Visual Basic Programming for Excel

Zhenming Su, Ph.D.

Institute for Fisheries Research
212 Museums Annex Building
1109 N. University Ave.
Ann Arbor, MI 48109-1084
USA

Phone: (734) 663-3554 Ext. 123
Fax: (734) 663-9399
Email: SUZ@michigan.gov
Website: http://personal.www.umich.edu/~zsu/

Part 1. Programming Basics

1. A Small VB program

Example 1 **Demonstration of a VB program**

```
Sub ASmallVBProg()
    ‘Input
    x = Range(“A1”).Value
    y = Range(“A2”).Value

    ‘Calculation
    z = x+y

    ‘Output
    Range(“A3”).Value = z
    Range(“A3”).Interior.ColorIndex = 3 ‘Red
    Range(“A3”).Font.ColorIndex = 4 ‘Green
End Sub
```

ANALYSIS This ‘sub’ program demonstrates the ease of using VB to solve simple problems. Any programs, large or small, have three basic parts: (1) *Input:* Obtaining data from outside. Here we get data for x from cell “A1” and for y from cell “A2” of an Excel spreadsheet “sheet1”. (2) *Processing part:* doing calculations. Here the calculation is z = x + y. (3) *Output:* exporting your results to outside place. Here we put the value of z to the cell “A3” and change the color of that cell to highlight the result.

2. Writing, Editing, Running and debugging Code in the VB Editor

VB editor is a programming environment, using it we can create, edit, debug, and run our programs.

2.1 Open VB Editor

In Excel, to enter the VB Editor, open the **Tools** menu, point to **Macro**, and then click **Visual Basic Editor.**

2.2 Write and Edit your program in VB Editor

Find the **Project Explorer** window on the VB editor, double click "sheet1 ("Example1")" in it. This way you open a code window of this sheet. This code window is the place you enter you programs.

Enter the program above line by line, watching what happens after you input each statement.

2.3 Run the code

(1) In Excel sheet1 "Example1", put 10 in cell "A1", and put 20 in cell "A2";
(2) Open **Tools** menu, point to **Macro**, then click **Macro…** button;
(3) From **Macro dialog box**, run the program named **ASmallVBProg.**

Or

You can run a program in the VB Editor by placing the insertion point anywhere in the program you want to run, and then press F5.

2.4 Edit and debug your program

(1) Click the left edge of the code window corresponding to a line you want to check, see what will happen. (The shortcut for setting breakpoint is F9);
   - You set a breakpoint to suspend execution at a specific statement in a procedure; for example, where you suspect problems may exist. You clear breakpoints when you no longer need them to stop execution.
   - To clear a breakpoint

   Position the insertion point anywhere on a line of the procedure containing the breakpoint. From the Debug menu, choose Toggle Breakpoint (F9), or click next to the statement in the Margin Indicator Bar (if visible.). The breakpoint is cleared and highlighting is removed.
(2) Re-run the program in Excel following step 2.3 or run it in the VB editor by pressing F5. The program will stop at the breakpoint and that line will be highlighted. Then you can run the program line by line by pressing F8. Put the mouse pointer in a variable name and see what happens.
(3) As you note in the last step, when you position the mouse pointer in a variable name, the present value of that variable will show up. Combining breakpoint setting and this step, you can check your program for possible bugs.

Part 2 Language

1. Variables, Constants, Arrays, Operators and built-in functions

(1) Data type

In VB, as in all high-level programming languages, you use variables and constants to store values of different data type. Such as, a name is usually expressed as a string; a count will be a integer; the length of a fish will be a real number and the answer to the question "Are you married?" will be "True" or "False". Data type is important, because we process each data type with particular operators, functions and expressions and is stored in the computer by different ways. Each date type has its own syntax. The data types provided by VB are :

Byte, Integer, Long, Single, Double, String, Boolean, Data, Currency, Object

(2) Declaring a Constant, Variable, or Array

Constant: When you first begin programming, you will find that some of your programs require you to use the same value over and over, sometimes this value may be difficult to remember or too long to enter. Such as when your program does some calculations on the area of some circles, you must use Pi values, for instant for 8 times. You must put Pi value in your code for 8 times! You can make your code easier to read and maintain using constants. A constant is a meaningful name that takes the place of a number or string that does not change. *The value of a constant cannot change during the execution of the program.*

Declare a constant:

Const MyVar = 459

Const Pi=3.141596

Const NewYear=01/01/1999

<u>Variable</u>: A variable is similar to a constant in that it is a name given to a value, but with one big difference. The value of a variable can change during the program execution.

Usually using keyword **Dim** to declare a variable:

Dim I,j,k as integer

Dim x,y,z as double

Dim aString as string

Dim myMoney as currency

Dim IsGraduateStudent as Boolean

(Also see: **Private, Public, Static**)

<u>Array</u>: An array is a data type that represents *a bunch of data of the same data type*. We use one name for each array, using indexes to distinguish the elements of this bunch of data. For example:

For 15 Chinook caught in a trip, we may using w(1),w(2),w(3),…,w(15) to represent the weight of each fish, instead of giving the weight of each fish a separate name.

The purpose of setting up an array type is simplifying the coding of these kinds of data.

Before you use an array in the code, you must define (or declare) it. Declaring an array is similar to that of a variable, but here we must also need to indicate the size and the dimensions of an array. One-dimensional arrays represent sets of data, such as the chinook weights. In math, they are called vectors. Two-dimensional arrays represent data tables, such as the response of the fish to the combined changes of light and temperature. In math, they are called matrixes. Multi-dimensional arrays represent changes with multiple factors.

<u>Declaring one-dimensional array:</u>

Dim w(1 to 15) as double ‘*Array w, one-dim, size 1 to 15, data type--double*

Dim strArray(0 to 100) as string ‘*String type array strArray, one-dim, size 0 to 100 (it can contain 101 elements)*

Dim x(10) as single ‘ *Single type array, one-dim, can contain 11 elements (0 to 10)*

<u>Declaring two-dimensional array</u>:

Dim fishResp(1 to 15, 1 to 100) as boolean *‘Array fishresp, two-dim; 1st dim--size 1 to 15, 2rd dim 1 to 100; data type--boolean*

<u>Access elements of an array</u>: you use the name of an array and an index to access an element. For example:

W(1) = 25.0 *‘Assign 25.0 to the 1st element of the array w*

W(2) = 50.0

StrArray(9) = “A jack sockeye”

FishResp(2, 6) = true *‘Assign boolean value true to element (2,6) of the array fishResp*

We will practice Example 2 through 14 in worksheet "Example2-14"

Example 2 **Simple program using a one dimensional array**

```
Sub ArrayDemo()
  Dim sum As Double
  Dim w(1 To 10) As Double


  w(1) = Sheets("Example2-14").Range("B2").Value
  w(2) = Sheets("Example2-14").Range("B3").Value
  w(3) = Sheets("Example2-14").Range("B4").Value
  w(4) = Sheets("Example2-14").Range("B5").Value


  sum = w(1) + w(2) + w(3)+w(4)


  Sheets("Example2-14").Range("E2").Value = sum
End Sub
```

ANALYSIS This program declared an array called w, it can hold 10 real values. Then we loaded data for its first four elements: w(1), w(2), w(3), w(4) and calculate the sum. But see how ugly the program is! It uses 4 lines to express a repeated action: load a value for an element of an array. As we learn how to use loops, we can use array more elegant.

(3) Operators: +, -, *, /, ^ (power), AND, OR

(4) Standard or Built-in functions: Four confusing functions

1) Int(x) Returns the integer portion of a number.
Fix(x)

Eaxample:

```
MyNumber = Int(99.8)     ' Returns 99.    MyNumber = Fix(99.2)     ' Returns 99.
MyNumber = Int(-99.8)    ' Returns -100.  MyNumber = Fix(-99.8)    ' Returns -99.
MyNumber = Int(-99.2)    ' Returns -100.  MyNumber = Fix(-99.2)    ' Returns -99
```

This example illustrates how the Int and Fix functions return integer portions of numbers. In the case of a negative number argument, the Int function returns the first negative integer less than or equal to the number; the Fix function returns the first negative integer greater than or equal to the number.

2) Log(x) Returns the natural logarithm of a number. Pay special attention to this function. Because in the worksheet, you use "=ln(x)" calculate the natural logarithm.

3) sqr(x) Returns the square root of a number. This is also confusing. In the worksheet you use sqrt(x) to calculate the square root.

To learn more about VB functions, use Help to look up math functions.

2. Conditional Statement

Until now we can only use VB to do some simple, straightforward stuff. VB has more powerful statements to control the executions of your application. They are *conditional statements* that switch the direction of execution according to some conditions, which is the magic that makes your program thinking and making decisions, and *loop statements* which make your program to have replicate ability.

First we introduce conditional statements of VB

(1) If… Then…

This statement can let your program do something according to a condition. Look at this example:

Example 3 **Simple program using If…Then statement**

```
Sub IfThenDemo()
        Dim sum As Double
        Dim w(1 To 10) As Double

        w(1) = Sheets("Example2-14").Range("B2").Value
        w(2) = Sheets("Example2-14").Range("B3").Value

        'If the first fish is heavier than the 2nd one (or with the same weight)  then color it with red
        If w(1) >= w(2) Then
           Sheets("Example2-14").Range("B2").Interior.ColorIndex = 3
        End If  'Do not forget end if

        'If the first fish is lighter than the 2nd one then color it with green
        If w(1) <= w(2) Then
           Sheets("Example2-14").Range("B2").Interior.ColorIndex = 4
        End if
End Sub
```

(2) If…Then…Else…
Example 3 can be improved by if…then…else…

Example 4 **<u>If…Then…Else statement</u>**

```
Sub IfThenElseDemo()
        Dim sum As Double
        Dim w(1 To 10) As Double

        w(1) = Sheets("Example2-14").Range("B2").Value
        w(2) = Sheets("Example2-14").Range("B3").Value

        If w(1) >= w(2) Then
           'If the first fish is heavier than the 2nd one (or with the same weight) then color it with red
           Sheets("Example2-14").Range("B2").Interior.ColorIndex = 3
        Else
                'If the first fish is lighter than the 2nd one then color it with green
                Sheets("Example2-14").Range("B2").Interior.ColorIndex = 4
        End if
End Sub
```

Example 5 **<u>Nested if statements</u>**

```
Sub NestedIf()
      Dim HeaviestFish, sum As Double
      Dim w(1 To 10) As Double

      w(1) = Sheets("Example2-14").Range("B2").Value
      w(2) = Sheets("Example2-14").Range("B3").Value
      w(3) = Sheets("Example2-14").Range("B4").Value

      'Select the heaviest fish among the first three fish and color it with red

      If w(1) >= w(2) Then
         If w(2) >= w(3) Then
```

```
            HeaviestFish = w(1)
            Sheets("Example2-14").Range("B2").Interior.ColorIndex = 3
          End If
        Else
          If w(2) >= w(3) Then
            HeaviestFish = w(2)
            Sheets("Example2-14").Range("B3").Interior.ColorIndex = 3
          Else
            HeaviestFish = w(3)
            Sheets("Example2-14").Range("B4").Interior.ColorIndex = 3
          End If
        End If
    End Sub
```

Note: you must match each If with an End If in the correct place

(3) Select Case
Sometimes a condition may have multiple results. Decision must be made according to one of the results. In such case you can use **Select Case** statement.

Example 6 **Select Case Statement**

```
Sub SelectCaseDemo()
    Dim Number
    Number = 8  ' Initialize variable.

    Select Case Number  ' Evaluate Number.
       Case 1 To 5 ' Number between 1 and 5.
            Debug.Print "Between 1 and 5"
       ' The following is the only Case clause that evaluates to True.
       Case 6, 7, 8    ' Number between 6 and 8.
                 Debug.Print "Between 6 and 8"
       Case Is > 8 And Number < 11 ' Number is 9 or 10.
                Debug.Print "Greater than 8"
       Case Else   ' Other values.
                Debug.Print "Not between 1 and 10"
    End Select
End Sub
```

Note: **Debug.print** outputs the result in **the immediate window.** Choose Vb Editor **view** menue, click the **immediate window** button to open the immediate window

3. Loops
You can use loop structure to repeatedly run a section of your program. The commonly used ones are **For…Next, Do while…Loop** and **Do…Loop** statements.

(1) For…Next
If you know how many times you will replicate, you can use **For…Next** statement. Here is a simple example copied from Excel VB Help.

Example 7 **Sum**

```
Sub Sum()
  Total = 0
   For j = 2 To 10 Step 2
     total = total + j
   Next j
```

```
    MsgBox "The total is " & total     'The & operator to force string concatenation. MsgBox is a window's dialog
       box to display a message
End Sub
```

ANALYSIS In this program, j changes from 2 to 10 at a step of 2 (j = 2, 4 , 6, 8, 10). For each j, total =total + j is performed. When Next j is encountered, j increments a step size 2: j = j + 2, and then it repeats the calculation again until j >10. This program does the following calculation: total = 2 + 4 + 6 + 8 + 10. We usually use the recurrence equation**: total $_t$ = total $_{t-1}$ + x** with the initial condition total $_0$ = 0 to perform the summation (Sum = $\sum_{j=1}^{n} x_j$).

Example 8 **For...Next Statement: product**

This program will calculate Prod = 1 x 2 x 3 x … F = F !, e.g., the factorial of F.

```
Sub Factorial()
    Dim I, F As Integer
    Dim Prod As Double

    F = InputBox("Calculate Factorial of :", "***Factorial Calculation***", "2")
    Prod = 1
     For I = 1 To F
        Prod = Prod * I
     Next I
     MsgBox "The factorial of F is " & Prod     'The & operator to force string concatenation.
End Sub
```

Note: Through InputBox, you can enter a single input value for a program. Learn more of InputBox by VB Help

(2) Do…Loop

If you don't know the run times, you can use **Do…Loop,** or **Do While** {Condition}… **Loop**, or **Do** … **Loop Until** {condition}.

Example 9 **Count the fish with weight greater than 20 lb**

```
Sub CountIf()
    Dim i, count As Integer
    Dim w As Double

    i = 2
    count = 0
    Do
       'Read the fish weight
       w = Sheets("Example2-14").Cells(i, 2).Value

       'Count only if w > 20
       If w > 20 Then
          count = count + 1
       End If

       'Change i for another loop
       i = i + 1
    Loop Until w < 0

    Sheets("Example2-14").Cells(2, 4).Value = "# fish with weight > 20 lb:  " + Str(count)
End Sub
```

ANALYSIS We use another way to refer to a cell in the spreadsheet here: Cells(Row #, Col #). It is similar to Range(“ “), but here its row no. and column no. can be changed according to the context whereas Range can only refer to fixed cell(s). Notice that by Do…Loop, we find the weight of each fish from top to bottom until encounter “-1”. The statement I = I + 1 is important for Do…Loop, without it the program will loop forever with I stays at its initial value 2. For…Next statement does this automatically.

Example 10 **Heaviest Chinook!**

```
Sub FindMax_HeavyFish()
  Dim Heavy As Double
  Dim w As Double

  i = 2

  'Read the first fish weight value and let the value of Heavy equals to it
  Heavy = Sheets("Example2-14").Cells(i, 2).Value
  Do
    i = i + 1
    'Read the next fish weight
    w = Sheets("Example2-14").Cells(i, 2).Value

    'If the next value is larger than the value of Heavy then put that value in Heavy variable
    If w > Heavy Then
      Heavy = w
    End If

  Loop Until w < 0

  Sheets("Example2-14").Cells(2, 4).Value = "Here is the heaviest Chinook:  " + Str(Heavy)

End Sub
```

Example 11 **Sorting the chinook in your trip**

This example demonstrates how loops and arrays are used together making your program more neat and clear. Notice that the sorting process is an extension to the Find_Max program above: after we find the heaviest fish, then we look for the next heaviest fish and then next… using the same algorithm as Find_Max.

```
Sub RippleSort()
      Dim i, j, N, NTimes As Integer
      Dim x(1 To 30), xtemp As Double

     ‘Input
      N = 9
      For i = 1 To N
        'Read the fish weight
        x(i) = Sheets("Example2-14").Cells(i + 1, 2).Value
      Next i

     ‘Processing
      NTimes = N – 1

      For i = 1 To NTimes
        For j = i + 1 To N
          If x(j) < x(i) Then
```

```
            'Swap the value of x(j) and x(i)
              xtemp = x(j)
              x(j) = x(i)
              x(i) = xtemp
            End If
          Next j
        Next i

        ‘Output
        Sheets("Example2-14").Cells(1, 7).Value = "Sorted weight"
        For i = 1 To N
          'Output the sorted weight in column 5, row 2 to 10
           Sheets("Example2-14").Cells(i + 1, 7).Value = x(i)
        Next i
End Sub
```

ANALYSIS

4. **Subs (Macros)**
   There are two kinds of programs in VB: (1) Subs and (2) Functions. The difference between subs and functions is that you use the function name to return a value (mimic a math function), but for a sub, you need it does lots of tasks and don’t need it return a particular value.

(1) Subs without arguments—Macro
A Macro is a kind of subs without arguments. You can run it directly through the **Macro** dialog box. All our previous examples are Macros.

(2) Subs with argument(s)
We usually break a complex program into small blocks (subs). Each block will perform only one independent task. We make the blocks as independent as possible. This way, the bugs in each block will not influence other block. Changes made to each block can be done separately. We connect or string the blocks together into an entity (a program or module) by **call** statement and provide the input data need by a block through arguments. You can also call a macro, but it doesn’t need your input data.

Syntax

Sub name [(argumentlist)]
    [statements]
    [Exit Sub]
    [statements]
End Sub

An argument is a constant, variable, array or expression passed to a procedure.

Example 12 **A sub with arguments**

```
 Sub Ybar(N, x, Average)
   'calculate sample mean
   Dim sum As Double
   sum = 0
   For i = 1 To N
     sum = sum + x(i)
   Next i

   Average = sum / N
 End Sub
```

(3) Call a sub within a sub
Using:

    Call sub-name(argumentlist)

Example 13 **Bootstrapping**

```
Sub Bootstrap()
    '***********************************
    '*  Learn How to do Bootstrapping  *
    '***********************************
    'a) Draw a random sample from the observed Chinook catch data with replacement
    'b) Using the Ybar sub to calculate the mean of the sample
    'c) Replicate a and b for 1000 times
    'd) Calcualte the Variance of 1000 Ybar's produced in b and c
    'e) Using Excel "=VAR()" to calculate the estimated variance of the sample mean calculation for the original
        observed weight values:  var^(Ybar)=VAR(B2:B10)/N:
    'f) Comparing the two Variances just obtained

    Dim i, B As Integer
    Dim Index(1 To 30) As Integer
    Dim x(1 To 30), w(1 To 30) As Double
    N = 9
    For i = 1 To N
        'Read the observed fish weight data
        x(i) = Sheets("Example2-14").Cells(i + 1, 2).Value
    Next i

   Sheets("Example2-14").Cells(16, 1).Value = "BootRep"
   Sheets("Example2-14").Cells(16, 2).Value = "YbarBoot"

  'Bootstrapping
  For B = 1 To 1000

    'Draw replicate sample from the observed sample with replacement
    Call sampler(N, Index)

    'find the weight values corresponding to the indexes: Index(i)
    For i = 1 To N
        w(i) = x(Index(i))
    Next

    'call the sub Ybar to calculate the mean
    Call Ybar(N, w, mean)

    Sheets("Example2-14").Cells(B + 16, 1).Value = B
    Sheets("Example2-14").Cells(B + 16, 2).Value = mean
   Next B
End Sub

Sub Ybar(N, x, Average)
    'calculate sample mean
    Dim sum As Double
    sum = 0
    For i = 1 To N
        sum = sum + x(i)
    Next i

    Average = sum / N
```

```
End Sub

Sub sampler(N, Index)
  Dim i As Integer
  For i = 1 To N
   'a) Using RND function to draw a random number between [0, 1):The Rnd function returns a value less than 1
      but greater than or equal to zero.
   'b) and convert this random number to an integer in the range 1 to 9 using equation: Index(i) = Int(N * Rnd) +
      1
   Index(i) = Int(N * Rnd) + 1  'if Rnd=0, then Index = 1, if Rnd near 1, Index = N
  Next i
End Sub
```

5. Functions

(1) The differences between a sub and a function

Syntax

Function name [(arglist)] [As type]
    [statements]
    [……]
    [statements]
    [name = expression]
End Function

Example 14 **A function with arguments**

```
Function YbarB(N, x)
  'calculate sample mean
  Dim sum As Double
  sum = 0
  For i = 1 To N
    sum = sum + x(i)
  Next i

  YbarB = sum / N
End Function
```

(2) Using a function
Just as using a standard function.
For example, in sub Bootstrap(), you can use the function Ybarb() by
```
    Sheets("Example2-14").Cells(B + 16, 3).Value = YbarB(N, w)
```
(3) A function can return multiple values—return an array!

Example 15 **A function return an array**
In worksheet-“Transpose”, use the following function to transpose the matrix there.

```
Function MatrixTranspose(N, A)
  ReDim x(1 To N, 1 To N)

  For i = 1 To N
    For j = 1 To N
      x(j, i) = A(i, j)
    Next j
  Next i
```

```
    MatrixTranspose = x()
End Function
```

6. Modules

A module is a package wrapping up all subs and functions.

- Insert a new module

  You can insert a new module in your project by the insert **Module** button on the **Insert** menu of the VB Editor.

- Import or Export a module

  On the **File** menu of VB, choose **Import** or **Export** to load or save your modules separately in your directories.

Part 3 Learning by yourself--Getting Help and Recording a Macro

(1) Record a macro

1 On the **Tools** menu, point to **Macro**, and then click **Record**.
2 In the Macro **name box**, enter a name for the macro.
3 Click OK.
4 Carry out the actions you want to record.
5 On the Stop Recording toolbar, click Stop Recording

Example 16 Using Record Macro to take down the sorting process of Excel
Try this example in sheet "Example 16"

```
Sub RecordSort()
'
' RecordSort Macro
' Macro recorded 11/23/98 by ftzs
'

'
    Range("B2:B10").Select
    Selection.Copy
    Range("D2").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    Selection.Sort Key1:=Range("D3"), Order1:=xlAscending, Header:=xlGuess, _
        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
End Sub
```

2. Get Help

Position the insertion point anywhere in a keyword, then press F1 to get help. As an example, try to do so for **Range** or **Cells**. Try to find their **formula** property. For example, the following statements sets the formula for cell A2: ActiveSheet.Cells(2, 1).Formula = "=sum(A1:A2)". This is a very useful property. Because Excel has lots of worksheet functions you are very familiar with, this is the way you use the worksheet functions in your VB programs.

Example 17

Here, I want to learn something about "sort" in example 16. So I followed the above instructions. You see I copied some of the content in the help for "sort" to my program GetHelp, and copied some lines from example 16, and put them together, I got a workable sub like this

```
Sub GetHelp()
    'This example sorts the range A1:C20 on Sheet1,
    'using cell A1 as the first sort key and cell B1 as the second sort key.
    'The sort is done in ascending order by row, and there are no headers.
    Range("B2:B10").Select
    Selection.Copy
    Range("D2").Select
    ActiveSheet.Paste

    ActiveSheet.Range("D2:D10").Sort  Key1:=Range("D1")
End Sub
```