20. VB Programming Fundamentals – Variables and Procedures

20.1 Variables and Constants

VB, like other programming languages, uses variables for storing values. Variables have a name and a data type. Array can be used to store indexed collections of related variables.

Declaring Variables

You declare a variable with the Dim statement, supplying a name for the variable:

Dim variablename [as data type]

A variable name:
- Must begin with a letter.
- Can't contain an embedded period or embedded type-declaration character.
- Must not exceed 255 characters.
- Must be unique within the same scope, which is the range from which the variable can be referenced.

The Scope of Variables

The scope is the lifetime of a variable. Depending on how it is declared, a variable is scoped as either a procedure-level (local) or module-level variable (Table 1).

Table 1. Scope of variables.

Scope	Private	Public
Procedure-level	Variables are private to the procedure in which they appear.	Not applicable. You cannot declare public variables within a procedure.
Module-level	Variables are private to the module in which they appear.	Variables are available to all modules.

Implicit vs. Explicit Declaration

You don't have to declare a variable before using it, which refers to as Implicit declaration. While it is convenient, it can lead to subtle errors in your code if you misspelled a variable name.

You can avoid the problem of misnaming variables by using Explicit declaration. To explicitly declare variables,
- Simply place this statement "Option Explicit" in the Declarations section of a class, form or standard module, or

- From the Tools menu, choose Options…, click the Editor tab and check the Require Variable Declaration.

Private vs. Public

Static vs. Dim

Constants also store values, but as the name implies, those values remain constant through the execution of an application. Using constants can make your code more readable by providing meaningful names instead of numbers.

[Public|Private] Const constantname[As Type] = expression

Public Const conPaxPlanets As Integer = 9

20.2 Data Types

Data types control the internal storage of data in VB. By default, VB uses the Variant data type. There are a number of other available data types in Visual Basic.

Numeric data types
- Integer
- Single
- Double
- Currency

String data type

```
Private S As String          'Variable-length string
Private S As String * 10     'Fixed-length string
```

If you assign a string of fewer than 10 characters, S is padded with enough trailing spaces to total 10 characters. If you assign a string that too long for the fixed-length string, VB simply truncates the characters.

Boolean data type

```
Dim blFlag As Boolean
```

Object data type

```
Dim objDB As Object
Set objDB = OpenDatabase("C:\LogInventory.mdb")
```

20.3 Arrays

If you have experience of programming with other language such as C/C++, I am sure you are familiar with the concepts of arrays. Arrays allow you to refer to a series of variables by the same name and to use a number (an index) to tell them apart. This help you to create smaller and simpler code in many situations, because you can set up loops that deal efficiently with any number of cases by using the index number. We should always try to avoid declaring an array larger than necessary.

In VB, there are two types of arrays:

- A fixed-size array which always remains the same size, and
- Dynamic array whose size can change at run-time.

Declaring Arrays

```
Dim arrayValue(20) as Single

Dim arrayValue() as Single
Redim arrayValue(ComputedSize)
```

Multidimensional Arrays

With VB, you can declare arrays of multiple dimensions. For example, the following statement declares a two-dimensional 10-by-10 array within a procedure:

```
Dim StandStock(9, 9) as Single
```

Or

```
Dim StandStock(1 to 10, 1 to 10) as Single
```

20.4 Procedures

You can simply programming tasks by breaking programs into smaller logical components. These components, called procedures, can then become building blocks that let you enhance and extend VB. Procedures are useful for condensing repeated or shared tasks, such as frequently used calculations, text and control manipulation, and database operations.

There are two major benefits of programming with procedures:

- debug more easily
- reuse the procedures for other program with little or no modification

Three basic types of procedures used in VB:

- Sub procedures do not return a value.
- Function procedures return a value.
- Property procedures can return and assign values, and set references to objects.

Sub Procedures

A Sub procedure is a block of code that is extended in response to an event. There are two types of Sub procedures – General and Event procedures. The syntax for a Sub procedure is:

```
[Private][Public][Static]Sub procedurename (args)

    statements

End Sub
```

Event Procedures

When an object in Visual Basic recognizes that an event has occurred, it automatically invokes the event procedure using the name corresponding to the event. Because the name establishes an association between the object and the code, event procedures are said to be attached to forms and controls.

An event procedure for a control combines the control's name, and underscore "_" and the event name. For example, if you are using a command button to invoke actions by clicking the button, this event procedure will be:

```
Private Sub command1_click()

…

End Sub
```

An event procedure for a form combines form name, underscore, and event. For example, if you are using form1, you can have the following event procedures for this form:

```
Private Sub Form1_click()
    …
End Sub

Private Sub Form1_Load()
    …
End Sub
```

General Procedures

A general procedure tells the application how to perform a specific task. Once a general procedure is defined, it must be specifically invoked by the application. By

contrast, an event procedure remains idle until called upon to respond to events caused by the user or triggered by the system.

Why use general procedures?

- Several different event procedures might need the same actions performed.
- This will eliminate the need to duplicate the code.
- This is the way of modular programming.

Function Procedures

VB includes built-in functions, like Sqr, Cos or Sin. In addition, we can build our own functions. The syntax for a function procedure is:

```
[Private][Public][Static]Function functionname (args) [As type]

    statements

End Function
```

Like a Sub procedure, a function is a separate procedure and can perform a series of statements. However, there are two basic differences between a Sub procedure and a function:

- a function can return a value. While calling a function, we need to use the expression such as returnvalue = functionname(args).
- A function have data type.

Example

Here is an example to show you how to use procedures. Suppose we would like to calculate the basal area of a tree and display it in a list box.

(1) Start a new VB project.
(2) Create the interface

We need to add the following controls on form1 (Figure 1):

- two labels
- a text box
- a list box
- two command buttons

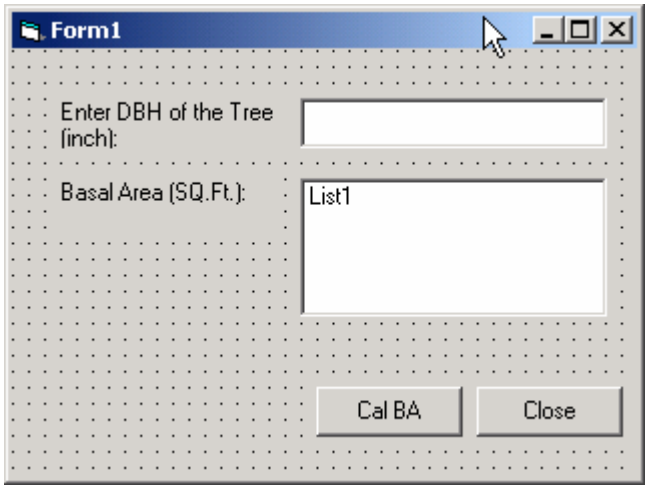

Figure 1. Interface of your application.

(3) Code the procedures

It might be a good way of programming if we use a function to calculate the basal area and a sub procedure to display the basal area in the list box.

```
'In General Declaration
Dim DBH As Single
Dim BA As Single

'A Function for calculating basal area
Private Function CalBA(d As Single) As Single

   CalBA = 0.005454154 * d * d

End Function

'A Procedure for displaying basal area
Private Sub DisplayBA(basal_area As String)

   List1.AddItem basal_area

End Sub

'Event procedure of clicking command1
Private Sub Command1_Click()

   DBH = CSng(Text1.Text)
   BA = CalBA(DBH)
   Call DisplayBA(CStr(BA))

End Sub
```

```
'Event procedure of clicking command2
Private Sub Command2_Click()

   End

End Sub
```

You should notice that two VB built-in functions we used in the program:

CSng – convert single
CStr – convert to string

(4) Run the Project

- Enter a dbh in the text box
- Click "Cal BA" button
- Repeat the above procedures

Passing by value vs. passing by reference

Only a copy of a variable is passed when an argument is passed by value. If the procedure changes the value, the change affects only the copy and not the variable itself. Use the ByVal keyword to indicate an argument passed by value.

Passing arguments by reference gives the procedure access to the actual variable content in its memory address location. As a result, the variable's value can be changed permanently. Passing by reference is the default in VB.

20.5 Control Structures

Control structures allow you to control the flow of your program's execution. In this lecture, we are going to learn Decision Structures and Loop Structures.

Decision Structures

VB procedures can test conditions and then, depending on the results of the test, perform different operations. The decision structures that VB supports include:

(1) If … Then

```
If condition Then statement

If condition Then

        Statements

End If
```

(2) If … Then … Else

```
If condition1 Then
        Statement block 1
ElseIf condition2 Then
        Statement block 2
…
Else
        Statement block n
End If
```

(3) Select Case

```
Select Case testexpression
        Case expressionlist1
                Statementblock1
        Case expressionlist2
                Statementblock2

        …

        Case Else
                Statementblockn
End Select
```

Loop Structure

Loop structures allow you to execute one or more lines of code repetitively. The loop structures that VB supports include:

(1) While … Wend

```
While conditions
    Statements
Wend
```

(2) For … Next

```
For counter = start To end [Step increment]
```

```
    Statements
Next counter
```

(3) For Each … Next

```
For Each element In group
    Statements
Next element
```

Class Exercise (Homework 3):

We have data of 10 trees in Table 2. Create a simple VB project to calculate the total basal area and volume of these 10 trees. You should implement the procedures to calculate and sum basal area and volume respectively. Declare three arrays to hold tree data and initialize them in the General Declaration of your project. You are also required to use one command button to invoke either basal area or volume calculation.

Table 2. Tree data.

Tree	DBH (inches)	Merchantable height (Logs)
1	27	1
2	13	2
3	12	1
4	15	2.5
5	17	2
6	25	2
7	28	0.5
8	10	1
9	29	2.5
10	13	0.5

Basal area (BA) in ft^2:

$$BA = 0.005454154*(DBH)^2$$

Volume (V) in Doyle board foot:

```
V =   ((0.55743 * L ^ 2 + 41.51275 * L - 29.37337) +
      (2.78043 - 0.04516 * L ^ 2 - 8.77272 * L) * d +
      (0.04177 - 0.01578 * L ^ 2 + 0.59042 * L) * d ^ 2)
```

where L = number of logs;
d = DBH in inches;