HELP FOR DEBUGGING VB CODES
Debugging is an essential part of programming. Even for an experienced programmer, most codes will not work in the first attempt due to typo, mathematical errors and logical flaws. These errors or bugs have to be corrected before your program will yield desired results. The goal of this tutorial is to introduce you to tools that can help fix your buggy code. There are 3 kinds of errors (bugs) in the code:

Compilation errors or syntax errors: When you run your VB code, the visual basic compiles your code into binary language which your computer can understand. If the compiler comes across some syntax that it cannot understand, it issues a compiler error. Most of the compiler errors are typos. For example, typing “Iteger” instead of “Integer,” not ending an If statement or a For loop. They are easy to fix and are caught by the compilers and pointed out to you. Adding OPTION EXPLICIT will help flag most typographic errors in the variable names.

Run time errors: These are errors which occur when the program encounters a mathematically incorrect situation under certain conditions. For example, in an expression x = a/b; the effort will occur only when b becomes zero.

Logic errors: They are due to a wrong logic in the code. These are difficult to debug since it needs expert knowledge. For example, if the programmer used a wrong formula for a system (say computing moment of inertia) then only an expert can identify the problem!

The most straightforward method of debugging is to carefully go through the code line-after-line and see if you can identify the problem. Sometimes, a visual inspection may be sufficient to fix most errors. But if visual inspection does not indicate an error, then debugging tools can come handy.

Tip: You can use Microsoft Visual basic HELP to provide some assistance. By clicking help and typing a keyword or by selecting a keyword and pressing F1 you can get information about the key word. For example typing “single” shows what the data type "single" means. It also has examples of programs using that uses this keyword.
VBA Error messages

VBA has built-in error detection features for debugging common mistakes. For example, we have included an error in the below VB code (example from Chapra, 2003), when you type it in and try to run, there will be an error display as shown below in screenshot.

```
Option Explicit
Option base 1
Private Sub CommandButton1_Click()
Dim a As Single
a = 5_3
End Sub
```

Another example (from Chapra, 2003) involves division by zero (a common error) as shown below. Type in the following code and run.

```
Option Explicit
Option base 1
Private Sub example_Click()
Dim a As Single, b As Single, c As Single
a = 5
b = 0
c = a / b
End Sub
```

When you click the debug mode, something different happens, the line with error is highlighted and you will be able to see the values by pointing to the name of the variable as shown below.
You can also use the active debugging option using the functions in debugging toolbar in the visual basic editor. Create a program (from Chapra, 2003) below using command button method.

```vba
Option Explicit
Option base 1
Private Sub CommandButton1_Click()
Dim pi As Single, radius As Single, area As Single, volume As Single
pi = 3.14159
radius = Cells(3, 2) ' Read the value of radius from excel

'compute area
area = pi * radius ^ 2

'compute volume
volume = (4 / 3) * pi * radius ^ 3

'Output area
Cells(5, 2) = area
Cells(6, 2) = volume
End Sub
```

Now go to the debug drop down menu and select *Step Over* (F8), you can see a yellow highlighted line. If you press the F8 button, the highlighted line moves down one line at a time. If you move the cursor over the variables, you can see their current value in a box. To stop debugging press reset button in the debug drop down menu.
Break Points
Since moving one step at a time can be painful, there is another debugging tool available called “breakpoint.” When you run a program, the execution stops at the selected line and you can check the values. Visual basic shows a large brown dot in the margin to remind that there is a breakpoint in that line. For example if you wanted to check the volume calculation, you would put a breakpoint in the line for volume calculation. You can put a breakpoint by clicking in gray area in the left of the code. If you click once you will see red DOT, which indicates breakpoint. To clear the breakpoint, simply click on the DOT again. When you run the code, it will stop at the line where a breakpoint is placed. You can use multiple breakpoints in a code.

```vba
'compute area
area = pi * radius ^ 2

'compute volume
volume = 4 / 3 * pi * radius ^ 3
```

Debugging Loops
Debugging within loops is a particularly difficult task. Type in the following program and step through it using the F8 button. Observe that the program crashes for i=5, why?

```vba
Option Explicit
Option Base 1
Private Sub Fx_calculator_Click()
Dim i As Integer, fx As Single
    For i = 1 To 5
        fx = 1 / (i - 5)
    Next i
End Sub
```

Tip: To come out of debug mode, use the Reset button. To check if the program is running properly, make calculations using a calculator and compare with values from the program. You can also use a breakpoint within the loop to stop at every loop.

Review Problems [examples from Chapra (2003)]:
Use the quadratic formula to solve for the roots of the following equations
a) \( x^2 - 5x + 2 = 0 \)
b) \( 5x - 3 = 0 \)

c) \( x^2 - 2x + 3 = 0 \)

The following VBA procedure performs the same calculations. Enter it into the VBE

```
Option Explicit
Option base 1
Private Sub Quadroots_Click()
  Dim a As Single, b As Single, c As Single, d As Single, r1 As Single, r2 As Single
  MsgBox "Enter the coefficient for the quadratic equation: \( ax^2 + bx + c=0 \)"
  a = Cells(4, 2)
  b = Cells(5, 2)
  c = Cells(6, 2)
  ' Quadratic formula
  d = b ^ 2 - 4 * a * c
  r1 = (-b + d ^ 0.5) / (2 * a)
  r2 = (-b - d ^ 0.5) / (2 * a)
  Cells(8, 2) = r1
  Cells(9, 2) = r2
End Sub
```

Debug the program so that it yields correct result for equation a).
When you apply the program to equation b) and c), what error messages does VBA display? What mathematical problems actually caused the errors?

2) Enter the following code into the VBA and add breakpoints as shown. Run the program and record the value of I as each breakpoint is reached. Describe how the loop operates in light of your results.

```
Option Explicit
Option Base 1
Private Sub ForNext_Click()
  Dim i As Integer
  For i = 8 To -4 Step -5
  Next i
  'after the loop
End Sub
```