Review for Programming Exam and Final

Larry Caretto
Mechanical Engineering 209

Computer Programming for Mechanical Engineers

May 4-9, 2017

California State University Northridge

Outline

- Schedule
- Excel Basics
- VBA Editor and programming variables
- Arithmetic statements
  - Order of precedence
- If Statements
- Looping
- Arrays

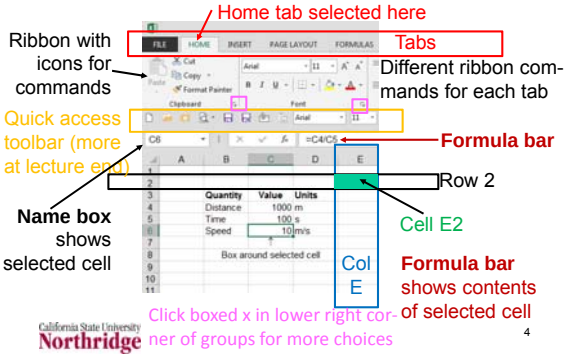
California State University Northridge 2

Semester Calendar

Date	Assignment	Date	Assignment
May 2		May 4	Quiz 5 on Arrays
May 9	Assignment 6 Due	May 11	Programming Exam
		May 18	Final Exam 12:45 to 2:45 pm

California State University Northridge 3

Final Review: Excel Basics



California State University Northridge 4

Using Excel

- Entering and copying formulas
  - Understanding absolute/relative references
- Excel functions like sin(), exp(), etc.
- Formatting cells
- Creating/modifying plots not on final
- Creating macros (subs without arguments) can be called from worksheet
- Advanced topics like data validation and range names not on final

California State University Northridge 5

Using VBA

- Transfer to and use of editor
- Writing functions and subs
  - Functions transfer information through the argument list, **based on position in list**
  - A macro is a sub with no arguments
  - Information transfer to/from subs
    - Range("M12").Value = x * y — Write property
    - Cells(12,13).Value = x * y — Write property
    - Z = Range("C5").Value — Read property
    - Z = Cells(5,3).Value — Read property

California State University Northridge 6

Example Function

```
Const getMolarMassError as Double = -9999
Const universalGasConstant as Double = 8.3144621
Function idealGasDensity(substanceName As _
   String, pressure As Double, temperature As _
                              Double) As Double
    'Computes ideal gas density

    Dim molarMass As Variant
    molarMass = getMolarMass(substanceName)
    If molarMass = getMolarMassError Then _
        idealGasDensity = "Masslookup error"
    Else
        idealGasDensity = molarMass * pressure _
           / (universalGasConstant * temperature)
    End If
End Function
```

=idealGasDensity(B2, C3, D4)

Function reference passes three inputs: name, pressure, and temperature in that order.

Use of Example Function

- Call from worksheet

	A	B
1		
2	Substance	Nitrogen
3	Pressure	101.325
4	Temperature	400
5	Density	=idealGasDensity(B2,B3,B4)

Order of arguments must match function header

- May also use constant or range name
- Call from other VBA procedures
  - May use variables or constants

```
density = idealGasDensity("Nitrogen", _
                                  400, 350)
pressure = 400 : temperature = 350
substance = "Nitrogen"
density = idealGasDensity(substance, _
                pressure, temperature)
```


Same Example as Sub

```
Const getMolarMassError as Double = -9999
Const universalGasConstant as Double = 8.3144621
Sub idealGasDensity()
    Dim substanceName As String
    Dim pressure As Double
    Dim temperature As Double
    Dim molarMass As Variant
    substanceName = Range("B2").Value
    pressure = Range("B3").Value
    temperature = Range("B4").Value
    molarMass = getMolarMass(substanceName)
    If molarMass = getMolarMassError Then _
        Range("B5").Value = "Masslookup error"
    Else
        Range("B5").Value = molarMass * pressure _
           / (universalGasConstant * temperature)
    End If
End Sub
```

Will need to use Range().Value on final

Generic Function

Function <name> (<arguments>) As <type>

    <statements>

End Function

<statements> must include at least one statement that sets <name> = <value>

Inputs to function come from <arguments> not statements like Range.Value

Generic Sub

Sub <name> (<arguments>)

    <statements>

End Sub

<arguments> in subs and functions is the argument list which has zero or more items like the following: <variable> As <Type>

Multiple items are separated by commas

Declaring Variables/Consts

- Use the following syntax

```
Dim x As Double  'Distance
Dim k As Long
Dim s As String, d as Date
Const g As Double = 9.80665
```

Note common data types

- Do ***not*** use the following syntax

```
Dim x, y As Double
```

- Use `Option Explicit` to find errors in undeclared variables

Assignment Statements

- Assigns a value to a variable
  - Examples shown below

```
degreesToRadians = PI / 180
force = mass * accel
```

- What is really being done here?
  - Names refer to computer memory locations
  - The value of the expression on the right of the = sign is assigned to the memory location of the variable on the left
    - Can have statements like x = x + 1

California State University Northridge 13

String Variables

- String variables hold text information
- Declare string variables using Dim <variable> As String
  - E. g.: `Dim s As String, msg as String`
- String constants defined with quotation marks: "ME 209" is a string constant
  - String assignment: `s = "ME 209"`
- Concatenate (join) two strings with the concatenation (&) operator

```
msg = "This course is " & s
```

California State University Northridge 14

Operators in Precedence Order

Parentheses (Evaluate me first)
Exponentiation ^
Unary minus – (*E.g.* –x)
Multiply/Divide * /
Integer Division \
Mod (remainder, *e.g.* 7 Mod 4 = 3)
Addition/Subtraction + –
String concatenation &
Relational < <= = >= > <>
Logical in following order: Not And Or

California State University Northridge 15

Writing Equations

- Important to remember operator precedence rules
  - Use parentheses to get
  - Example: $z = \frac{p+qz}{s^2 r + \cos y}$
  - Write a statement to compute z

```
z = (p + q * z) / (s^2 * r + cos(y))
```

  - Write statement on one line (or continuation)
  - Use x*y to multiply x times y
  - Parentheses for multiplication/division

On final do not use [] in place of () and write formulas as z = (a+b)/(c+d) instead of $z = \frac{(a+b)}{c+d}$

California State University Northridge 16

Relational/Logical Operators

- Relational operators compare other variables and have true or false results
  - VBA operators are <, <=, =, >, >=, <>
  - Not equal <> is less than or greater than
- Logical operators (Not, And, Or) combine Boolean (true/false) results
  - **x And y** is true if both x and y are true
  - **x Or y** is true if either x or y is true
  - **Not x** is true if x is false; is false if x is true

California State University Northridge 17

If – Else If

```
If <condition1> Then
   <Statements done if condition1 is true>
ElseIf <condition2> Then
   <Statements done if condition2 is true>
ElseIf <condition3> Then
   <Statements done if condition3 is true>
<May be other conditions>
Else
   <Statements done if all conditions false>
End If
<Execute here after any statements done>
```

A <condition> is TRUE or FALSE

May have zero or many ElseIf's

Else statement is optional

California State University Northridge 18

Boolean Variables

- Boolean variables have two possible values: TRUE or FALSE
- Can set Boolean variables equal to a relational or logical expression
- Declare variables as Boolean

```
Dim converged As Boolean
```

- Example of use

```
converged = diff < absErr Or _
    diff < relErr * Abs(reference)
```

California State University Northridge 19

The Do While Loop

- The Do While <condition> loop executes as long as the <condition> is true
- The Do While x < 10 ends only when x ≥ 10 at the start of the loop
- In the operation of the loop, a new value of x is tested before the loop starts
- Once the loop starts all statements before the "Loop" statement are completed until a new test is made unless there is a conditional exit

California State University Northridge 20

Loop Until Command

- An alternative to the Do While … Loop
  - Starts with the command Do
  - Followed by loop body with all code executed in loop
  - Concludes with Loop Until <condition>
- Loop continues as long as <condition> is false
- Loop is exited when <condition> is true
- Loop Until condition opposite of Do While

California State University Northridge 21

Count-controlled Loops

- Modified structure of VBA For loop

```
For <counter> = <start> To _
        <finish> Step <increment>
   <statements>
Next <counter>
```

Step usually omitted if <increment> = 1

- Here <counter> is changed by <increment> each time through loop
- Loop is executed nTimes times where nTimes = Int((<finish> – <start>) / <increment>) + 1
  - Not executed if nTimes ≤ 0
  - Executed once if <start> = <finish>

California State University Northridge 22

Do While Example

```
Function mySqrt(A As Double) As Variant
  Const and Dim statements omitted

  mySqrt = 1: iterations = 0: converged = False
    Do While Not converged And _
                    iterations < maxIterations
        mySqrt = mySqrt / 2 + A / (2 * mySqrt)
        iterations = iterations + 1
        converged = Abs(mySqrt ^ 2 - A) < _
                              maxRelErr * Abs(A)
    Loop
    If Not converged Then mySqrt = _
                                "No Convergence"
End Function
```

California State University Northridge 23

Same Example With Loop Until

```
Function mySqrt2(A As Double) As Variant
  Const and Dim statement omitted


  iterations = 0 : mySqrt = 1
  Do
    mySqrt2 = mySqrt2 / 2 + A / (2 * mySqrt2)
    iterations = iterations + 1
    converged = Abs(mySqrt2 ^ 2 - A) < _
                              maxRelErr * Abs(A)
  Loop Until converged Or iterations >= _
                                  maxIterations
  If Not converged Then mySqrt2 ="No Convergence"
End Function
```

Opposite of Do While condition

California State University Northridge 24

Same Example With For Loop

```
Function mySqrt3(A As Double) As Variant
  Const and Dim statements omitted

  mySqrt = 1  'No other initializations
  For iteration = 1 To maxIterations
    mySqrt3 = mySqrt3 / 2 + A / (2 * mySqrt3)
    If Abs(mySqrt3 ^ 2 - A) < maxRelErr * _
                        Abs(A) Then Exit Function
  Next iteration
     'maxIterations exceeded if we get here
  mySqrt3 ="No Convergence"
End Function
```

California State University Northridge 25

Tracing Loops

x = 1 : term = x : sum = term
For n = 1 to 10
    term = term * x / n
    sum = sum + term
Next k

Exercise: Show that sum for n = 6 is 1957/720

Initialization: x = 1 term = x = 1 sum = term = 1
n=1: term = 1 * 1 / 1 = 1; sum = 1 + 1 = 2
n=2: term = 1 * 1 / 2 = 1/2; sum = 2 + 1/2 = 5/2
n=3: term = 1/2 * 1 / 3 = 1/6; sum = 5/2 + 1/6 = 16/6
n=4: term: 1/6 * 1 / 4 = 1/24; sum: 16/6 +1/24 = 65/24
n=5: term :1/24*1/5=1/120;sum: 65/24+1/120=163/60

California State University Northridge 26

VBA Arrays

Math	VBA
x_1	x(1)
x_2	x(2)
x_3	x(3)
x_4	x(4)
x_5	x(5)
x_6	x(6)

- Math notation, x_i, is replaced by VBA notation x(i)
  - Name for (i) is index or subscript
  - View this one-dimen-sional array as one row or one column of data
  - Must declare maximum array size

California State University Northridge 27

Declaring Arrays

- Use Dim with array size
  - Usual statement: Dim x As …
  - Array statement: Dim x(<size>) As …
  - <size> specifies maximum subscript
  - May be done for individual arrays where <size> is <minimum> To <maximum>
  - May use <size> as <maximum>
    - In this case default <minimum> is normally zero
    - Can change default <minimum> to one with statement: `Option Base 1`

California State University Northridge 28

Using Array Variables

- We can use individual array elements like any other programming variable
  - E = stress(3)/strain(3)
  - k = 3 : E = stress(k)/strain(k)
  - For k = 1 To N : sum = sum + x(k) : Next k
  - j = 2 : power(j) = current(j) * voltage(j)
  - Dim pressure(1 to 10) as Double
    meanPressure = Application. _
    Worksheetfunction.Average(pressure)

California State University Northridge 29

For Loops and Arrays

- A common application of arrays uses for loops to set elements in an array

```
For k = 1 To 10
    x(k) = cos(k*PI/10)
Next k
```

- Can sum all elements in an array

```
sum = x(1)
For k = 2 To NX
    sum = sum + x(k)
Next k
```

California State University Northridge 30

Two-dimensional Arrays

- View as two-dimensional table of data
- Row and column subscripts
- Must set both subscripts in Dim statement
  - Dim e(1 to 4,1 To 6) As Double
  - Same default minimum behavior as 1D

	I(1)	I(2)	I(3)	I(4)	I(5)	I(6)
V(1)	e(1,1)	e(1,2)	e(1,3)	e(1,4)	e(1,5)	e(1,6)
V(2)	e(2,1)	e(2,2)	e(2,3)	e(2,4)	e(2,5)	e(2,6)
V(3)	e(3,1)	e(3,2)	e(3,3)	e(3,4)	e(3,5)	e(3,6)
V(4)	e(4,1)	e(4,2)	e(4,3)	e(4,4)	e(4,5)	e(4,6)

California State University Northridge 31

Two For Loops for 2D Arrays

- Two-way table from two-dimensional arrays and nested for loops
  - Need to define P(k) and T(j) arrays prior to executing this code

```
For k = 1 to 10
  For j = 1 to 20
    rho(i,j) = M*P(k) / (R*T(j))
  Next j
Next k
```

California State University Northridge 32

Worksheets and Arrays

- Can replace x = Range("A1:T10").Value by code below (using numbers only)

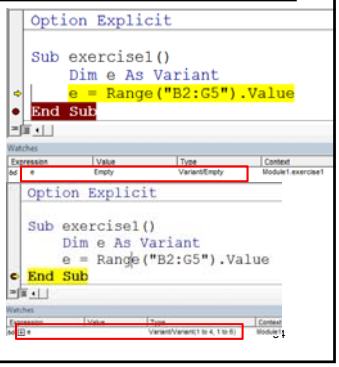
```
Const NX As Long = 10, NY As Long = 20
Dim row As Long, col as Long
Dim x(1 to NX, 1 To NY)
For row = 1 To NX
   For col = 1 To NY
     Cells(row,col).Value= x(row,col)
   Next col
Next row
```

Note use of Const for array bounds and for loop limits. Changes to NX or NY can be done by changing Const values

California State University Northridge 33

The Mysterious Variant

- Type Variant Variables, declared as scalars, can be equated to an array
- They then become arrays
- Used for getting cell data to VBA

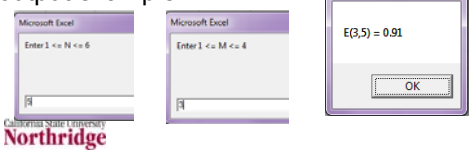


California State University Northridge 34

The Mysterious Variant II

- Can use array created from variant like any other array
- Simple input/ output example

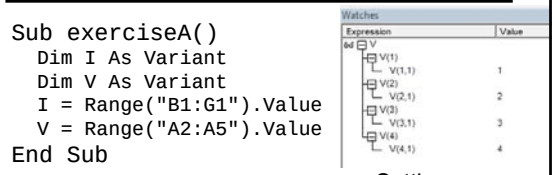
```
Sub exercise1()
    Dim e As Variant, N As Integer, _
                     M As Integer
    e = Range("B2:G5").Value
    N = InputBox("Enter 1 <= N <= 6")
    M = InputBox("Enter 1 <= M <= 4")
    MsgBox "E(" & M & "," & N & _
                    ") = " & e(M, N)
End Sub
```



California State University Northridge 35

Worksheet Range to 1D Array

```
Sub exerciseA()
  Dim I As Variant
  Dim V As Variant
  I = Range("B1:G1").Value
  V = Range("A2:A5").Value
End Sub
```

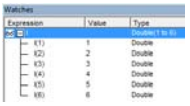


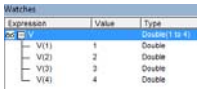
Setting a Variant to a single row or column gives a two-dimensional array

California State University Northridge 36

Cells Method for 1D Array

```
Sub exerciseB()
  Dim I(1 to 6) As Double
  Dim V(1 to 4) As Double
  Dim k As Integer
  For k = 1 To 6
    I(k) = Cells(1,k+1).Value
  Next k
  For k = 1 To 4
    V(k) = Cells(k+1,1).Value
  Next k
End Sub
```



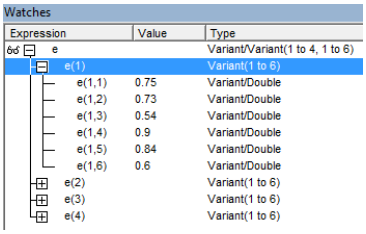


California State University Northridge

Worksheet Range to 2D Array

```
Sub exerciseC()
  Dim e As Variant
  e = Range("B2:G5").Value
End Sub
```

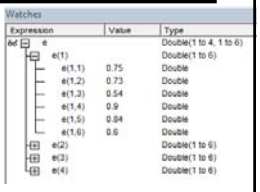
Here e is declared as a scalar variant, but becomes an array when set to a worksheet range



California State University Northridge

Cells Method for 2D Array

```
Sub exerciseD()
   Dim e(1 To 4, _
   1 To 6) As Double
   Dim k As Long
   Dim m As Long
   For k = 1 To 4
     For m = 1 To 6
       e(k, m) = Cells(k + 1, _
                       m + 1).Value
     Next m
   Next k
End Sub
```



California State University Northridge

LBound and UBound

- These VBA functions get lower and upper bounds for array dimensions
  - Recall arrays can have more than one dimension, e.g. p(k,m)
- The arguments are (<array Name>, <dimension>) where dimension = 1, 2, etc. for the first, second, etc. dimension
  - Only look at one- and two-dimensional arrays in this course
  - If <dimension> is omitted it has a value of 1

California State University Northridge

Review Arrays to Procedures

- When passing arrays to procedures it is not necessary to Dim the array
- Instead, the argument to handle an array is written with empty parentheses

```
Function mean(x() as Double) As Double
  Dim k as Long
  mean = 0
  For k = LBound(x) To UBound(x)
     mean = mean + x(k)
  Next k
  mean = mean /(UBound(x) -LBound(x) + 1)
End Function
```

California State University Northridge

Adjustable Size Arrays

- Two steps needed if we do not know the maximum size of the array
- First declare array without size:
  - Dim() <ArrayName> As <Type>
- When size is known use ReDim
  - ReDim(<size>) <ArrayName>

```
Dim x() As Double, N As Long
N = InputBox("Enter Size: ")
ReDim x(1 To N)
```

California State University Northridge

Programming Exam

- Thursday, May 11, 12 – 1:25 pm
- Write/debug a simple program using all items except arrays covered during the semester
  - Some students will complete work, many will not
    - Objective is to see how well you can take an assignment, turn it into code, debug the code, and get an operating program
    - Can use text and online help, but cannot use internet to search for code

California State University Northridge 43

Final Exam

- Friday, May 18, 12:45 to 2:45 pm
- Will have questions similar to midterm and quizzes
- Closed book with same information sheet on commands provided for midterm and quizzes

California State University Northridge 44