Engineering Fundamentals
ENG1100 - Session 14B

Final Exam Review:
VBA
Final Programming Exam
Topics

• Functions
  – Definition Line
  – How to call
  – What is returned to the spreadsheet

• Macros (sub)
  – Definition Line
  – How do you get variables into/out of sub

• Flowcharts

• Assigning Variables

• Flow Control
  – Conditional Statements
  – For Loops
What is going on??

Function sind(angle)
'Function to compute the sine of an angle in degrees
'angle=angle, degrees

'calculate the value for pi
Pi = 4 * Atn(1)

'calculate sine
sind = Sin(angle * Pi / 180)

End Function
The Concept of Assignment

Information within program is stored:
- **Directly** as *constant* (example, 9.81), whose value does not change
- **Symbolically** as a *variable* (*a*=9.81), whose value can be changed

Assignments must have:
- single variable on left
- valid value on right

\[ a = 10 \]

Assignment

Variable (memory location)

Constant (Value)
Equal Sign

• In programming the *equal sign* means:
  – “Is replaced by”
  – “Is assigned the result of”
  – **Valid** assignment examples:
    c=a+b, x=x+1

• Equal sign does **NOT** mean equality as in algebra
  – **Invalid** assignment examples:
    a+b=c; 3=x+y+99
What is going on??

```vba
Function sind(angle)
'Function to compute the sine of
'an angle in degrees
'angle=angle, degrees
'calculate the value for pi
Pi = 4 * Atn(1)
'calculate sine
sind = Sin(angle * Pi / 180)
End Function
```

These are comments. They are important.

VBA has some functions (like Atn* and Sin) but doesn't have others (like Pi)

*Atn is arctangent
## Some Built-In Numeric VBA Functions

<table>
<thead>
<tr>
<th>Purpose</th>
<th>VBA Function</th>
<th>Excel Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute value</td>
<td>Abs(x)</td>
<td>ABS(x)</td>
</tr>
<tr>
<td>Truncate to integer</td>
<td>Int(x)</td>
<td>INT(x)</td>
</tr>
<tr>
<td>Round x to n digits after decimal</td>
<td>Round(x,n)</td>
<td>ROUND(x)</td>
</tr>
<tr>
<td><strong>Square root</strong></td>
<td>Sqr(x)</td>
<td>SQRT(x)</td>
</tr>
<tr>
<td>Exponential, e</td>
<td>Exp(x)</td>
<td>EXP(x)</td>
</tr>
<tr>
<td><strong>Natural log</strong></td>
<td>Log(x)</td>
<td>LN(x)</td>
</tr>
<tr>
<td>Base-10 log</td>
<td>-</td>
<td>LOG10(x)</td>
</tr>
<tr>
<td>Base-b log</td>
<td>-</td>
<td>LOG(x,b)</td>
</tr>
<tr>
<td>Value of pi</td>
<td>-</td>
<td>PI()</td>
</tr>
<tr>
<td>Sine</td>
<td>Sin(x)</td>
<td>SIN(x)</td>
</tr>
<tr>
<td>Cosine</td>
<td>Cos(x)</td>
<td>COS(x)</td>
</tr>
<tr>
<td>Tangent</td>
<td>Tan(x)</td>
<td>TAN(x)</td>
</tr>
<tr>
<td>Arccosine</td>
<td>-</td>
<td>ACOS(x)</td>
</tr>
<tr>
<td>Arccosine</td>
<td>-</td>
<td>ACOS(x)</td>
</tr>
<tr>
<td>Arctangent</td>
<td>Atn(x)</td>
<td>ATAN(x)</td>
</tr>
<tr>
<td>Arctangent (4 quadrant)</td>
<td>-</td>
<td>ATAN2(x,y)</td>
</tr>
<tr>
<td>Degrees to radians</td>
<td>-</td>
<td>RADIANS(x)</td>
</tr>
<tr>
<td>Radians to degrees</td>
<td>-</td>
<td>DEGREES(x)</td>
</tr>
<tr>
<td>X modulo y</td>
<td>X Mod y</td>
<td>MOD(x,y)</td>
</tr>
<tr>
<td>Random number</td>
<td>Rnd()</td>
<td>RAND()</td>
</tr>
</tbody>
</table>

**Bold** indicates functions that differ between VBA and Excel.

Table from: Chapra, S.C. Power Programming With VBA/EXCEL, pg. 93.
Using Excel Functions in VBA

• Some functions can be called from Excel to use in your VBA code.

• Calling an Excel function in VBA:
  
  Application.WorksheetFunction.functionname(argument)

• Example: inverse cosine of 0.5
  
  Application.WorksheetFunction.acos(0.5)

• Do a help search on “Visual Basic Functions”
  
  – “List of Worksheet Functions Available to Visual Basic”
  
  – “Using Microsoft Excel Worksheet Function in Visual Basic”
Sending Multiple Arguments to a Function

- $A^2 + B^2 = C^2$
- VBA Function Hyp(A,B)
Flowchart Symbols

- Graphically represent sequential problems and decision choices.
- Typical Symbols are:
  - Start/Stop
  - Input/Output
  - Process Step
  - Test for Branching (decision)
  - Flow of Process

Start or Stop

Start or Stop

Input or Output

Input or Output

Input or Output

Test for Branching

Test for Branching

Flow of Process

Flow of Process

\[ x = x_{\text{min}} + 5 \]
If/Then/Else Statements in VBA

• VBA syntax for an if/then/else statement:

\[
\text{If condition Then} \quad \text{True statements} \\
\text{Else} \quad \text{False statements} \\
\text{End If}
\]

Indentation is IMPORTANT
If/Then Statements in VBA

• Do not need to include the **else** if it's not needed
• VBA syntax for an **If/Then** statement

\[
\text{If condition Then} \\
\quad \text{True statements} \\
\text{End If}
\]
If/Then/ElseIf Statements in VBA

• Can also have nested *If* statements
• VBA syntax for an *If/Then/ElseIf* statement

If condition 1 Then
    True statements 1
ElseIf condition 2 Then
    True statements 2
Else
    False statements
End If
Loops

• Used for repetitive calculations

• For Loops
  – Continue loop for known number of times
  – Repetitively execute statements
For/Next Loop

• If we know that the loop should run “n” times, there is a special loop to deal with these cases.

• For/Next loop VBA syntax:

\[
\text{For } \text{counter} = \text{start} \text{ To } \text{finish} \\
\text{Statements} \\
\text{Next } \text{counter}
\]
**Factorial Function**

**Function** `fact_calc(n)`

‘This Function will calculate ‘the Factorial of n

‘initialize fact_calc

`fact_calc=1`

For i = 1 To n

`fact_calc = fact_calc * i`

Next i

End Function

---

<table>
<thead>
<tr>
<th>i</th>
<th>fact_calc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>=fact_calc*i</td>
</tr>
<tr>
<td></td>
<td>=1*1 = 1</td>
</tr>
<tr>
<td>2</td>
<td>=fact_calc*i</td>
</tr>
<tr>
<td></td>
<td>=1*2=2</td>
</tr>
<tr>
<td>3</td>
<td>=fact_calc*i</td>
</tr>
<tr>
<td></td>
<td>=2*3 = 6</td>
</tr>
<tr>
<td>4</td>
<td>=fact_calc*i</td>
</tr>
<tr>
<td></td>
<td>=6*4 = 24</td>
</tr>
<tr>
<td>5</td>
<td>=fact_calc*i</td>
</tr>
<tr>
<td></td>
<td>=24*5 = 120</td>
</tr>
</tbody>
</table>
For/Next Loop Options

• Specifying step size

  For counter = start To finish Step increment
  Statements
  Next counter

• Default step size is one.
• If you want a different step size you can specify it.
• Example: For i = 1 To 10 Step 2
  Will count: 1,3,5,7,9
For/Next Loop Options

• Alternative way of exiting the loop:
  
  For counter = start To finish
  Statements
  If condition Then Exit For
  Statements
  Next counter

• The loop will terminate when it encounters an Exit For statement.
• Any number of Exit For statements may be placed in the loop.
• Exit For statements may be placed anywhere.
Functions Review

• We can now write our own functions in Excel
• Functions are good for...
  – Doing repetitive calculations
  – Returning a single value (answer) to the cell
• What if we want to do more?...
Writing Macros

• Ability to do many, many things including:
  – Formatting cells
  – Performing calculations
  – Grabbing data from cells
  – Putting data into cells
  – Display pop-up messages, dialog and input boxes
  – And more...

• Let's start with a simple example.
What is going on??

Forces you to define the type of all variables (good programming practice).

Declaring the macro program to be a subroutine program, “Factorial”. The parentheses are for the routine's arguments. In this case, there are no arguments.

Defining the variable types

Option Explicit
Sub Factorial()
'Names, Section #, Team #
'Date
'Variable Declaration
Dim n As Single, c As Single, initial As Single
Dim i as Integer
'input data
Sheets("Sheet1").Select
Range("B8").Select
n = ActiveCell.Value
Initial = 1
'calculation
For i = 1 To n
    'compute factorial
    c = initial * i
    initial = c
Next i
'output results
Range("B9").Select
ActiveCell.Value = c
End Sub
Declaring Variables

• Why declare variables?
  – Macros will run faster and use less memory
  – You will avoid problems with misspelled variable names

• VBA syntax for declaring variables:
  
  Dim varname As type, varname As type, ...

• Good practice to:
  – *Place all your Dim statements at the beginning of the program.*
  – *Group them by type.*
Numeric Variable Types

- **Integers** (whole numbers used for counting)
  - *Integer* type (range from -32,768 to 32,757 or 2 bytes)
  - *Long* type (range from -2,147,483,648 to 2,147,483,647 or 4 bytes)

- **Real or floating-point** (decimal numbers used for measuring)
  - *Single* type (about 7 significant digits of precision or 4 bytes)
    - Good for most engineering and scientific calculations
  - *Double* type (about 15 significant digits of precision or 8 bytes)
Variant Data Type

- Your code will work if you declare the variables like this, but not as you may expect...
  
  Dim x, v As Single

- VBA will make the following assignments:
  - v is a single type
  - x is a variant type
    - The variant type allows the macro to choose which data type it should be.

- **Warning**: this will slow down your macro.
<table>
<thead>
<tr>
<th>What is going on??</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grabbing data from your spreadsheet and assigning the values to variables</td>
</tr>
<tr>
<td>Outputting value of variable “c” to cell B9.</td>
</tr>
<tr>
<td>These are Object-Oriented Programming (OOP) statements</td>
</tr>
</tbody>
</table>

```
Option Explicit
Sub Factorial()

' Names, Section #, Team #
' Date
' Variable Declaration
Dim n As Single, c As Single, initial as Single
Dim i as Integer
' input data
Sheets("Sheet1").Select
Range("B8").Select
n = ActiveCell.Value

Initial = 1
' calculation
For i = 1 To n
    ' compute factorial
    c = initial * i
    initial = c
Next i
' output results
Range("B9").Select
ActiveCell.Value = c

End Sub
```
More Useful OOP Statements

• Clear a section of the worksheet
  – Range(“t3:z40”).ClearContents

• Move the current active cell
  – ActiveCell.Offset(0,1).Select
    • Moves the active cell 0 rows, 1 column to the right
  – ActiveCell.Offset(1,0).Select
    • Moves the active cell 1 row down, 0 columns
Running The Program

• Go back to Excel
  – Go to: Tools – Macro – Macros
  – Select the macro you want (Factorial)
  – Click the run button

• Or..
  – Go to: View – Toolbars – Forms
  – In the forms toolbar select the “button” button (rectangle)
  – Size the button on your spreadsheet
  – In the Assign Macro dialogue box, select Factorial
  – Rename the button to something descriptive like “Run Factorial”
## Simple Factorial Program

- **Results...**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Factorial Program</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Names</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Section #</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Team #</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Date</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Final Value</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Factorial</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Run