About the Other Lecturer

- Dr Geoff Whale is a UNSW Engineering graduate, sat up the back in CLB-7 in 1972 (yes, really).
- Senior Lecturer in the School of Computer Science and Engineering (CSE), now part time
- Also works on software design projects with UNSW IT Services (on myUNSW and others)
- Married*, two boys 20 and 24, both musicians
- Many interests are bushwalking, mapping, and writing and publishing walking guides

Week 5
Introduction to Programming and Visual Basic for Applications

References

- Chapra (std text Part 2), Topics 8, 9 (Chapters 1, 2).
- Some of this week’s material is derived from originals by Maurice Pagnucco

A Note on the Textbook

- Chapra is written in an informal style
- It covers topics in an unusual order, with key concepts deferred until late (such as data types and decision structures)
- It has many examples of poor programming practice, such as the use of absolute cell references, literals like 3.14159 instead of named constants (PI), and no indenting to show structure
  but
- It does eventually cover most of the principles that you need to apply in labs, assignments and exam

Notices – Week 5

- Mid-Session Exam
  - occupies the first part of the week 6 lab
  - can use MS Office apps only, no web access
  - answers in workbook/db also submitted
  - feedback in a week
- VBA Labs (weeks 6 to 11)
  - you must be prepared for each one
  - programming is not easy for many people
  - tutors will help with detail, but cannot teach you how to program
  - multiple tasks: subset OK (usually the first exercise) for students having difficulties
  - online assessment will require minimal effort

Context
### Summary of Slides 8–16

- Computers are used to model real-world situations
- A **problem** is a general question to be answered about a model
- A **problem instance** is where problem parameters are assigned specific values
- An **algorithm** describes how to solve a problem
  - Must be finite, correct, effective and definite
  - Steps may involve abstract
- Algorithms are implemented as **programs** using a particular language or notation
  - Abstract steps expressed using specific language features
  - Excel's formulas are a kind of limited programming notation
- Validate solutions at each part of the design process
  - Pick both typical and extreme instances
  - Algorithm: simulate steps; Test implementation directly
- Unlike many kinds of engineering design, program design is often iterative: implement, test, revise

### Problems and Problem Solving

- **Problem** is a general question to be answered
  - Usually has several parameters
- How would you specify a problem?
  - Describe nature of parameters
  - Describe questions we would like to ask about them
- **Easy example:**
  - Calculate the area of a circle
  - Parameters: radius of circle
- **Harder example:**
  - Solve a particular maze
  - Parameters: topology, must be **modelled** (represented in computing terms)

### Algorithms

- A **problem instance** is an assignment of values to parameters
- An **algorithm** for a problem is a general step-by-step procedure taking any problem instance with well-formed parameters and giving a correct answer for that instance
- An algorithm is correct (for a problem) if it gives the correct answer for every instance of the given problem

### Characteristics of Algorithms

Algorithm: a sequence of steps to accomplish a task
- **Input:** can have zero or more inputs
- **Output:** must produce one or more outputs in all cases
- **Finiteness:** must produce a result after a finite number of steps
- **Correctness:** must produce correct results in all cases
- **Effectiveness:** must consist of basic operations (each step is feasible)
- **Definiteness:** each step is precisely defined (specific course of action for every eventuality)

(see Knuth, "The Art of Computer Programming", Volume 1)

### Algorithms vs. Programs

- Aren’t they the same thing?
  - A computer can only perform tasks specified by simple operations it can execute
  - We need to **express** how the task can be completed in these terms; which operations to perform in what order
  - This description is known as an **algorithm**
  - A **program** is a representation of an algorithm that can be executed by a computer
Your Turn (#1, easy)

- Devise an algorithm to calculate the average of two real numbers
  - obvious, solved with Excel in a jiffy
  - different implementations may use different notations, but the algorithm is essentially the same

Your Turn (#2) The Celebrity Problem

A Celebrity is someone whom everybody knows, but doesn’t (need to) know anybody.

Who knows whom is represented by a matrix of order N. Each matrix cell is 1 if Person A knows Person B, and 0 otherwise (e.g., person 6 knows 3 but 3 doesn’t know 6)

Devise an algorithm to find the celebrity (if any), inspecting as few cells of the matrix as possible.

Analysis and two solutions discussed in lecture

Designing Algorithms

- Need clear specification of problem at hand
- Think of all situations that may arise and know what output to expect
- Does this resemble a standard problem (many identified; some broad classes exist)?
- Even if problem appears to be a new one, it can often be attacked by a small number of general strategies
- Once obtained, need to analyse algorithm for memory consumption, speed, etc.
- May need to repeat this a few times

Algorithm Correctness

- Algorithms can be complex and the tasks they solve difficult
- Errors are easily introduced
- Bugs: cost can be expensive (not only financially)
- Can reduce incidence of bugs in two ways: testing and proving
  - Testing: executing program on (lots of) test data – you can do this and must do this
  - Proving: certifying program produces correct result on all permissible data (rarely easy, plus errors may be introduced during coding) – you probably can’t do this

Programming

- We will be programming using Visual Basic for Applications (VBA)
- VBA is bundled with all recent versions of Microsoft Office except Office 2008 for Macintosh
- Programs are edited and run within Office apps (especially MS Excel)
- Programs saved with the document
- Why program?
  - Allows a much greater range of problems to be solved than can be done with Excel’s built-in features
  - Extends query processing in Access
  - Automates repetitive functions in Word, Access, Excel, or PPT
- We will just use VBA with Excel for this lecture series

Why is Programming in ENGG1811?

1. Useful in its own right (extends the application and thus the range of available solutions)
2. As a professional engineer, you will need to communicate with software designers/developers
   - need to know what’s achievable using straightforward programming principles (like VBA coding)
   - need to appreciate the complexities and process involved in development, and something of the software development lifecycle:
Visual Basic

- VB and VBA use structures inherited from BASIC (Beginners All-purpose Symbolic Instruction Code) with many extensions
- VBA is stored with document, and can interact directly with document data or other apps
- VB/VBA allows for object-oriented programming (OOP) like C++, Java, etc.
  - OOP helps programmers solve problems by providing a convenient method for problem decomposition

File Formats

- Microsoft maintains compatibility (mostly) with older formats
- Common format .xls derives from Excel 97 (and still usable)
- Excel 2007 introduced new formats:
  - .xlsx workbook without stored VBA
  - .xlsm workbook with VBA
- Be extremely careful when you use Save As
  - if the saved format is Excel workbook (*.xlsx) and you continue past a dialogue warning box you will lose all your VBA code (program text)
- Excel 2010 and 2007 use identical formats

Creating VBA

- VBA program code can be created by
  - Excel etc, using the Macro Recorder
    - Developer tab – Record Macro ...
  - The user, with the Visual Basic Editor (VBE)
    - Developer tab – Visual Basic
    - or press Alt-F11
    
      (Excel2003 menu path: Tools – Macro – ...) 
    
    Macros can be edited and incorporated into larger programs
  - This is an easy way to find out how to access and change Excel objects

Developer tab? What Developer tab?

- Microsoft are so paranoid about users hurting themselves with the tools they sell that they deliver the software with VBA disabled (a bit like buying a circular saw without blades)
- See Excel2007Notes (PDF on class web page) for a full description of how to setup your home system
- To get the Developer* tab back: press and

Visual Basic Editor

* Developer: one who designs and implements software, but apparently includes people who simply want to record and replay macros without coding.
**Initiating Action**

VBA program code can be run or executed by:
- Selecting Developer tab - Macros (Run button) (or press Alt-F8)
- Calling a VBA function from a worksheet formula
- An event occurring in the application, such as opening a document or creating a worksheet
- An interactive event such as a mouse click
- Linking code to a VBA control (button etc) placed on the document
- Via a form created using the VBE

*Note: the VBE is virtually identical in Office 2003/2007/2010* See Excel2007notes for how to enable macros

**Creating a Sample Program**

- First kind of initiation (user-initiated)
- Framework provided by VBE:
  - Select Insert – Module from menu
  - Module1 created, Project Explorer:
    - Type `Sub SwapAdjacent()`
    - Shows skeleton (Sub = subprogram):
      ```vba
      Sub SwapAdjacent()
      End Sub
      ```
- Add VBA statements before `End Sub` (listing over)

**First Program**

```
' This subprogram exchanges the contents of the active cell with the contents of the cell to its right.
' Comments like this start with a single quote ('),
' and are used to document the program.
' They have no effect on execution.
Sub SwapAdjacent()
  Dim tmp ' names a temporary location to hold a value
  tmp = ActiveCell.Value ' save value first
  ActiveCell.Value = ActiveCell.Offset(0,1).Value
  ActiveCell.Offset(0,1).Value = tmp
  MsgBox 'Swapped cells at ' & ActiveCell.Address ' is inserted by the editor
End Sub
```

**Program Execution**

- Return to Excel, place values in adjacent cells, select left-hand one
- press Alt-F8
- Select SwapAdjacent and press Run
- Cells are swapped, then a message appears

**Program Components**

- Comments (begin with single quote '), ignored)
  - Comments are important and serve to explain code, improving its readability, have no effect on execution
- Subprogram (can be executed by user)
  - between `Sub name()` and `End Sub`
  - VBA procedures are either subprograms or functions
- Variables (Dim name, or Dim name As type)
  - names locations that can be used in calculations
- Assignment (variable = newvalue)
  - fundamental programming operation

**Program Components**

- `object . property`
  - `ActiveCell` is a built-in object
  - `Offset(row, col)` refers to another cell relative to the address of the object
  - `Value` is the normal displayed property
- `MsgBox`
  - Built-in VBA procedure to display dialogue box
  - `"..."` are literal strings
    - used for displaying text of some kind
- `&` operator concatenates (joins) strings
  - same notation as used in formulas
Reserved Words

- Words like Dim, Sub, End are known as **reserved words** or keywords in VB.
- You cannot use them as variable names, procedure names, etc.
- Standard procedure names like MsgBox are not reserved but avoid them to prevent ambiguity.
- VBE highlights reserved words in blue.

Identifiers

- Words like tmp in the example program are called **identifiers**.
- Identifiers are used for names of procedures, variables, and properties.
- Identifiers are sequences of letters (a-z, A-Z), digits (0-9) and underscores (_).
- Identifier can only begin with a letter.
- Examples of valid identifiers:
  - Module1, x42, temp, blnFound, y_origin
  - Invalid identifiers:
    - 2day, $24, see-saw

Identifier Conventions

- Identifier conventions have been devised to make programs more readable:
  - Use title case for procedure names:
    - FindEmptyCell, IsNumeric, ToString
  - Prefix non-trivial variable names with type (more on this later):
    - dblTemperature, intCount, blnFound
  - OK to use short names for minor or short-lived data, as in the second sample program (overleaf).

A Second Program

- Most VBA programs are collections of procedures stored in **modules** and initiated by forms and buttons.
- Our second program will be stored this way, but initiated within the VBE:
  - Select Insert -- Module from menu.
  - Module1 is created (but empty).
  - Project Explorer window shows.
- Type program (overleaf) into Edit window.

```vbnet
Option Explicit

Sub ShowSum()
    ' Declare variables for storing numbers and total
    Dim num1 As Integer
    Dim num2 As Integer
    Dim total As Integer

    ' Assign (a constant) value to first variable
    num1 = 12

    ' Assign value to second variable
    num2 = -3

    ' Add the numbers together and store the sum
    total = num1 + num2

    ' Show result on immediate window
    Debug.Print "The sum of "; num1; " and "; num2; " is ", total

    ' Copy to worksheet too
    ActiveCell.Value = total
End Sub
```

Demo

Includes a quick look at the execution tracing tools used to find out what a program is doing (and find out why it may be misbehaving)

```vbnet
Option Explicit

Sub ShowSum()
    ' Declare variables for storing numbers and total
    Dim num1 As Integer
    Dim num2 As Integer
    Dim total As Integer

    ' Assign (a constant) value to first variable
    num1 = 12

    ' Assign value to second variable
    num2 = -3

    ' Add the numbers together and store the sum
    total = num1 + num2

    ' Show result on immediate window
    Debug.Print "The sum of "; num1; " and "; num2; " is "; total

    ' Copy to worksheet too
    ActiveCell.Value = total
End Sub
```
Formulas vs Programs

- Excel’s formulas are **functional**
  - specify what the answer should be as a single large expression
  - if too complex, intermediate values have to be stored in cells
- VBA statements are **procedural**
  - each one is executed in turn
  - all storage locations are explicitly named
  - location values can be updated

\[ =E2+F4 \quad \text{(in cell A2, say)} \]
\[ \text{num1} = 12 \]
\[ \text{num2} = -3 \]
\[ \text{total} = \text{num1} + \text{num2} \]

Running a Macro

- The second program can be run from within the VBE. A **macro** is just a subprogram that can be started by the user from Excel’s menus.
  - Select Run – Macros from VBE menu
  - Pick ShowSum from the list (the only one)
  - Press Run
- Results appear on the Immediate window, and in the active cell (better interaction with worksheets comes later):

```
Immediate

The sum of 12 and -3 is 9
```

(not very interesting so far, hang in there, it gets better)

Program Style

Programs are both for the computer to run and for people to read

- program code is hierarchical (statements are inside Sub ShowSum, so **indent**
- leave white space (between elements and between lines) for clarity
- long lines continued with space and underscore —
- VBE helps by
  - capitalising keywords (but not your identifiers until declared)
  - spacing between elements (but not between lines)
- add meaningful comments
  - before procedure explaining purpose, parameters
  - next to important variable declarations
  - before or next to important statements

Variables

- Variables store values for calculation and later use
  - These values are actually stored in the computer’s memory
- Variables need to be **declared** before use with the keyword **Dim**
- Each variable has a **data type** describing the range of valid values
- Variable names are identifiers (see earlier rules for valid identifier names)
- Name prefix indicates data type (later)

```
Dim var1 As datatype, var2 As datatype, ...
Dim intX As Integer, intY As Integer
  - Their data type is Integer (i.e., whole numbers); these variables can be assigned integer values of either sign, but only up to a limit
Dim dblArea As Double
  - Double = real number approximation using double precision (about 16 significant figures)
Dim strUserName As String
  - Declares one variable – strUserName
  - Data type is String – a sequence of characters
```

Data Types

- Each variable must have an associated data type
- The data type determines what values can be assigned to variables
- Also determines the amount of memory required to store value of variable
- Data types are important because they allow the compiler to check for errors in program
- Program also uses data types to determine how to convert a value of one type to another (e.g., an integer to a string)
- **Variable prefix** convention used to help readability (unless following some obvious algebraic convention)
VBA Primitive Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Prefix</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bln</td>
<td>True, False</td>
</tr>
<tr>
<td>Byte</td>
<td>byt</td>
<td>0-255</td>
</tr>
<tr>
<td>Date</td>
<td>dtm</td>
<td>Dates and times</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
<td>Whole numbers, -32768 to 32767</td>
</tr>
<tr>
<td>Long</td>
<td>lng</td>
<td>Large integers, +/- 2 billion or so</td>
</tr>
<tr>
<td>Single</td>
<td>sng</td>
<td>Floating point (real numbers, ~ 7 dec digits)</td>
</tr>
<tr>
<td>Double</td>
<td>dbl</td>
<td>Higher precision floating point (~ 16 dec digits)</td>
</tr>
<tr>
<td>Object</td>
<td>obj</td>
<td>Generic structured data type</td>
</tr>
<tr>
<td>String</td>
<td>str</td>
<td>Sequence of characters, variable length</td>
</tr>
<tr>
<td>Currency</td>
<td>cur</td>
<td>Monetary value with up to 4 dec places</td>
</tr>
<tr>
<td>Variant</td>
<td>vnt</td>
<td>Dynamic data type (used in special cases)</td>
</tr>
</tbody>
</table>

The most important are Boolean, Integer/Long, Double, String and Variant.

Assigning Values to Variables

• A variable can be assigned a value using the assignment operator =
  \[ Var = Expression \]

  - \( Expression \) is evaluated and the result stored in the location named by the variable \( Var \)
  - Replaces any previous value

• Examples:
  \[ Total = 2 + 3 \] \( \text{constant expression} \)
  \[ dblArea = 2 * PI * dblRadius \] \( \text{real expression} \)
  \[ strGreeting = "Hello World!" \] \( \text{literal string} \)
  \[ intYing = intYang \] \( \text{copy variable value} \)
  \[ blnCorrect = (Total = 5) \] \( \text{(last one is a comparison assigning True or False)} \)

Constant Definitions

• Fixed or \textit{constant} values are often required at several places in a program
• By giving a name to the constant...
  - The reader understands what the value \textit{means}
  - for example, only hard-core physicists would recognise \( 1.3806503 \times 10^{-23} \) in an equation (it’s Boltzmann’s constant)
  - The value could be changed in one place later if new conditions apply (limits or resource requirements)
• Name format convention: ALL_CAPS

  \begin{align*}
  \text{Const} & \ PI = 3.141592653589793 & \text{\textit{fundamental value}} \\
  \text{Const} & \ BOLTZ = 1.3806503e-23 & \text{\textit{k}\text{B in J/K}} \\
  \text{Const} & \ DAYS_IN_LEAP_YEAR = 366 & \\
  \text{Const} & \ MAX_SHEETS = 16 & \text{\textit{some limit}} \\
  \text{Const} & \ DEBUGGING = True & \text{\textit{controls output}} \\
  \text{Const} & \ VERSION_CODE = "V1.0 beta" & \text{\textit{info}} \\
  \end{align*}

Arithmetic Expressions

• Used to perform numeric calculations (real or integer)
• Can comprise
  - Literal constants (152, -3, 12.75, 1.39e7)
  - Named constants (PI, MAX, NUM_SHEETS)
  - Numeric variables (dblX, intTotal)
• Arithmetic operators: +, -, *, \, /, \textit{Mod}, ^
• Parentheses: ( )

### Examples of Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 + 2 * 3 - 4</td>
<td>3 (not 5)</td>
</tr>
<tr>
<td>5 / 2</td>
<td>2.5</td>
</tr>
<tr>
<td>5 \ 2</td>
<td>2</td>
</tr>
<tr>
<td>14 Mod 5</td>
<td>4</td>
</tr>
<tr>
<td>2 ^ 3</td>
<td>8</td>
</tr>
</tbody>
</table>

| intSum + 1 | curPrincipal * (1 + dblRate) ^ intYears |
| (a + b) Mod 18 | (R1 * R2) / (R1 + R2) |
| a^n * b^m * c |
### Precedence

- When evaluating arithmetic expressions, order of evaluating operations determined by **precedence**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Higher precedence</th>
<th>Lower precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- (unary: sign)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* / Mod (remainder)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ - (binary: add, subtract)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- You can look this up when needed

### Evaluating Expressions – Rules of Precedence

- When evaluating expressions, operations of higher precedence are performed before those of lower precedence

  \[
  2 + 3 \times 4 = 2 + (3 \times 4) = 14
  \]

- Otherwise operations performed from left to right

  \[
  2 ^ 3 ^ 4 = (2 ^ 3)^ 4 = 4096
  \]

  \[
  10 + 2 - 3 = 9
  \]

- Use parentheses if in any doubt

### Summary

- Algorithms express solutions to problems
- Programs implement algorithms
- VBA is a particular language with its own way of representing data and action
- VBA is bundled with MS Office since 97
- Use the VBE to edit and test
- Programming concepts
  - procedures for grouping code
  - variables, types, constants
  - assignment (change value of variables)
  - arithmetic expressions for evaluation