I101/B100
Problem Solving with Computers

By:

Dr. Hossein Hakimzadeh
Computer Science and Informatics
IU South Bend
What do we need to learn in order to write computer programs?

- Fundamental programming constructs:
  - Variables,
  - Arithmetic operators,
  - Input and output
  - Conditionals,
  - Loops,
  - Procedures and functions,
  - **Arrays**,
  - Structures, classes and objects,
  - Files
What is an Array?

- Array is a homogeneous aggregate of data elements.

- Array is a collection of objects of the same type.

- Array is a collection of similar variables which are identified under the same name.
Why use an Array?

- It is often the case when writing programs that one needs a large number of storage locations. Ordinary variables have to be declared individually, and are not well suited for this purpose.

- Example:
  - If we wish to have 5 variables to hold test scores, one might make the following declaration:

    ```
    DIM Score1, Score2, Score3, Score4, Score5 AS Double
    ```
Why use an Array?

- Arrays allow the programmer to create a series of variables and reference them using a single variable name.
Declaring an Array

- Syntax:

  Dim ArrayName(UpperSubscript) as DataType
Declaring an Array

- Create 10 variables of type “double” under the name TestScore.

- Dim TestScore(10) As Double

- NOTE: When declaring arrays, VB provides us with one more array elements than it is asked for. (The reason for this has to do with being backward compatible.)
Putting Values in the Array Elements:

TestScore (0) = 2
TestScore (1) = 5
TestScore (4) = 9
Retrieving Data from An Array:

To retrieve the element in location 1 in the above TestScore array, we perform the following:

\[
\text{DIM Test1 AS Double}
\]

\[
\text{Test1} = \text{TestScore}(1)
\]
Array Properties:

- The Array’s “Length” property returns the length of the array.

Example:
- Dim Score(10) As Double
- Console.WriteLine("Array length = \{0\}", Score.Length)

Output:
- Array length = 11
Array Methods:

- The Array’s **GetUpperBound()** property returns the highest index of the array.

- Example:
  - Dim Score(10) As Double
  - Console.WriteLine("Array upper bound = {0}", Score.GetUpperBound(0))

- Output:
  - Array upper bound = 10
Operations on an Array:

- Initializing the array
- Inserting data in the array
- Displaying the cell contents of the array
- Searching an Array
- Sorting an Array
Operations on an Array:

- Initializing the array:

  Dim Index As Integer
  Dim Score(10) As Double

  For Index = 0 To 10
    Score(Index) = 0.0
  Next Index
Can you write a procedure to initialize the array?

Class Exercise
Operations on an Array:

- Inserting data into the array:

  For Index = 0 To 10
  Console.Write("Enter array element ")
  Console.Write(Index)
  Console.Write(": ")
  Score(Index) = CDbl(Console.ReadLine())
  Next Index
Can you write a procedure to load the array using user input?

Class Exercise
Operations on an Array:

- Displaying the cell contents of the array:

```
For Index = 0 To 10
    Console.WriteLine(Score(Index))
Next Index
```
Can you write a procedure to print the elements of the array?

Class Exercise
Operations on an Array:

- **Searching an Array:** (Linear Search)

```vbnet
Dim value As Double
Console.WriteLine("Enter the value to search for:")
value = CDbl(Console.ReadLine()) 'value to search for

For Index = 0 To 10
    If Score(Index) = value Then
        Console.WriteLine("Location {0}, Found it!", Index)
    Else
        Console.WriteLine("Location {0}, Not Found..", Index)
    End If
Next Index
```
Can you write a function to find a value in the array and return its location?

Class Exercise
Operations on an Array:

- **Sorting an Array** (Bubble Sort)

```vbnet
Private Sub BubbleSort(ByRef TheArray() As Integer)
    Dim Pass, Index, Hold As Integer
    For Pass = 1 To TheArray.GetUpperBound(0)
        For Index = 0 To TheArray.GetUpperBound(0) - 1
            If TheArray(Index) > TheArray(Index + 1) Then
                Hold = TheArray(Index)
                TheArray(Index) = TheArray(Index + 1)
                TheArray(Index + 1) = Hold
            End If
        Next Index
    Next Pass
End Sub
```
Operations on an Array:

- Binary Search

```vbnet
Private Function BinarySearch(ByVal value As Integer, ByRef TheArray() As Integer) As Integer
    Dim Low, High, Middle As Integer
    Low = 0
    High = TheArray.GetUpperBound(0)

    Do While Low <= High
        Middle = (Low + High) \ 2
        If value = TheArray(Middle) Then 'It’s a match!!
            Return (Middle)
        ElseIf value < TheArray(Middle) Then 'Search the low end of array
            High = Middle - 1
        Else
            Low = Middle + 1
        End If
    Loop
    Return (-1) 'return -1 to indicate the value was not found

End Function
```
More Array Operations...

- Write a function named Sum_Score() which calculates and returns the sum of all elements in the array.

- Write a function named Average_Score() which calculates and returns the average score in the array.

- Write a function named Min_Score() which finds and returns the minimum score in the array.

- Write a function named Max_Score() which finds and returns the maximum score in the array.
Exercise:

☐ Try writing sub-programs that calculates the sum of all the values in an integer array and return the result:

```
Function Sum(ByVal TheArray()) as integer
    Dim the_sum AS integer = 0
    return (the_sum)
End Function
```
Exercise:

- Try writing sub-programs that calculates the average of all the values in an integer array and return the result:

```vbnet
Function Average(ByVal TheArray()) as double
    Dim the_avg AS double = 0.0
    return (the_avg)
End Function
```
Multi-Dimensional Arrays:

- Arrays can have more than one dimension
- Like a table of values
- Or a Cube of values.
Multi-Dimensional Arrays:

- Syntax:

```
Dim ArrayName(HighestRowSubscript, HighestColumnSubscript) as Datatype
```
Example-1

- Two Dimensional Array

Dim ScoreBoard(1,8) As Integer

This declaration creates 18 storage locations (2x9) in which to put **Integer** values.
Manipulating the Array

- In order to access each variable (array element) we must use two array indexes.

\[ \text{Score}(0,0) = 2 \]
\[ \text{Score}(1,0) = 5 \]
\[ \text{Score}(1,4) = 9 \]
Example-2

☐ Two Dimensional Array

Dim strName(2, 3) as String

☐ This declaration creates 12 storage locations (3x4) in which to put string values.
Example-3

- Two Dimensional Array

Dim ScoreBoard(9, 4) As Integer

- This declaration creates 50 storage locations (10x5) in which to put **Integer** values.
Operations on a 2D Array:

- Initializing the array with zeros

- Displaying the cell contents of the array

- Initializing the array with random numbers
Initializing a 2D Array:

'Make a 10x5 storage locations
Dim ScoreBoard(9, 4) As Integer

Dim Row, Col As Integer

For Row = 0 To 9
    For Col = 0 To 4
        ScoreBoard(Row, Col) = 0
    Next Col
Next Row
Let’s write a Procedure to do the same Initialization

Private Sub Initialize(ByRef TheArray(,) As Integer)

    Dim Row, Col As Integer

    For Row = 0 To TheArray.GetUpperBound(0)
        For Col = 0 To TheArray.GetUpperBound(1)
            TheArray(Row, Col) = 0
        Next Col
    Next Row

End Sub

Returns “9”

Returns “4”
Print the Content of the Array

Private Sub Print(ByVal TheArray(,) As Integer)
    Dim Row, Col As Integer
    For Row = 0 To TheArray.GetUpperBound(0)
        For Col = 0 To TheArray.GetUpperBound(1)
            Console.Write("{0,4}", TheArray(Row, Col))
        Next Col
        Console.WriteLine()
    Next Row
End Sub
Output after calling Initialize()
Put Random Numbers in the Array

Private Sub InitializeRandom(ByRef TheArray(,) As Integer)

    Dim Row, Col As Integer

    Randomize() ' Seed the random number generator

    For Row = 0 To TheArray.GetUpperBound(0)
        For Col = 0 To TheArray.GetUpperBound(1)

            TheArray(Row, Col) = CInt(Rnd() * 10) ' return a number in the range 0 to 10

        Next Col
    Next Row

End Sub
Output after calling InitializeRandom()

4 6 8 4 5
6 1 6 10 0
1 7 10 5 5
2 5 4 4 8
8 3 2 3 2
9 8 5 6 1
4 6 4 6 4
5 4 2 3 6
7 9 8 0 5
2 10 8 7 0

Note:

If we call the InitializeRandom() procedure again, the result will be different!