Week 6
Control Structures for Programming: Selection

The Story So Far

You should have some understanding of
- Subprogram structure: Sub Foo() ... End Sub
- Variable declarations: Dim myvar As Integer
- Types: Integer, Double, Boolean, String
- Assignment statements: myvar = -1
- Arithmetic expressions:
  myvar = 2*yourvar + hervar/3.5 - 1.6
- Running programs from VBE and the ribbon
- Displaying results:
  Debug.Print "Result is "; myvar
  ActiveCell.Value = myvar
  MsgBox "Result is "; & myvar

Program State

- The **program state** is the set of active variables and objects, external device state plus current execution position
- Control structures let the programmer cause actions to occur (or recur) based on decisions about the program state
- Main control structures are:
  - procedures (for grouping subtasks)
  - selection (do something or perhaps do something else)
  - iteration (do something as long as required)

Boolean Expressions

- All decisions are represented by **Boolean expressions**
- Boolean expressions evaluate to True or False
- Involve relational operators to compare values
  - VBA: =, <, >, <>, <=, >=
  - Maths: =, <, >, <=, >=
- Examples:
  2 < 3
  num1 > num2
  IntSum <> 0
  strName < "Zanzibar"
  NewDate > "2011-02-22"
  j = 3
  strID = "AA00"
  - context determines whether the last two are relational (comparison) expressions or assignment statements

Selection – If/Then

- The If/Then statement is used to make decisions using Boolean expressions
  - Simplest form:
    ```vba
    If boolean-expression Then
    statements
    End If
    ```
    - boolean-expression is evaluated
      - If it evaluates to True, statements are executed
      - otherwise (i.e., it must be False) skip over statements and continue with rest of program
    - Also referred to as a conditional statement

References

- Chapra (Part 2 of ENGG1811 Text)
  - Topic 18 (chapter 11) Decisions
  - Topic 11 (chapter 4) Customised Worksheet Functions
  - Topic 12 (section 5.2 to 5.4) Modular Programming
  Structure: two lectures + 1 hour worked examples
If/Then example

```plaintext
intAbsVal = intVal
If intAbsVal < 0 Then
    intAbsVal = -intAbsVal  ' unary minus*
End If
' now we know that intAbsVal = |intVal|
' regardless of whether intVal is positive or negative
' *much preferred to these change-sign equivalents:
    intAbsVal = intAbsVal * -1  ' yuk!
    intAbsVal = 0 - intAbsVal  ' double yuk!
```

More complex forms of If/Then

- Symmetric form:
  ```plaintext
  If boolean-expression Then
      statement-list1
  Else
      statement-list2
  End If
  ```
  - `boolean-expression` is evaluated
    - If it evaluates to True, `statement-list1` is executed
    - Otherwise, `statement-list2` is executed

More complex forms, cont.

Chained form, symmetric form generalised:

```plaintext
If boolean-expression1 Then
  statement-list1
ElseIf boolean-expression2 Then
  statement-list2
ElseIf boolean-expression3 Then
  statement-list3
Else
  ...
End If
```

Examples

' Larger value ' s = sign of a (+1,-1,0)
```plaintext
If a > b Then
    max = a
ElseIf a > 0 Then
    s = 1
Else ' a < 0'
    s = -1
End If
End If
' (chained form left as exercise)
' * justified a few slides later
```

Chained Selection example: Classification

Often need to classify a value based on ranges, such as deriving UNSW grade from mark (excluding PC):

```plaintext
' Precondition (assumption): 0 <= mark <= 100
If mark >= 85 Then
    grade = "HD"
ElseIf mark >= 75 Then  ' And Not (mark >= 85)
    grade = "DN"
ElseIf mark >= 65 Then
    grade = "CR"
ElseIf mark >= 50 Then
    grade = "PS"
Else
    grade = "FL"
End If
```

Boolean Expressions

Boolean expressions comprise
- Boolean constants True False
- Boolean variables
- Relational operations (comparisons giving Boolean values)
- Boolean operators
  - binary And Or Xor
  - unary Not
Truth Tables

Truth tables establish meaning of operators by enumerating each combination of operands and showing what the operation yields.

Notation: T = True, F = False

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A And B</th>
<th>A Or B</th>
<th>A Xor B</th>
<th>Not A</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

And – both True  Or – either True  Xor – either (but not both) True  Not – complement

Examples

- $x \geq 0.0$ And $x < 1.0$  'x lies between 0 (inclusive) and 1 (exclusive)
- $a = b$ And $b = c$  'all three are the same
- $j < 0$ Or $j > 0$  'equivalent to $j \neq 0$
- $x \geq 0.0$ Or $x < 1.0$  'true (why?)
- Not ($y > 0$)  'same as $y \leq 0$
- Not $\text{blnFound}$  'complement of variable
- Not ($E1$ And $E2$)  'xor $E1$ And $E2$
- Not ($E1$ Or $E2$)  'xor $E1$ And $E2$

Boolean Reasoning

Recall the $\text{sgn}()$ example:

If $a \neq 0$
If $a > 0$
Then $s = 1$
Else ' $a < 0$
' $s = -1$
...

How did we know that $a \neq 0$ at the point shown?
- We’ve passed the first condition, so $a \neq 0$
- But we’ve failed the second condition, so Not ($a > 0$)

Hence we have $a \neq 0$ And Not ($a > 0$)
- $a \neq 0$ And $a \neq 0$  'by definition of Not
- $a < 0$  'by definition of $<$ and $\neq$

Example: Leap Year

Simplistic form of leap year test for value stored in a variable called $year$:

year Mod 4 = 0

Complete form of leap year test (century years are not leap years unless they are divisible by 400, such as 2000, but not 1900 or 2100):

year Mod 4 = 0 And (year Mod 100 <> 0 Or year Mod 400 = 0)

Note the logic: divisible by 4 and either not a century or (is a century and) divisible by 400

Mixed And/Or expressions: do not rely on precedence: always use parentheses

Functions

- **Functions** are procedures (named groups of statements) that evaluate and return a single value of a specified type
  (subprograms perform actions but don’t return anything)
- VBA functions can be used in formulas in worksheet cells, or called from other parts of a program
- Declaration is always
  
  Function name(parameters...) As type declarations for local variables
  - statements, including somewhere...
  
  name = expression  Assign expression of the right type to the function name to return answer
  
  End Function

(examples will be provided)

Scope of Variables

- **Scope** is the region throughout which a variable is accessible (value used and/or changed)
- Variables declared outside procedures are accessible throughout the module (or between modules if fully qualified: Module1.MyList)
- Variables declared inside procedures are accessible only within that procedure
- Variables should be declared as locally as possible (to avoid ambiguity, clutter, confusion)
- Only persistent, important objects should have global scope
- Use parameters and function return for inter-procedure communication
Module Structure

Option Explicit
Const E = 2.718281828459045
Dim numWidgets As Integer
Sub IncrementWidgetCount()
    Dim intLocal As Integer
    intLocal = intLocal + INCR
    numWidgets = numWidgets + INCR
    Debug.Print "intLocal: " & intLocal & ", numWidgets: " & numWidgets
End Sub
Sub DoSomething()
    ' un-comment the next statement and try to compile
    intLocal = intLocal + 1
End Sub

Parameters

- Values are passed to a function (or subprogram) via
  parameters
- Look like variable declarations (and are)
- Corresponding values in call: arguments

UNSW Grade Function

- Embed algorithm (earlier slide) in function
- One parameter, mark (type is Integer), returns String
- Use on worksheet, fill down formula
- What if mark is out of range? Or not a number?
  (complete solution must cover all cases, Variant type enables this)

UNSWgrade with error checks

Function UNSWgrade(mark As Variant) As String
    If Not IsNumeric(mark) Then
        UNSWgrade = "Not a mark"
        Exit Function
    End If
    If mark < 0 Or mark > 100 Then
        UNSWgrade = "Mark out of range"
        Exit Function
    End If
    ' algorithm coded as before, rely on valid mark range; change variable name
    grade to function name
    UNSWgrade = "OK"
End Function

OK to return immediately after detecting error condition:
IsNumeric(VBA) vs IsNumber(worksheet). So much for consistency!

Tracing and Debugging

Suppose you tested UNSWgrades and it sometimes produced the wrong result. What do you do?
- Look at the code for incorrect logic (you’ll have to find the bug this way eventually)
To gather evidence about the problem,
- Include debugging statements at critical points
- Use the VB interpreter’s ability to pause execution and display state
- Try to pinpoint where the wrong value is assigned, or the wrong path was taken
- Go back to the first dot point.
Velocity of an object of mass $m$ falling under the influence of gravity and subject to drag is

$$v(t) = v_0 e^{-t/c_d} + \frac{g m}{c_d} (1 - e^{-t/c_d})$$

Where $v_0$ is the initial velocity, $g$ is acceleration due to gravity ($\text{ms}^{-2}$) and $c_d$ is the drag coefficient (in $\text{kg} \cdot \text{s}^{-1}$)

The exponential factor decays in magnitude, so the velocity asymptotically approaches $g m / c_d$

For a free-falling 70kg parachutist with $c_d = 12.5$, this terminal velocity is $\approx 55 \text{ m s}^{-2}$ ($200 \text{km/hr}$)

When the parachutist pulls the rip-cord ($t = t_c$), the same equation applies subsequently except that...

- $v_0$ is the velocity at the instant the parachute opens
- there is a new, larger, drag coefficient $c_{dp}$
- the time parameter is $t - t_c$ instead of $t$

We can write a function that, given all parameters, calculates the velocity at any time $t$
- The function can be filled down a worksheet
- The required algorithm is

```vba
If t < t_c Then
    Calculate free-fall velocity
Else
    Calculate velocity at time t₀
    Store in dblVelRipcord
    Calculate velocity using cdp and dblVelRipcord
End If
```

1. The common expression $e^{-t/c_d}$ should have its own function:

```vba
Function Decay(dblDrag As Double, dblMass As Double, dblTime As Double) As Double
    Decay = Exp(-dblDrag * dblTime / dblMass)
End Function
```

2. The terminal velocity should have its own variable (to make the expression more meaningful)
3. Chapra’s implementation does neither of these!

Selections from
- Fluid flow: calculating the Moody friction factor in a circular cross-section pipe carrying fluids of various kinds
- Damped oscillation of a mechanical object: displacement as a function of time and the dynamic characteristics of the system
- Triangle feasibility: given the lengths of the three sides, can a triangle be formed?

- Boolean expressions enable decisions to be made
- If-Then used for selection (three forms)
- Functions return a single value
- VBA functions can be used in worksheet formulas
- VB interpreter can be used to trace a program to assist in debugging