
    Next
End Sub
```

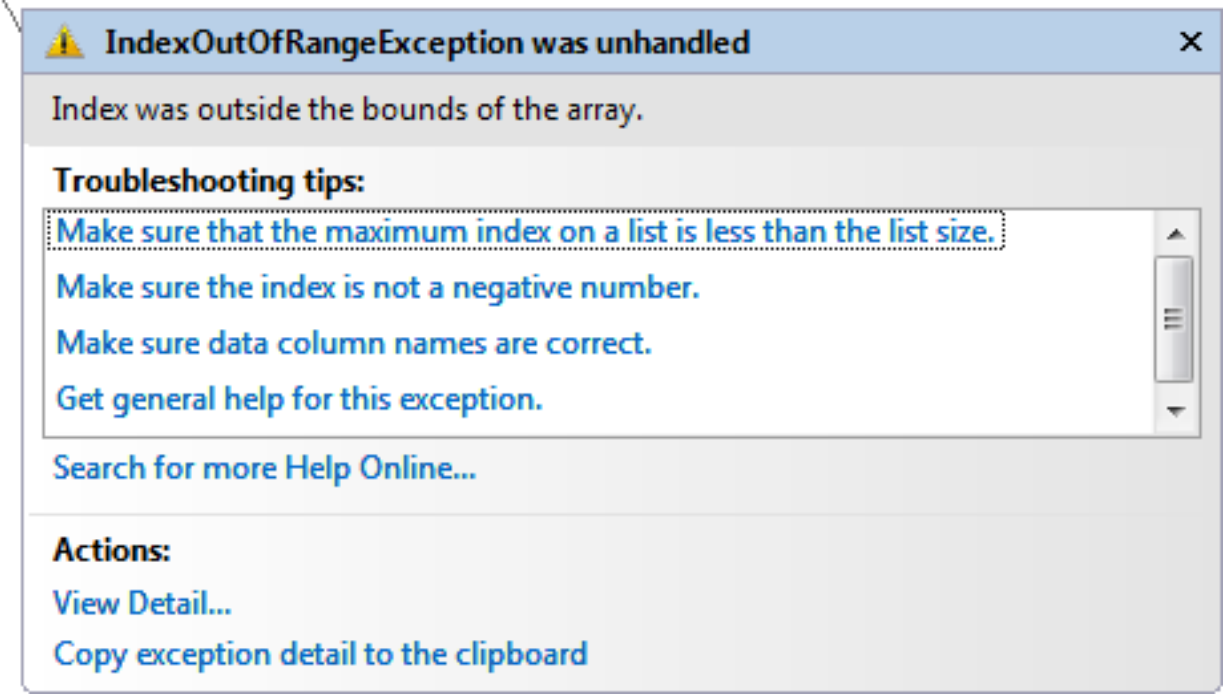
Example 4 Output



Out of Bounds Error

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

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



Assignment Statement for 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*().

Actually, they share the same location in memory

Assignment Statement for Arrays

- How to make an *independent* copy?

```
Array1 = Array2.clone
```

- Changing Array1 does not change Array2.

User-Defined Array-Valued Functions

Headers have the form

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

Erasing an Array

- An array can consume a large block of memory.
- After the array is no longer needed, we can release all memory allocated to the array by executing the following statement:

Erase *arrayName*

Using .SPLIT

```
'read the entire file in, 1 line per array entry
Dim arrayName() As String = IO.File.ReadAllLines("input.txt")

'iterate through all lines in the input file
For i = 0 To arrayName.GetUpperBound(0)

        'split all elements of the line into various entries
        Dim Stuff() As String
        Stuff = arrayName(i).Split(vbTab)

        MsgBox("HERE")

Next i
```

7.2 Using Arrays

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

Ordered Arrays

An array has **ascending order** if
[each element] ≤ [next element]

An array has **descending order** if
[each element] ≥ [next element]

An array is **ordered** if it has ascending or descending order

Searching Ordered Arrays

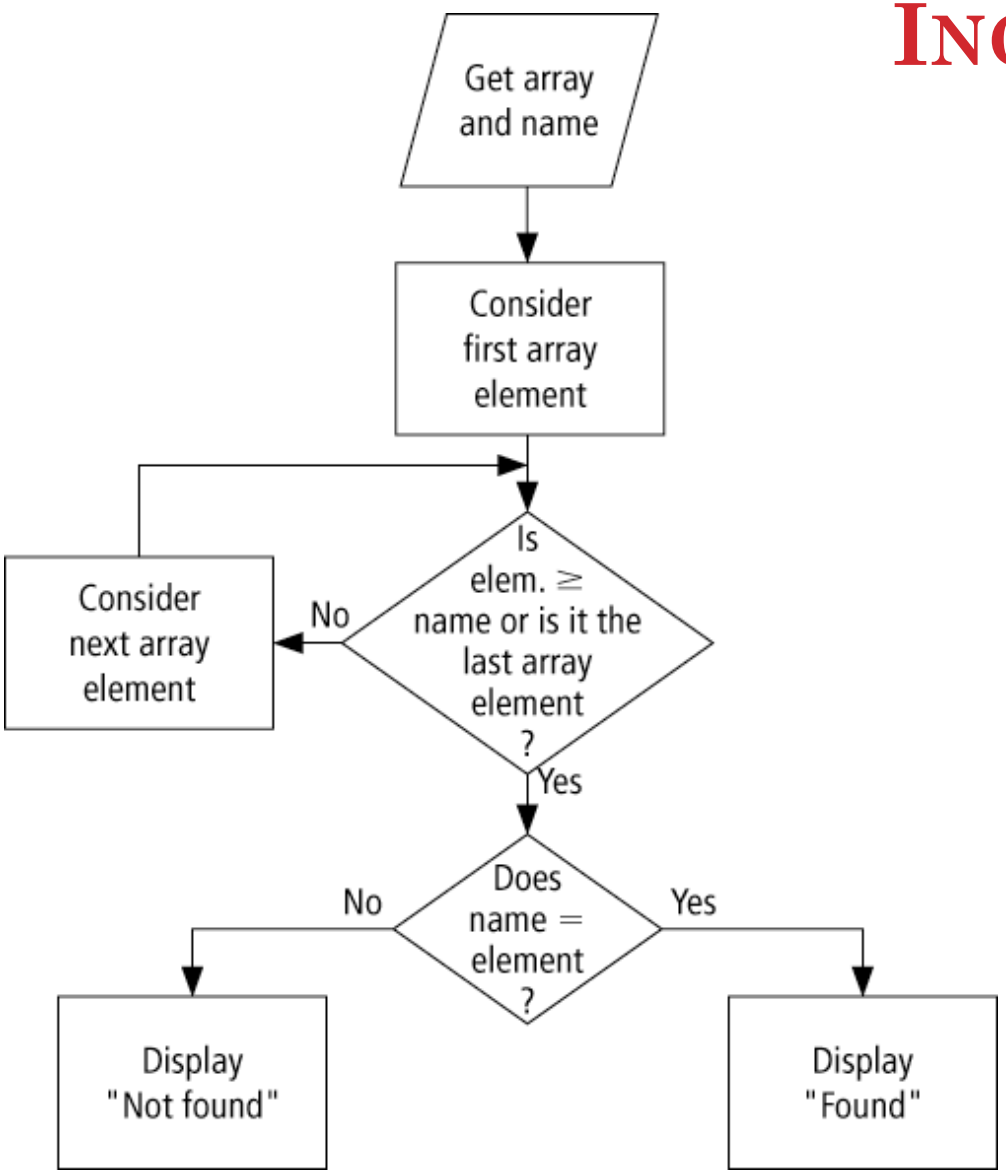
Ordered arrays can be searched more efficiently than unordered arrays

For instance, when searching an array having ascending order, you can terminate the search when you find an element whose value is ≥ the sought-after value.

Example 1: Task

Given a name input by the user, determine if it is in an increasing list of ten names

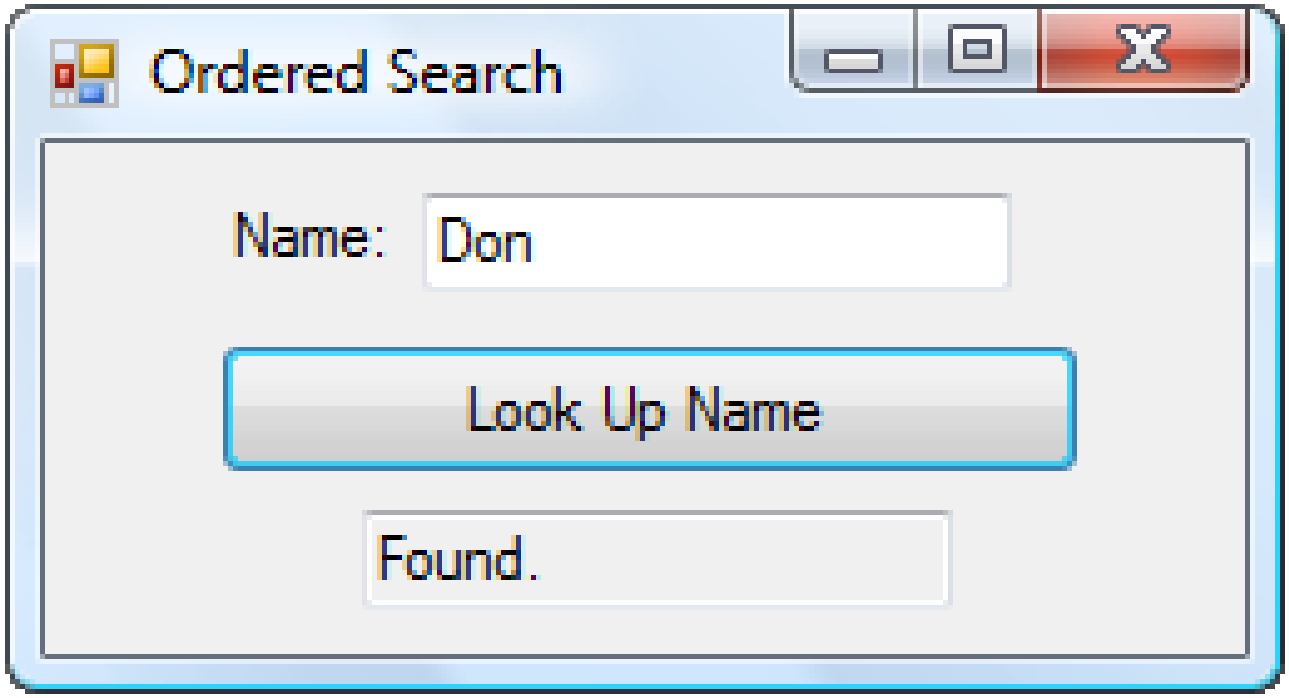
Flowchart for a Search of an Increasing Array



EXAMPLE 1: CODE

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

Private Sub btnSearch_Click(...) Handles btnSearch.Click
  Dim name2Find As String
  Dim n As Integer = -1    'Subscript of the array
  name2Find = txtName.Text.ToUpper
  Do
    n += 1      'Add 1 to n
  Loop Until (nom(n) >= name2Find) Or (n = 9)
  If nom(n) = name2Find Then
    txtResult.Text = "Found."
  Else
    txtResult.Text = "Not found."
  End If
End Sub
```

Example 1: Output



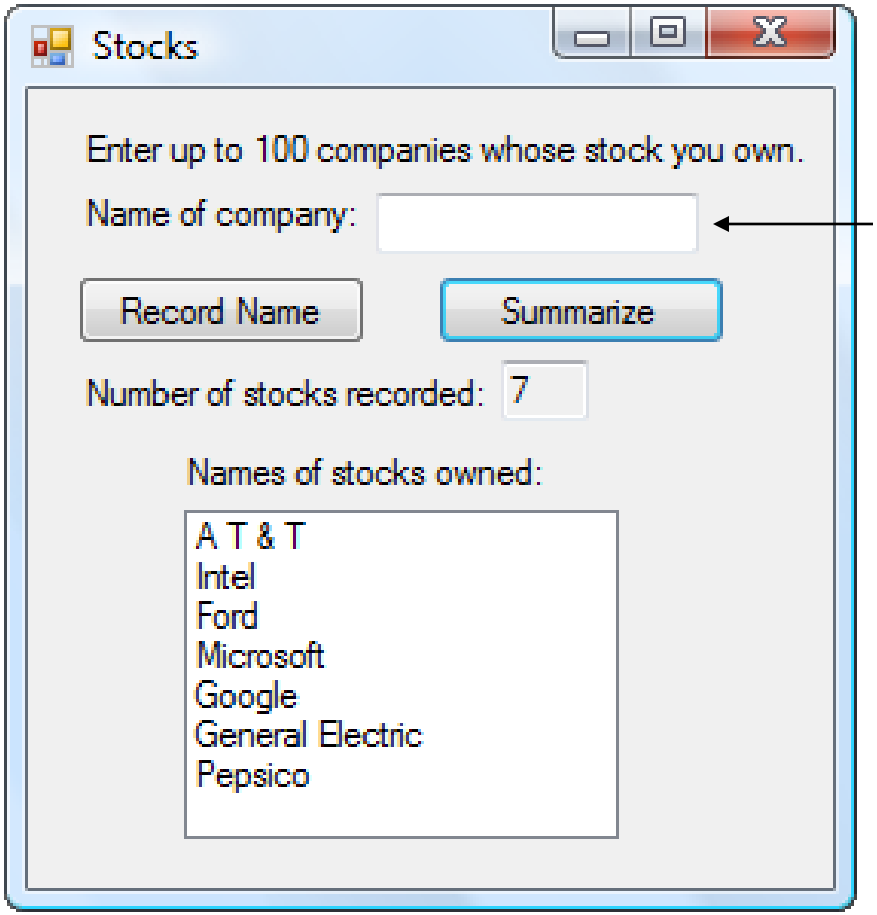
Using Part of an Array

- Sometimes we do not know how many elements will be needed in an array

- We can declare a large array, say of 100 elements, and use a counter variable to record the number of elements used

- In Example 2, the names are an unknown number of companies is placed into an array.

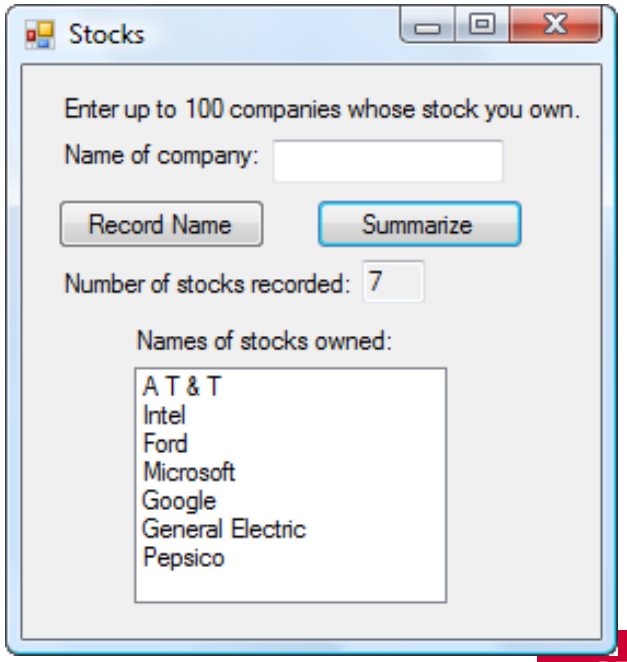
Example 2: Output



txtCompany

Example 2: Code

```
'Demonstrate using part of an array
Dim stock(99) As String
Dim counter As Integer
Private Sub btnRecord_Click(...) Handles btnRecord.Click
  If (counter < 99) Then
    counter += 1          'Increment counter by 1
    stock(counter - 1) = txtCompany.Text
    txtCompany.Clear()
    txtCompany.Focus()
    txtNumber.Text = CStr(counter)
  Else
    MessageBox.Show("No space to record _
           more companies.")
    txtCompany.Clear()
  End If
End Sub
```



Example 2: Code Continued

```
Private Sub btnSummarize_Click(...) _
                    Handles btnSummarize.Click
  'List companies that were recorded
   lstStocks.Items.Clear()
   For i As Integer = 0 To counter - 1
     lstStocks.Items.Add(stock(i))
   Next
End Sub
```

Merging Two Ascending Arrays

To consolidate the two lists into a single ordered third 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 into the third list.

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 header of the Sub of Function procedure uses the name with empty set of parentheses.

Example 4

- This example uses a Function procedure to add up the numbers in an array. The GetUpperBound method is used to determine how many numbers are in the array.

EXAMPLE 4

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim score() As Integer = {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 As Integer = 0
  For index As Integer = 0 To s.GetUpperBound(0)
    total += s(index)
  Next
  Return total
End Function
```

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
```

7.3 Some Additional Types of Arrays

- Control Arrays
- Array of Structures
- Displaying and Comparing Structure Values

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

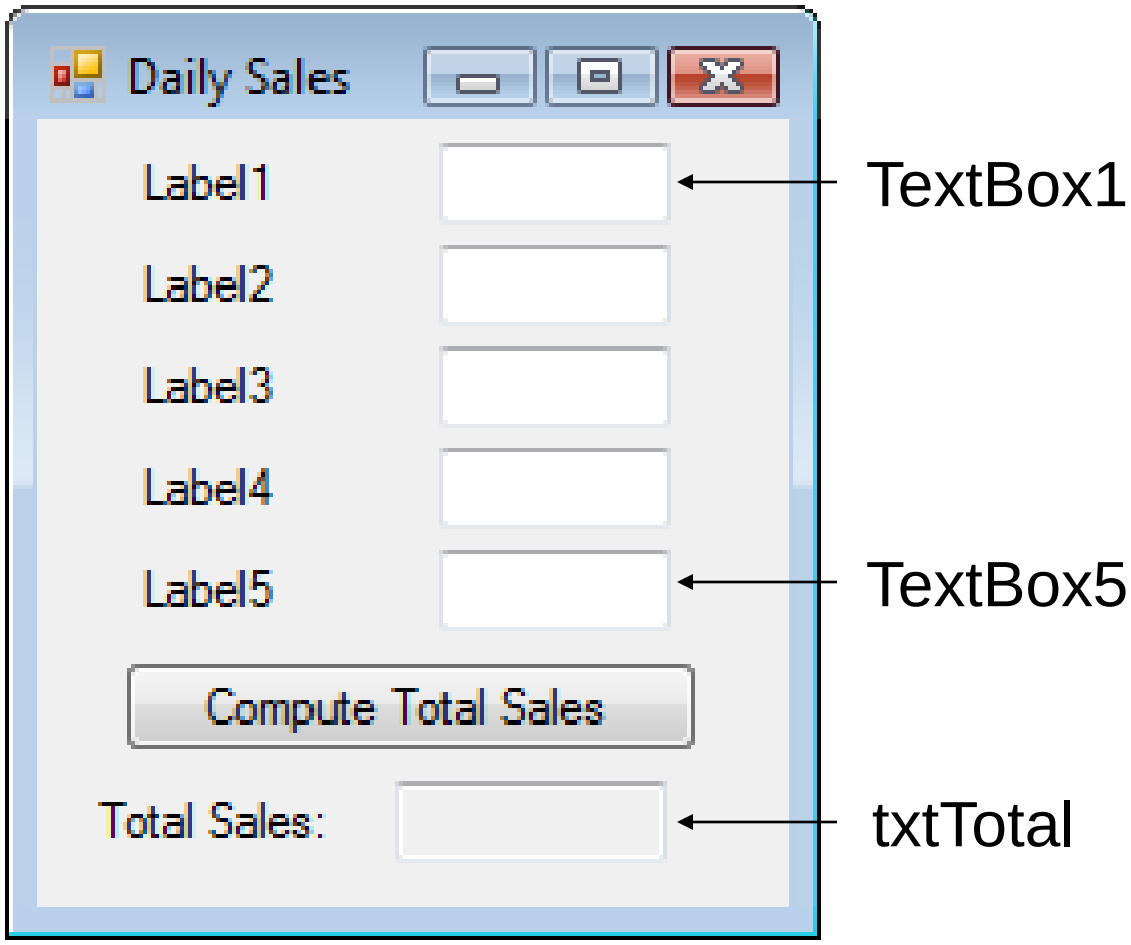
```
Dim arrayName() As ControlType
```

Control Arrays continued

- The following statements declare control arrays.

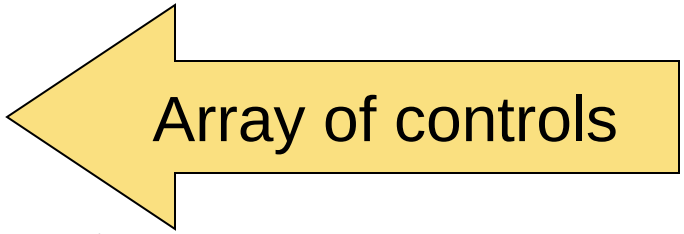
```
Dim lblTitle(10) As Label
Dim txtNumber(8) As TextBox
Dim btnAmount() As Button
```

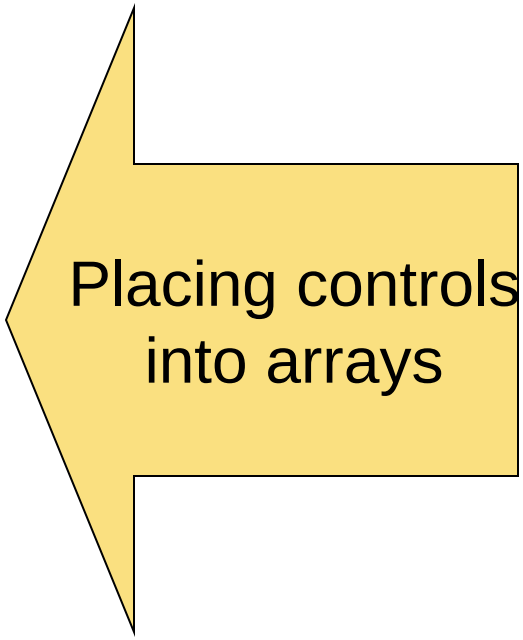
Example 2: Form



EXAMPLE 1

```
Dim lblDept(4) As Label
Dim txtDept(4) As TextBox
Private Sub frmSales_Load(...) Handles MyBase.Load
   lblDept(0) = Label1
   lblDept(1) = Label2
   lblDept(2) = Label3
   lblDept(3) = Label4
   lblDept(4) = Label5
   txtDept(0) = TextBox1
   txtDept(1) = TextBox2
   txtDept(2) = TextBox3
   txtDept(3) = TextBox4
   txtDept(4) = TextBox5
```

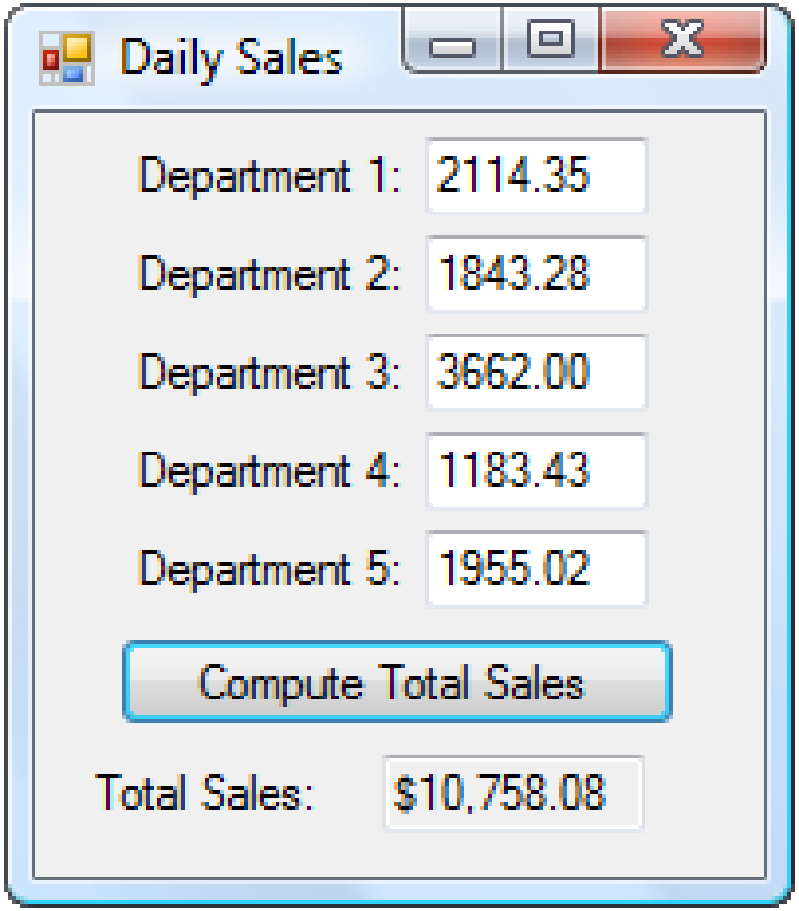




EXAMPLE 1 CONTINUED

```
Private Sub Clear() Handles btnClear.Click
  For depNum As Integer = 1 To 5
    lblDept(depNum - 1).Text = "Department " & depNum
    txtDept(depNum).Clear()
  Next
End Sub

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

Example 1 Output



Structures

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
```

Structure Definition

Each subvariable in a structure is called a **Member**

To declare a variable of a structure type:

```
Dim college1 As College
```

Each member is accessed via

variable name.member name

```
college1.state = "Maryland"
```

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 college.", "Name")
  college1.state = InputBox("Enter state.", "State")
  prompt = "Enter the year the first college was founded."
  college1.yearFounded = CInt(InputBox(prompt, "Year"))
End Sub
```

Structure Members

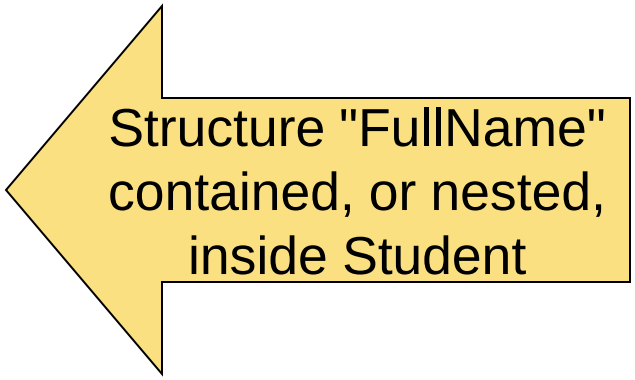
- Integer, String, Double, etc.
- Another User Defined Type
- Arrays
  - Must not specify range
  - Range must be set using ReDim

Example 4

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

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
```



EXAMPLE 4 CONTINUED

```
Private Sub btnGet_Click(...) Handles btnGet.Click
  Dim numYears As Integer
  Dim person As Student
  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
```

EXAMPLE 4 CONTINUED

```
Sub DetermineStatus(ByVal person As Student)
  Dim total As Integer = 0
  For i As Integer = 0 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
```

7.4 Sorting

7.4 Sorting and Searching

- Bubble Sort
- Shell Sort
- Searching

Sorting

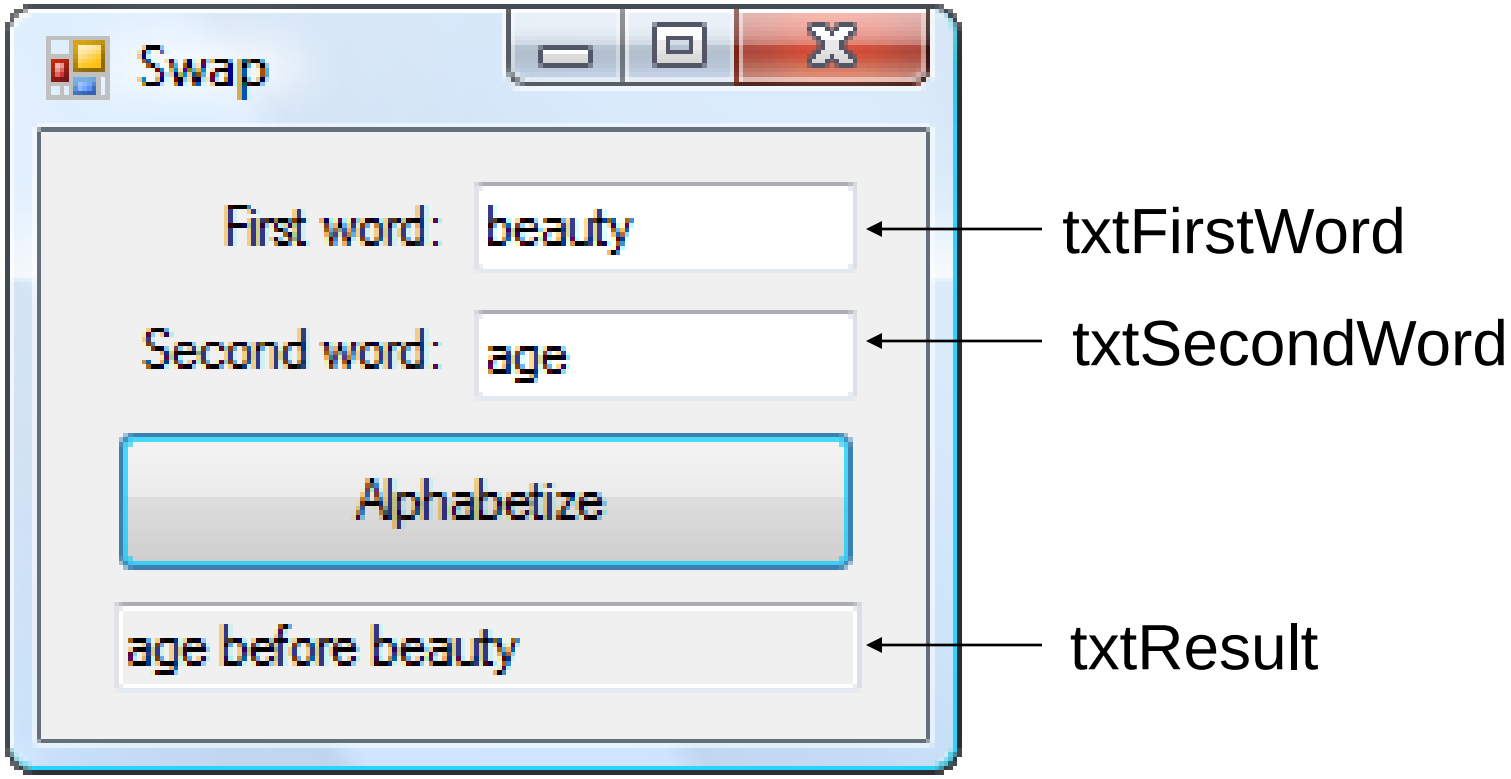
- Sorting is an algorithm for ordering an array

- We discuss two sorting algorithms:
  - **bubble sort**
  - **Shell sort**

- Both use the swap algorithm:

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

Example 1 Output



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
```

Bubble Sort Algorithm: *N* Items

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 items
4. The last item will be at its proper place
5. Do another pass through first $n - 1$ items
6. Repeat this process with one less item for each pass until a pass uses only the first and second items

Bubble Sort

```
Sub BubbleSort(arr As Variant, Optional numEls As Variant, Optional descending As
Boolean)
    ' account for optional arguments
    If IsMissing(numEls) Then numEls = UBound(arr)
    firstItem = LBound(arr)
    lastSwap = numEls
    Do
        indexLimit = lastSwap - 1
        lastSwap = 0
        For index = firstItem To indexLimit
            value = arr(index)
            If (value > arr(index + 1)) Xor descending Then
                ' if the items are not in order, swap them
                ' … swap values at index and index + 1
            End If
        Next
    Loop While lastSwap
End Sub
```

Demo:

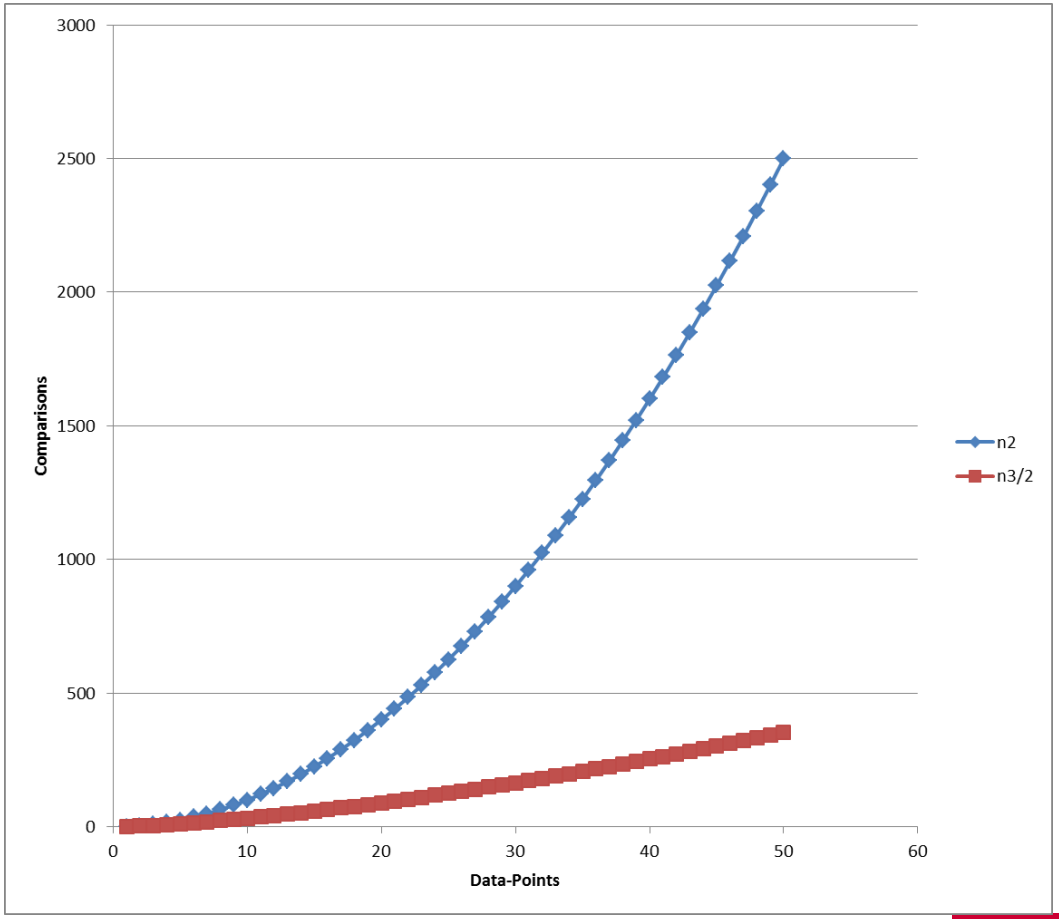
http://www.sorting-algorithms.com/random-initial-order

Shell Sort Algorithm

1. Begin with a gap of $g = Int(n/2)$
2. Compare items 0 and g, 1 and $1 + g$, . . ., $n - g$ and n. 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

Comparison

- Bubble Sort
  - $O(n^2)$ comparisons and swaps
  - O(n) when nearly sorted
- Shell
  - $O(n^{3/2})$ time as shown
  - O(n lg(n)) time when nearly sorted



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

BINARY SEARCH: ALGORITHM

- Given: an array in ascending order and a sought-after value, *query*, that may be in the array

- Repeatedly halve the range of indices where *query* might be found.
- Halving routine looks at the middle value of the current range and compares it to *query* with =, >, and <.
  - If *middle value* = *query*, then search is over.
  - If *middle value* > *query*, then we can limit our search to the half of the range below the middle value.
  - If *middle value* < *query*, then we can limit our search to the half of the range above the middle value.

Binary Search

1	5	7	9	11	15	17	20	24	29

BINARY SEARCH: VARIABLES

first – lower limit of range of values to search

last – upper limit of range of values to search

middle = Int((*first* + *last*) / 2)

a() – ordered array to be searched

foundFlag – True when *query* is found

Note: If *query* is not in the array, eventually *last* will be greater than *first*.

Note: Initially *first* = 0 and *last* = a.GetUpperBound(0)

Binary Search: Code

```
Do While (first <= last) And (Not FoundFlag)
  middle = CInt((first + last) / 2)
  Select Case a(middle)
    Case query
      foundFlag = True
    Case Is > query
      last = middle - 1
    Case Is < query
      first = middle + 1
  End Select
Loop
```

Binary Search: Notes

- If a binary search ends with *foundFlag* = True, the subscript of the found item might be useful
- This would be the case if the array were an array of structures that was ordered with respect to one of its members
- The binary search would serve as an efficient **table lookup** process

7.5 Two Dimensional Arrays

- 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 the statement

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

and the item in the *i*th row, *j*th column is denoted

```
arrayName(i,j)
```

Road-Mileage Table

	Chicago	LA	NY	Philly
Chicago	0	2054	802	738
LA	2054	0	2786	2706
NY	802	2786	0	100
Philly	738	2706	100	0

```
Dim rm(3, 3) As Double
rm(0,0)=0, rm(0,1)=2054, rm(1,2)=2786
```

Populating a Two-Dimensional Array

```
Dim rm(3, 3) As Double
Private Sub frmDistances_Load(...) Handles MyBase.Load
  'Fill two-dimensional array with intercity mileages
  Dim sr As IO.StreamReader = _
                          IO.File.OpenText("DISTANCE.TXT")
  For row As Integer = 0 To 3
    For col As Integer = 0 To 3
      rm(row, col) = CDbl(sr.ReadLine)
    Next
  Next
  sr.Close()
End Sub
```

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 = {{ROW0},
  {ROW1},... {ROWm}}
```

Notes on Two-Dimensional Arrays

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

ReDim and Two-Dimensional Arrays

- An already-created array can be resized with

```
ReDim arrayName(r, s)
```

- which loses the current contents, or with

```
ReDim Preserve arrayName(r, s)
```

- When **Preserve** is used, only the column can be resized

- ReDim cannot change the number of dimensions in an array

Notes on Two-Dimensional Arrays

- The upper bound of the row (the first coordinate) of the array:

  arrayName.GetUpperBound(0)

- The upper bound of the column (the second coordinate) of the array:

  arrayName.GetUpperBound(1)