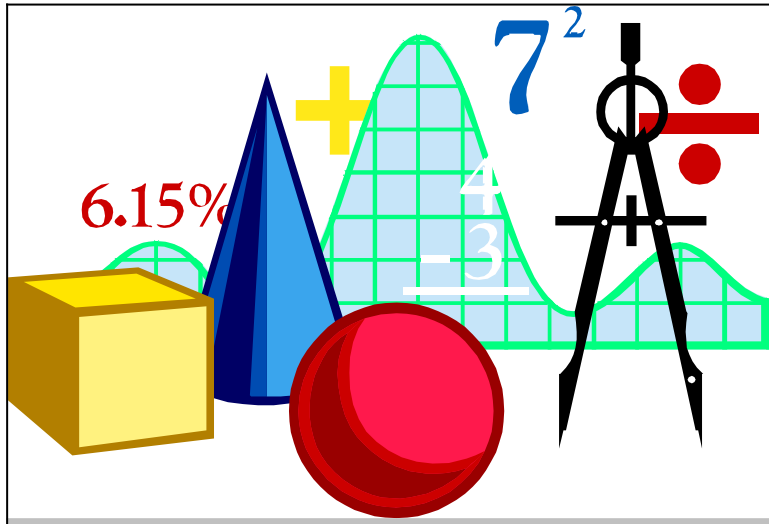


A PRIMER ON MACRO PROGRAMMING WITH Visual BASIC for Applications/Excel



by

Steven C. Chapra

Room 223 (Telephone: 617 727 3654)

e-mail: schapr01@tufts.edu

Civil and Environmental Engineering Department

Tufts University

Medford, MA

October 14, 1999

BIBLIOGRAPHY

- Walkenbach, John, Microsoft Excel 2000 Power Programming With Vba, IDG Books, Foster City, CA, 1999, ISBN: 0-7645-3263-4, \$39.99 + shipping (www.amazon.com).
- Walkenbach, John, Excel 2000 for Windows for Dummies : Quick Reference (--For Dummies), IDG Books, Foster City, CA, 1999, ISBN: 0-7645-0447-9, \$10.39 + shipping (www.amazon.com).

The Basics of Programming in Visual BASIC

Computer programming is not difficult. It really amounts to

- Learning several general concepts
- Being careful, organised and logical
- Practice

In this primer, we will outline the general concepts using the Visual BASIC computer language.

There are only 8 fundamental concepts underlying computer programming to solve numerical problems confronted by engineers. These are

1. The “idea” of a program
2. Constants, variables, assignment and types
3. Mathematics
4. Decisions
5. Loops
6. Arrays
7. Macros, functions and subroutines
8. Input/output

The following sections provide some detail on each of these concepts.

1. THE “IDEA” OF A PROGRAM

A *program* is a set of instructions to tell a computer to do something. The simplest type is a *sequence* of instructions that the computer implements one after another in a mindless fashion. These instructions are sometimes called *statements*. For example, in Visual BASIC

```
Sub Adder ()  
a = 10  
b = 33  
c = a + b  
Msgbox c  
End Sub
```

Although some of the words might seem alien, it's not too difficult to see that the computer will add two numbers together and then display the answer. In this case, because we're using Excel, we employ a “message box” to display the answer on the spreadsheet.

The implication of a program as a series of instructions is that they must be unambiguous and totally logical. If not, the program won't work the way you want it to. For example, suppose you redid the program and switched two of the statements

```
Sub Adder ()  
a = 10  
b = 33  
Msgbox c  
c = a + b  
End Sub
```

This wouldn't give the desired result because it would try to display the answer before it had been calculated.

2. CONSTANTS, VARIABLES, ASSIGNMENT AND TYPES

Constants

There are three types of values (formally called *constants*) that are commonly used by engineers:

- Integers. That is, whole numbers like 86400 or -37.
- Real. That is, numbers with decimal points. These are also called *floating point* numbers in computer jargon. Examples would be 3.14159 and -9.81. Scientific notation is handled as in 1E-5 is equivalent to 1×10^{-5} .
- Character. Alphanumeric information like names and labels such as "Sarah Jones" or "Oct. 1, 1998"

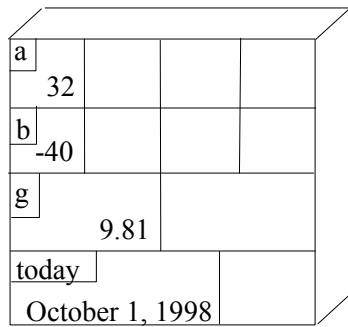
Variables and assignment

These are symbolic names that can be used to store constants. This is often accomplished with an assignment statement like

```
a = -32  
b = 7.325E2
```

These would store a value of -32 in the variable *a*. Therefore, you can say *a* has a value of -32. Similarly, *b* has a value of 732.5 (i.e., 7.325×10^2).

It is useful to understand what the computer does when a line like $a = 32$ is executed. In essence, it sets up a location in its memory which it labels with an *a* and into which it stores the value of 32. The analogy of postal box locations is useful in this regard (Fig. 1).



```
Visual BASIC Program

SUB Assign()
Dim a, b As Integer
Dim g As Single
Dim today As String
a = 32
b = -40
g = 9.81
today = "October 1, 1998"
END SUB
```

FIGURE 1 Analogy between computer memory locations and postal boxes. Notice that different size boxes are used for each variable type.

Question: Which of the two following Visual BASIC statements is incorrect? Why?

$x = x + 1$

$a + b = c + d$

Types

The postal box analogy stresses the fact that different memory locations are required to store the 3 basic types of information. Some languages require that you declare the type of each variable. In VBA, variable “typing” is voluntary, but highly recommended. Because we believe that it’s excellent programming practice to type variables, we can force the issue by placing the following line at the beginning of a module,

```
Option Explicit
```

If we include such a line, we make typing mandatory. In such cases, all local variables (that is, those that do not originate elsewhere) must be “dimensioned” with a DIM statement. For example,

```
Dim a, b As Integer
Dim g As Single
Dim y as Double
Dim today As String
Dim x as Variant
```

Although you can “type” your variables in this fashion, Visual BASIC does not require that you do so.

For the foregoing Dim statements, you could not include the line

```
a = "October 1, 1998"
```


This would result in an error message, because you're telling the computer to put a square peg in a round hole; that is, you're telling it to store a string constant in a real, single precision memory location. If you do this, the program will not execute and you'll get an error message.

3. MATHEMATICS

Mathematics are based on three key concepts: operator priority, left-to-right and parentheses.

Operator Priority

Visual BASIC follows the following priorities:

OPERATOR	PRIORITY
Parentheses ()	Highest
Exponentiation ^	
Negation -	
Multiplication, *, and Division, /	
Addition, +, and Subtraction, -	

Therefore, if you had

```
x = 5 + 7 * 2
```

the computer would first multiply $7 * 2$ to yield 14. It would then add this to 5 to give 19. The result would be assigned to the variable x.

Another example is

```
z = -2 ^ 2
```

The computer will first square the 2 to give 4 (since exponentiation supercedes negation). Then it will negate it to give the final answer: -4.¹

Left to Right

¹ Remember that Excel works differently. That is, negation supercedes exponentiation. Hence $-2 ^ 2$ in Excel would equal 4. (Hint: good test question)

When choosing between two operations with equal priority, the computer implements them left-to-right. For example,

$$d = 10 / 5 * 7$$

It will first divide 10 by 5 to yield 2. Then it will multiply $2 * 7$ to give the final answer 14. Notice how going right-to-left would give an entirely different answer:

$$10 / 5 * 7 \rightarrow 10 / 35 \rightarrow 0.285714285714286$$

Parentheses

The preceding rules can be overridden by using parentheses. That is the computer will evaluate operations in parentheses first and for nested parentheses will evaluate the innermost first. For example, suppose that we want to evaluate

$$y = \frac{4 + 9}{7}$$

If we write it as

$$y = 4 + 9 / 7$$

the computer will first divide 9 by 7 and then add the result to 4. To make it do the addition first, we enclose it in parentheses

$$y = (4 + 9) / 7$$

Intrinsic Functions

Note that some mathematical operations are performed so commonly that built-in or intrinsic functions are available for their implementation. For example, suppose you wanted to evaluate the square root of 2,

$$x = \text{Sqr}(2)$$

or evaluate the absolute value of the discriminant of the quadratic formula

$$d = \text{Abs}(b^2 - 4 * a * c)$$

The most common are listed below

Operation	Function	Example	Result
Square root	Sqr	a = Sqr(144)	12
Absolute value	Abs	b = Abs(-3 * 1.3)	3.9
Integer (returns integer part of number)	Int	c = Int(35.86)	35
Exponential function	Exp	c = Exp(1)	2.7182818
Natural logarithm	Log	b = Log(2.7182818)	0.9999999
Sine	Sin	r = Sin(30 * 3.14159 / 180)	0.4999996

Cosine	Cos	s = Cos(30 * 3.14159 / 180)	0.8660256
Tangent	Tan	t = Tan(45 * 3.14159 / 180)	0.9999867
Arc tangent	Atn	pi = 4 * Atn(1)	3.1415926

Question: Write the following expression in Visual BASIC

$$y = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

4. DECISIONS

There are two primary ways to get the computer to deviate from its line-by-line sequential execution of statements. These are called decisions and loops.

As the name implies, decisions do one set of instructions if a logical statement is true and another set if it is false. The If/Then/ELSE statement is the simplest way to do this. For example,

```
d = b ^ 2 - 4 * a * c
If d > 0 Then
    r1 = (- b + SQR(d)) / (2 * a)
    r2 = (- b - SQR(d)) / (2 * a)
    i1 = 0
    i2 = 0
ELSE
    r1 = - b / (2 * a)
    r2 = - b / (2 * a)
    i1 = SQR(ABS(d)) / (2 * a)
    i2 = - I1
End If
```

5. LOOPS

As the name implies, loops do operations repetitively. There are two types of loops depending on how the repetitions are terminated.

Decision loops

As the name implies, these loops terminate if a decision is true. An example is

```

x = 22
Do
  x = x - 2
  If x <= 5 Then Exit Do
Loop

```

This loop will repeat until x has been reduced to a value that is less than or equal to 5.

Count-controlled Loops

These execute a prespecified number of repetitions

```

x = 1
For i = 1 To 5
  x = x * i
Next i

```

This loop will execute 5 times (from 1 to 5). At the end of it, x will hold a value of 5!

6. ARRAYS

Subscripted variables are commonplace in algebra. For example, two simultaneous algebraic equations can be written as

$$\begin{aligned}
 b_1 &= a_{11}x_1 + a_{12}x_2 \\
 b_2 &= a_{21}x_1 + a_{22}x_2
 \end{aligned}$$

The counterpart in Visual BASIC is called an *array*. Instead of subscripts, parenthesis are used, as in

```

b(1) = a(1, 1) * x(1) + a(1, 2) * x(2)
b(2) = a(2, 1) * x(1) + a(2, 2) * x(2)

```

Note that you are required to let the computer program know that variables are arrays. This is done with the Dim statement,

```
Dim a(10, 10), b(10)
```

This sets aside 100 and 10 memory locations for the variables *a* and *b*, respectively. Note that you can also employ the Dim statement to type the variables at the same time. For example,

```
Dim a(10, 10) as Single, b(10) as Single
```

This is the way we would do it because of Option Explicit.

Loops are often used in conjunction with arrays. For example, here's a little program that uses two nested loops to count from 1 to 10 using steps of 0.1. The values 1 through 10 are stored in the array, t:

```
Sub TimeCount()  
Dim t(20) as Single  
tt = 0  
t(0) = 0  
For i = 1 To 10  
    For j = 1 To 10  
        tt = tt + 0.1  
    Next j  
    t(i) = tt  
Next i  
End Sub
```

7. FUNCTIONS AND SUBROUTINES

Modular programming refers to the way programs are organized into modules designed to perform specific tasks. The modules come in two flavors: subroutines and subroutines.

Subroutine

A subroutine or Sub is a group of code that performs a task. In VBA, we usually invoke it with a button.

Function

A Function is designed to return a single answer to the calling program. For example, a function to add two numbers and return the sum can be written as

```
Function Sum(a, b)  
    Sum = a + b  
End Function
```

Notice how the single answer (the sum) is assigned to the name of the function.

The Function is invoked by its name. Here's an example of how this is done for the Function Sum,

```
Sub AddSub()  
    x = 22  
    y = 30  
    a = Sum(x, y)  
    b = Difference(x, y)  
End Sub
```

```

Function Sum(a, b)
    Sum = a + b
End Function

Function Difference(a, b)
    Sum = a - b
End Function

```

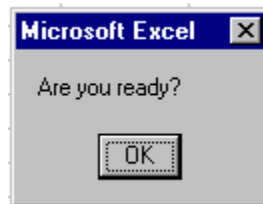
8. INPUT/OUTPUT

The final fundamental concept involved in programming is how information is passed in and out of a program.

Messages. When it is used as the macro language for Excel, the worksheet serves as the input vehicle for Visual Basic. The most fundamental operation is the message box. In its simplest form it merely displays a box containing information on the screen as in

```
Msgbox "Are you ready?"
```

would result in something like



being displayed on the worksheet. If the user selects OK, the program would go back to the next line and continue executing.

Input/Output. Some other fundamental statements that allow you to pass information to and from a worksheet are

To select to a specific worksheet:

```
Sheets("Sheet1").Select
```

To select a specific cell

```
Range("b3").Select
```

To assign the value in the selected cell and assign it to a variable

```
Vol = ActiveCell.Value
```

To print a variable from your program into the selected spreadsheet cell:

```
ActiveCell.Formula = c
```

Another useful operation is to enter tabular information. For example, the user might enter a table of times, flows and inflow concentrations:

	A	B	C	D	E
1	Forcing Functions:				
2	t(yrs)	Q(m3/yr)	pin(mg/m3)		
3	1930	1.25E+09	32.00		
4	1940	1.25E+09	32.00		
5	1941	1.25E+09	44.00		
6	1949	1.25E+09	44.00		
7	1950	1.65E+09	51.52		
8	1951	1.25E+09	64.80		
9	1956	1.25E+09	64.80		
10	1957	9.87E+08	94.43		
11	1958	1.25E+09	83.44		
12	1959	1.25E+09	92.24		
13	1960	1.25E+09	101.12		
14	1961	1.25E+09	109.92		
15	1962	9.18E+08	161.76		
16	1963	1.01E+09	154.95		
17	1964	1.50E+09	136.41		
18					
19					
20					

Now it should be noted that this list might not be a fixed length. For example, at a later date, the user might want to add additional years of data and run the program again. Therefore, you have to develop a means to determine the number of data every time the program is run. This can be done by the following code

```
'check length of forcing function
Range("a2").Select
Selection.End(xlDown).Select
bottom = ActiveCell.Row
nloads = bottom - 2
If nloads > 1000 Then nloads = 0

'input forcing functions
Range("a3").Select
For i = 1 To nloads
    t(i) = ActiveCell.Value
    ActiveCell.Offset(0, 1).Select
    Q(i) = ActiveCell.Value
    ActiveCell.Offset(0, 1).Select
    pin(i) = ActiveCell.Value
    ActiveCell.Offset(1, -2).Select
Next i
```

This code first goes to cell b2. Then the line

```
Selection.End(xlDown).Select
```

moves from a2 down to the last consecutive entry below it (in cell b17). The statement

```
bottom = ActiveCell.Row
```

enters the row number of the active cell into the variable bottom. Therefore, bottom = 17. We can then perform the calculation

```
nloads = bottom - 2
```

to determine that the number of data are equal to $17 - 2 = 15$. This information can then be subsequently used to input the results with a FOR/NEXT loop.

Notice how the offset is used to shift over and down in order to make our way through the values in the table and assign them to subscripted arrays.

The same sort of operation can be used for output. However, when outputting, you will know the amount of information you are displaying. For example, your program may have generated $np = 16$ data records consisting of time, inflow concentration and lake concentration as in

	A	B	C	D	E
1		<i>Output:</i>			
2	<i>t(yrs)</i>	<i>cin(mg/m3)</i>	<i>c(mg/m3)</i>		
3	1930	32	16		
4	1931	32	15.26084446		
5	1932	32	14.9826725		
6	1933	32	14.87798596		
7	1934	32	14.83858849		
8	1935	32	14.82376174		
9	1936	32	14.81818188		
10	1937	32	14.81608197		
11	1938	32	14.81529169		
12	1939	32	14.81499428		
13	1940	32	14.81488235		
14	1941	44	16.64896156		
15	1942	44	18.96986465		
16	1943	44	19.84330754		
17	1944	44	20.17201686		
18	1945	44	20.2957225		
19					
20					

The code to output this data is

```
Sheets("output").Select
Range("a3:c202").ClearContents
Range("a3").Select
For i = 1 To np
    ActiveCell.Formula = tout(i)
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Formula = pinp(i)
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Formula = pout(i)
    ActiveCell.Offset(1, -2).Select
Next i
Range("a3").Select
```