CSE 123
Introduction to Computing

Lecture 11
Programming with Arrays

SPRING 2012

Assist. Prof. A. Evren Tugtas

Marmara Üniversitesi

Array Variables Review

- For detailed information on array variables look at the notes of Lecture 7.
- Array Variables hold one bit of data under a name
- Array variable holds more bits of information under a name

```
Dim Vegetables (1) as String
Vegetables(0)="Carrot"
Vegetables(1)="Cauliflower"
```

Array Variables in VBA

- Array is a variable that can contain number of values that have the same data type.
- Array is treated as a single value in VBA
- You can refer to array itself to work with all the values it contains.
- You can also refer to individual numbers stored within the array by using their index numbers

MsgBox Vegetables(2) ---- Celery

Array Variables in VBA

- An array is bounded by a lower and upper bound
- By default the lower bound is ZERO, therefore, the first item in an array is indexed as ZERO

```
Dim Vegetables (1) as String
Vegetables(0)="Carrot"
Vegetables(1)="Cauliflower"
```

- This could be confusing because the index number is always one lower than the items position in an array.

Array Variables in VBA

- VBA lets you change the default lower bound.
- Using **Option Base 1** statement at the begining of your code makes the default index number of the first item in an array 1.
- **Option Base 1** statement makes the index number for each item in an array the same as the item's position in that array.
- Other programming languages do not have this option, by default their arrays are zero based.

Declaring an Array

- Alternatively lower bound of an array can be specified as;
- Dim Sample (1 to 10)

Declaring an Array

- Number of items in an array are declared by an array subscript.
- Following statement declares that the array named A assigns the Single data type and contains **6** items. A is a one dimensional array.

Dim A(5) as Single

Be carefull, it is a ZERO BASED array

Multidimensional Arrays

- In multidimensional arrays the information in any series does not have to be related to eachother.
  - You can assign 10 folder names to first dimension as string
  - 10 filenames to second dimension as string
  - Names of 10 cities to the third dimension
- Later on you can access the any information you like by specifying the location.
- Excel workbook of worksheets, rows and columns is a three-dimensional array.

Multidimensional Arrays

- Array with dimensions 500 rows and 2 columns
- Dim myarray(500, 2)

Returning Dimensions of an Array

- Dim Sample (4, 500)
- Ubound(Sample, 1) returns 4
- Uboung(Sample, 2) returns 500

Declaring a Dynamic Array

- Arrays can be declared as ***fixed-size arrays*** or ***dynamic arrays***.
- Dim A(6) → ***Fixed-size*** array
- ***Dynamic arrays*** are used when you are storing changing number of arrays
- If you do not know what array size you will need to handle a particular problem, you can create a dynamic array.
-

Declaring a Dynamic Array

- You should not specify the item number when you are declaring a dynamic array.
- **Dim Power( )** → dynamic array, size is not declared

Redimensioning an Array
ReDim Statement

- You can change the size of an Array by using ***ReDim*** Statement

ReDim Countries(5)

- When you redimension the arrays using ReDim statement, you lose the values currently in the array.

Redimensioning an Array
ReDim Statement

```
Dim MeanX(), MeanY()
.
.
'Get the number cells to use in calculation
n=Inputbox(Number of cells?)
ReDim MeanX(n), MeanY(n)
```

- If you use ReDim command to change the size of an array, all the stored data will be erased

Preserving data in dynamic arrays

```
Dim MeanX(), MeanY()
.
.
'Get the number cells to use in calculation
n=Inputbox(Number of cells?)
ReDim Preserve MeanX(n), MeanY(n)
```

- Limitation: Only upper bound of the last dimension will be preserved.
- If you use preserve, you cannot use redim command to change the number of dimensions

Array Formulas in Excel

- If you pull down the cursor all the cells will include the formula for the multiplication of the calues in A and B columns

STDEV =A3*B3

	A	B	C	D	E
1					
2	Concentration (mg/L)	Liquid Volume (L)	Mass (mg)		
3	150	1	=A3*B3		
4	100	0.9			
5	200	0.75			
6	120	0.5			
7					

Array Formulas in Excel

- An alternative way, treat them as matrices
- CTRL+SHIFT+ENTER

STDEV | =A3:A6*B3:B6

	A	B	C	D	E
1					
2	Concentration (mg/L)	Liquid Volume (L)	Mass (mg)		
3	150	1	=A3:A6*B3:B6		
4	100	0.9			
5	200	0.75			
6	120	0.5			
7					
8					

Working with Arrays in Sub Procedures
Passing Values from Worksheet to VBA

There are two ways to get data to VBA array.

1) Setup a loop and read the value of each cell and store the value in appropriate array element (easy)
2) You can assign the VBA array to a worksheet range

- If you need to access array elements a number of times, it will be more time efficient to store the values in an internal array

Working with Arrays in Sub Procedures

- If a variable in a VBA Sub is set equal to a range of cells in a worksheet, that variable can be used as an array;
- **Dim** statement is **not necessary**

Myarray=Range(“A2:A19")

Working with Arrays in Sub Procedures
Passing Values from VBA to Worksheet

- One Dimensional Array - Problems:
- Arrays can cause confusions when you try to write it back to the worksheet
- VBA considers a one-dimensional array to have the elements of the array in a row
- For Example

  Range("A1:A12").**Value**=MyArray
- MISTAKE: The first element of the array will be entered to all cells in the column

Working with Arrays in Sub Procedures
Passing Values from VBA to Worksheet

- Correct way:

  Range("A1:L1").**Value**=MyArray

- Each cell of the range will receive the correct value
- There are three solutions to this problem
  - Write a loop
  - Specify both row and column dimensions
  - Use TRANSPOSE worksheet function

Working with Arrays in Sub Procedures
Passing Values from VBA to Worksheet

```
Sub Example2()
'Second way to solve row-column problem
'by specifying the row and column dimensions
Dim MyArray(12,1)
  statements to populate the array
Range("A1:A12").Value=MyArray
End Sub
```

Marmara Üniversitesi

Working with Arrays in Sub Procedures
Passing Values from VBA to Worksheet

```
Sub Example3()
'Third way to solve row-column problem
'is the use of TRANSPOSE worksheet function
'Transpose creates 1-base array
Dim MyArray(12)
  statements to populate the array
Range("A1:A12").Value=Application.Transpose (MyArray)
End Sub
```

Marmara Üniversitesi

Arrays in Function Procedures

- A sub procedure is a program that you can run
- A Function procedure is a program that calculates a value and returns it
- A Function procedure cannot change the worksheet environment
- A range passed to a Function procedure can be used as an array

Arrays in Function Procedures

- Dim statement is not necessary

```
Function calc(y_values, x_values)
```

- Passes the worksheet ranges y_values and x_values to the VBA procedure

Passing indefinite number of Arguments to a Function

- E.g. Sum function requires indefinite number of arguments
- Sum (number1, number 2,.....)
- **ParamArray** keyword is used

Function Example4(**ParamArray** rng())

Passing indefinite number of Arguments to a Function

```
Function Example3(ParamArray rng())

For i=0 to Ubound(rng)
   n=rng(i)columns.count
   For K=1 to n
         statements
   Next K
Next i
```

Example

- Number of students
- Number of classes each student is taking
- Final grade of each class

```
Sub array1()
Dim namest(100) As String
Dim classnum(100) As Integer
Dim class(100, 100), grade(100, 100) As Double
1
n = InputBox("enter the total number of students")
If n > 100 Then
  MsgBox ("number of students should be less than 100, please enter again")
  GoTo 1
End If
For i = 1 To n
namest(i) = InputBox("Enter the name of " & i & ". student")
classnum(i) = InputBox("Enter the number of classes " & namest(i) & " is taking")
   For j = 1 To classnum(i)
   class(i, j) = InputBox("Enter the name of the " & j & ". class " & namest(i) & "
     is taking")
   grade(i, j) = InputBox("Enter " & namest(i) & "s grade for " & class(i, j) & ".")
   Next j
Next i
End Sub
```