Learning Objectives

- Arrays as subscripted variables
- **Dim** statements
- Dynamic dimensioning

Arrays

- Arrays are a group of numbers whose elements are arranged in a single or multi-dimensional format
- The elements are accessed using index notation
- The elements are of the same data type (e.g., matrix with real numbers or integer numbers)
- Used to store large sets of numbers

Array Declaration

- Arrays are indicated by subscripted variables in VBA programs
- Must tell computer that your variable is an array and tell the maximum number of members in the array
- If `x` is an array of integers with 30 members,
- `x[1] ... x[30]`
  - Use **Dim** `x(30)` as integer
- **Dim** statement must appear before any `x` is used

Array Declaration

- Array member type must be declared
- Can define more than one array in a line:
  ```vba
  Dim A(10) as double, B(20) as single, C(40) as integer
  ```

Array Declaration

- Array indexes go from 0 to the number specified
- For `A(30)`, you can store values at `A[0]`, `A[1]`, `A[30]`, and this (starting the index at 0) can be confusing
- Another approach to size the array is to use `A[1] to 30`, which will start from `A[1]`
- One way to default the index of the first array member to 1 is to use the statement "**Option Base 1**" at the top of the program. This will force the array to start at 1. For example:
  ```vba
  Option Base 1
  Dim AA(30) as integer
  In AA(30), you can store values at AA[1], AA[2]... To A[30]
  ```

Array Declaration

- Without **Dim**, VBA would think the array variable a function – you will see an error message
- It important that array members not exceed the maximum number specified in the **Dim** statement
- For example:
  ```vba
  Option Base 1
  Dim grades(15) as single
  Sum = 0
  For i = 1 to 20
    sum = sum + grades(i)
  Next i
  When i becomes 16, you will try to access grades(16) and this will overflow the array and generate an error message.
  ```
Arrays (Example)

Private Sub Mycode_Click()
Dim grade(10) As String
Dim Points(10) As Single
Dim i As Integer, n As Integer
n = 10
For i = 1 To n Step 1
Points(i) = Cells(1 + i, 1)
Next i
(continued on next slide)

<table>
<thead>
<tr>
<th>Points</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>A</td>
</tr>
<tr>
<td>84</td>
<td>B</td>
</tr>
<tr>
<td>77</td>
<td>C</td>
</tr>
<tr>
<td>91</td>
<td>A</td>
</tr>
<tr>
<td>43</td>
<td>F</td>
</tr>
<tr>
<td>86</td>
<td>D</td>
</tr>
<tr>
<td>71</td>
<td>C</td>
</tr>
<tr>
<td>56</td>
<td>F</td>
</tr>
<tr>
<td>69</td>
<td>D</td>
</tr>
<tr>
<td>80</td>
<td>B</td>
</tr>
</tbody>
</table>

Output:

For i = 1 To n
If Points(i) >= 90 Then
grade(i) = "A"
ElseIf Points(i) >= 80 Then
grade(i) = "B"
ElseIf Points(i) >= 70 Then
grade(i) = "C"
ElseIf Points(i) >= 60 Then
grade(i) = "D"
Else
grade(i) = "F"
End If
Next i
For i = 1 To n
Cells(1 + i, 2) = grade(i)
Next i
End Sub

Find average, high and low grades for a class

REM Code to arrange grades and find high, low, and average grades
Dim grade(50) as integer, n as integer
Dim sum as single, high as single
Dim low as single, average as single
' number of grades, grade values in spreadsheet
n = cells(1,1)
sum = 0
high = 0
low = 100
For i = 1 to n 'Transfer grades into an array
grade(i) = cells(i+1,1)
Next i
For i = 1 to n
sum = sum + grade(i)
If grade(i) > high then
high = grade(i)
End If
If grade(i) < low then
low = grade(i)
End If
Next i
average = sum/n
Msgbox "average" & average & " high " & high & " low " & low
End Sub

Example:
Average Grades and Find High and Low

For i = 1 to n
sum = sum + grade(i)
if grade(i) > high then
high = grade(i)
endif
if grade(i) < low then
low = grade(i)
endif
next i
average = sum/n
Msgbox "average" & average & " high " & high & " low " & low
End Sub

Example:
Average Grades and Find High and Low

• Note we use loop index "i" to specify the grade of a student being considered
• Combination of arrays and For indices is a very powerful way to manipulate large matrices
• Note, this program limits solution to 50 grades, but actual number of students can vary. If there are more than 50 students this code will not work…how can we handle this?

Dynamic Dimensioning

• Could use dynamic dimensioning to re-specify size of grades after n read in:

Dim grade() as integer:
' grade specified array but without size
n = cells(1,1)
Redim grades(n) 'size of grade set to n
Indefinite Input from Spreadsheet

Let us assume n entries from cells (1,1) to cells(n,1), then we can use:

\[
\begin{align*}
    n &= 0 \\
    \text{Do While Cells}(n+1, 1) &\neq \text{Empty} \\
    n &= n + 1 \\
    \text{Loop}
\end{align*}
\]

Redim grades(n)

- Program stays in loop as long as data encountered, i.e. Empty is false
- Note n is initialized and counted

Multi-dimensional Arrays

- Variables with 2 or more subscripts may be defined and used (VBA allows up to 60 dimensions!)
- Defined with dimension values separated by commas
- Example for set of 25 equations with augmented array

\[
\text{Dim a}(25, 26) \text{ as double, x}(25) \text{ as double}
\]

Multi-dimensional arrays (example)

- Multi-dimensional arrays most useful in matrix operations.
- One simple example is to obtain a transpose of a given matrix.
- The transpose of a matrix (A) is obtained by switching the rows of a given matrix as the columns of the transposed matrix (A').

Transpose of a matrix

```
Private Sub Transpose_Click()
    Dim A() as double, B() As Double
    Dim n as integer, m as integer
    Dim i as integer, j as integer
    n = Cells(1, 2)
    m = Cells(2, 2)
    ReDim A(n, m), B(m, n)
    'read the numbers of A
    For i = 1 To n
        For j = 1 To m
            A(i, j) = Cells(3 + i, j)
        Next j
    Next i

    'switch the rows and create matrix B
    For i = 1 To m
        For j = 1 To n
            B(i, j) = A(j, i)
        Next j
    Next i

    'print the elements of matrix B
    For i = 1 To m
        For j = 1 To n
            Cells(9 + i, j) = B(i, j)
        Next j
    Next i
End Sub
```

RExample- 2

REM code to multiply square matrix with a vector

```
Dim i As Integer, j As Integer, n As Integer
Dim A() As Single, B() As Single, C() As Single
n = Cells(1, 2)
ReDim A(n, n) As Single, B(n) As Single, C(n) As Single

'Read Matrix A from spreadsheet
For i = 1 To n
    For j = 1 To n
        A(i, j) = Cells(2 + i, j)
    Next j
Next i

'Read Vector B from spreadsheet
For i = 1 To n
    B(i) = Cells(2 + i, n + 2)
Next i
```

THIS IS EXPLAINED IN VB TUTORIAL EXAMPLE- 2
Perform matrix multiplication to compute C
For i = 1 To n
C(i) = 0#
For j = 1 To n
C(i) = C(i) + A(i, j) * B(j)
Next j
Next i

Output the results to spreadsheet
For i = 1 To n
Cells(2 + i, n + 4) = C(i)
Next i

REM General matrix multiplier (main code)
Dim a() As Single, b() As Single, c() As Single, n1 As Integer,
n2 As Integer, n3 As Integer
Dim i As Integer, j As Integer
n1 = 2
n2 = 2
n3 = 2
ReDim a(n1, n2) As Single, b(n2, n3) As Single, c(n1, n3) As Single
a(1, 1) = 1: a(1, 2) = 2
a(2, 1) = 4: a(2, 2) = 5
b(1, 1) = 10: b(1, 2) = 20
b(2, 1) = 30: b(2, 2) = 40
Call matmultiply(a, b, n1, n2, n3, c)
For i = 1 To n1
For j = 1 To n3
Cells(i, j) = c(i, j)
Next j
Next i

General matrix multiplier (using subroutine)
Sub matmultiply (aa() As Single, bb() As Single, cc() As Single, nn1
As Integer, nn2 As Integer, nn3 As Integer)
' This routine multiplies two matrices a and b and computes c
' aa (nn1 x nn2) and bb (nn2 x nn3) are the two input matrices and
c (nn1 x nn3) is the result
Dim i As Integer, j As Integer, k As Integer, sum As Single
For i = 1 To nn1
For j = 1 To nn3
sum = 0#
For k = 1 To nn2
sum = sum + a(i, k) * b(k, j)
Next k
cc(i, j) = sum
Next j
Next i
End Sub

Note: The arrays aa(), bb(), and cc() are not re-dimensioned in the
subroutine. The shape of these arrays matches the shape of the
arrays in the main program a, b, c.