Visual Basic - Variables

Introduction

Having discussed the procedure declaration statement we now need to discuss the statements contained within the procedure. We deal first with variable, constants and data type.

As with procedures we need to define the scope of the data. We can also specify the length of time to keep data in a variable or constant – its *lifetime*. This can prevent unexpected re-initialisation of variables and allow retention of a variable's value between calls to a procedure. We can also specify the amount of memory allocated to the data. Keeping this to the minimum for the task obviously uses less memory but also makes the macros run faster.

Variables

Variables are named areas in memory in which you store data while the workbook containing the code is open.

The rules for naming variables are the same as those for naming procedures. To minimise the time you spend understanding and debugging code use

- each variable for one purpose only,
- names that indicate what the variable represent in the problem.

By default VB allows you to create variables simply by using them in your code. (Other languages, e.g. C require you to explicitly declare variables before using them). Hence the statement

```
tempVal = Worksheets(1).Range("A1").Value
```

assigns the value in cell A1 to the variable tempVal. This is an *implicit* variable declaration. Used in this way VB creates a variable of **Variant** data type (see later for the types). [**Variant** data types can contain any data, including strings, numbers, dates and objects].

Although it's convenient to let VB create variables whenever you need them it is a DANGEROUS practice and is to be discouraged. It can lead to subtle errors which are hard to track down. For example

```
tempVal = Worksheets(1).Range("A1").Value
Worksheets(1).Range("A1").Value = ______
Worksheets(1).Range("A2").Value
```

```
Worksheets(1).Range("A2").Value = temVal
```

was written with the intention that it swops the contents of cells A1 and A2 – but it does not. After it's run cell A1 contains the value previously in cell A2, and cell A2 is empty! Do you see the error?

Explicit declaration of variables is a safer technique and makes the code more reliable. To explicitly declare a variable use the **Dim**, **Private**, **Public**, or **Static** keyword to specify the variable's scope, lifetime and data type.

Explicit variable declaration helps prevent you from inadvertently

creating a new variable by misspelling an existing one,

• re-using an existing one with a different meaning in the problem, when a new one should be created, so overwriting the value stored in it.

It also speeds execution of the code as VB doesn't have to determine the data type of each variable while the code is running.

The simplest form of explicit variable declaration creates a new variable of **Variant** type with procedure-level scope, the value in the variable only being preserved while the procedure is running, e.g.

```
Sub AreaOfCircle()
    Dim radius, area
    Radius = InputBox("Circle radius is ")
    Area = 3.14159 * radius ^ 2
    MsgBox "the area of the circle is " & area
End Sub
```

You can specify that VB generates an error message whenever it finds a name used as a variable that has not previously been declared explicitly as a variable. At the top of the module put Option Explicit. This forces variables to be declared before they are used. It is recommended that you use the Option Explicit statement in all your modules.

Variable scope and lifetime

When you declare a variable in a procedure, only code within that procedure can access or change the value of the variable. The scope is *local* to the procedure. Sometimes you need a variable available to all procedures within the same module or even to all modules in the same workbook. The scope attached to variable declarations is

Declaration	Scope
Dim or Static within a procedure	Procedure only
Dim or Private at top of module	Private, available to all procedures in module
Public at top of module	Public, available to all procedures in all
	modules in woorkbook

It's a good idea to use the narrowest scope for your variables as it helps reduce errors.

A variable's *lifetime* is the time for which VB preserves the value in the variable. The values in private and public variables are preserved while the workbook is open. However if you edit a module all modules in a workbook are recompiled and all variables are reset to their initial values. Although variables declared by **Dim** or **Static** have the same scope they have different lifetimes. Variables declared with **Static** exit the whole time the workbook is open and their values are retained between calls to the procedure. Variables declared by **Dim** exist ONLY while the procedure is running and their values are not preserved nor is the memory allocated to them. The next time the procedure is run the local variables are re-initialised. Thus to preserve the value of a local variable you must declare it with the **Static** keyword. Consider the following example,

Function RunningTotal(num)

```
Static accumulation
accumulation = accumulation + num
RunningTotal = accumulation
```

End Function

If accumulation were declared with **Dim** the previous value would not be preserved and the function would simply return the value with which it was called!

Private variable are available to all procedures in a module where they are declared. Public variables are available to every procedure in any module in the workbook where they are declared. A public variable retains its value from the time a value is assigned to it until the workbook is closed (or a procedure is edited).

Declaring a variable's data type

When a variable is declared you can specify a data type for it. This type defines the kind of data that can be stored in the variable. If you don't specify a data type VB automatically give the variable the **Variant** data type. This is the most general type as it can handle all types of data and automatically converts between them. You might be tempted, therefore, to always use this type. However, if, for example, a variable will only contain a small integer value you can significantly speed up the macro when performing arithmetic operations if you declare the variable as an **Integer**.

Data types

Each type takes up a certain amount of memory, allows a specific range of values and is best suited for particular kinds of data.

Type name	Storage size	Range
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,647
Single	4 bytes	-3.402823E38 to -1.401298E-45
_		(negative)
		1.4012298E-45 to 3.402823E38
		(positive)
Double	8 bytes	-1.79769313486232E308 to
		-4.94065645841247E-324 (negative)
		4.94065645841247E-324 to
		1.79769313486232E308 (positive)
Currency	8 bytes	-9223372033685477.5808 to
		922337203685477.5807
String	1 byte per character	0 to approx 65,500 characters
Boolean	2 bytes	True or False
Date	8 bytes	January 1, 100 to December 31 9999
Object	4 bytes	Any Object reference
Variant	16 bytes + 1 byte for each	Null, Error, any numeric value valid
	character	for any numeric data type, any text,
		object or array

Decimal numbers can be expressed in their usual form, 3.14159 or as *mmmEeee* (or *mmmDeee*) where *mmm* is the *mantissa* and *eee* is the *exponent*, e.g. 3.4024E-34 with *mmm* = 3.4024 and *eee* = -34.

Comments on the data types

Integer

If a variable will always contain integer numbers declare it as **Integer** or **Long**. If it may contain fractional numbers declare it as **Single**, **Double** or **Currency**. Beware. A variable (or constant) of any numerical type can contain any numerical data. Hence assigning a value of 3.7 to a variable of **Integer** data type does NOT produce a computing error (even though 3.7 is not an integer) but will produce a mathematical error as VB rounds off (it does NOT truncate) the fractional part before assigning it to the **Integer** Thus the value becomes 4 in this example.

A variable of the numeric data types is initialised to zero (0) after it's declared and before you assign a value to it.

String

This data type will always contain a character string and not a numeric value. It's initial value is "" (the empty string). When assigning a value use double quotation marks e.g.

Boolean

These contain a logical value i.e. **True** or **False**. The initial value is **False**.

Date

These contain date and time values, e.g. $someDate = #3-6-99 \ 13:21#$. The initial value is 12:00:00 on December 30 1899.

Object

These contain a reference to an object within Excel. You assign an object variable with the **Set** statement, e.g.

Dim wks As Object

Set wks = Worksheets(1) Object variables are initialised to the value **Nothing**.

Variant

This type allows the variable to hold most types of data. The initial value is **Empty**. VB automatically performs the conversions e.g.

```
Private varValue ` Variant type by default
varValue = "17" ` contains two character string 17
varValue = varValue - 9 ` now contains numeric value 8
varValue = "VP" & varValue ` now contains the three
` characters VP8
```

Constants

You may need to use quantities that have constant values (e.g. p). A *constant* is a name that takes the place of the number or string that doesn't change. It's like a variable but you cannot modify it. There are two types of constants,

- 1. Built-in (or intrinsic) provided by the application. All Excel built-in constants start with xl in the name.
- 2. Symbolic or user-defined.

You declare a constant using the **Const** keyword. You can specify its scope (private by default, or public), e.g.

```
Const constPi = 3.14159
```

```
Const constCodeName As String = "Enigma"
```

Const constTwoPi = constPi * 2