EXCEL VBA MADE EASY

Disclaimer

Excel VBA Made Easy - A Complete Tutorial for Beginners is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microsoft Corporation.

Trademarks

Microsoft, Visual Basic, Excel and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Liability

The purpose of this book is to provide basic guideline for people interested in Excel VBA programming. Although every effort and care has been taken to make the information as accurate as possible, the author shall not be liable for any error, harm or damage arising from using the instructions given in this book.

Acknowledgement

I would like to express my sincere gratitude to many people who have made their contributions in one way or another to the successful publication of this book.
My special thanks go to my children Xiang, Yi and Xun. My daughter Xiang edited this book while my sons Yi and Xun contributed their ideas and even wrote some of the sample programs for this book. I would also like to appreciate the support provided by my beloved wife Kim Huang and my youngest daughter Yuan. I would also like to thank the million of visitors to my **Excel VBA Tutorial** website at **http://excelvbatutor.com**/ for their support and encouragement.

About the Author

Dr. Liew Voon Kiong holds a bachelor degree in Mathematics, a master degree in management and a doctoral degree in business administration .He has been involved in programming for more than 15 years. He created the popular online Visual Basic Tutorial at **www.vbtutor.net** in 1996 and since then the web site has attracted millions of visitors and it is one of the top searched **Visual Basic** websites in many search engines including Google. In order to provide more support for Excel VBA hobbyists, he has written this book based on his online **Excel VBA tutorial** at **http://excelvbatutor.com/.** He is also the author of **Visual Basic 6 Made Easy, Visual Basic 2008 Made Easy** and **Visual Basic 2010 Made Easy.**

TABLE OF CONTENTS

Chapter 1

Introduction to Excel VBA

- ❖ Getting to know the concept of Excel VBA
- ❖ Learn how to use Visual Basic Editor in MS Excel
- ❖ Learn how to write simple Excel VBA code

1.1 The Concept of Excel VBA

VBA is the acronym for Visual Basic for Applications. It is an integration of the Microsoft's event-driven programming language Visual Basic with Microsoft Office applications such as Microsoft Excel, Microsoft Word, Microsoft PowerPoint and more. By running Visual Basic IDE within the Microsoft Office applications, we can build customized solutions and programs to enhance the capabilities of those applications.

Among the Visual Basic for applications, Microsoft Excel VBA is the most popular. There are many reasons why we should learn VBA for Microsoft Excel, among them is you can learn the fundamentals of Visual Basic programming within the MS Excel environment, without having to purchase a copy of Microsoft Visual Basic software. Another reason is by learning Excel VBA; you can build custom made functions to complement the built-in formulae and functions of Microsoft Excel. Although MS Excel has a lot of built-in formulae and functions, it is still not enough for certain complex calculations and applications. For example, it is very hard to calculate monthly payment for a loan taken using Excel's built-in formulas, but it is relatively easy to program a VBA for such calculation. This book is written in such a way that you can learn VBA for MS Excel in an easy manner, and everyone shall master it in a short time!

You can program Excel VBA in every version of Microsoft Office, including MS Office 97, MS Office2000, MS Office2002, MS Office2003, MS Office XP and MS Office 2007. The reason VBA is needed is due to the limitations in using the built-in functions of MS Excel and macro recording. By using VBA, you can build some very powerful tools in MS Excel, including financial and scientific applications such as getting financial data from the Internet as well as linear programming.

1.2 The Visual Basic Editor in MS Excel

There are two ways which you can start programming VBA in MS Excel. The first way is to place a command button on the spreadsheet and start programming by clicking the command button to launch the Visual Basic Editor. The second way is to launch the Visual Basic Editor by clicking on the Tools menu then select Macro from the drop-down menu and choose Visual Basic Editor. Lets start with the command button first. In order to place a command button on the MS Excel spreadsheet, you need to click View on the MS Excel menu bar and then click on toolbars and finally select the Control Toolbox after which the control toolbox bar will appear, as shown in Figure 1.1. Then you click on the command button and draw it on the spreadsheet, as shown in Figure 1.2.

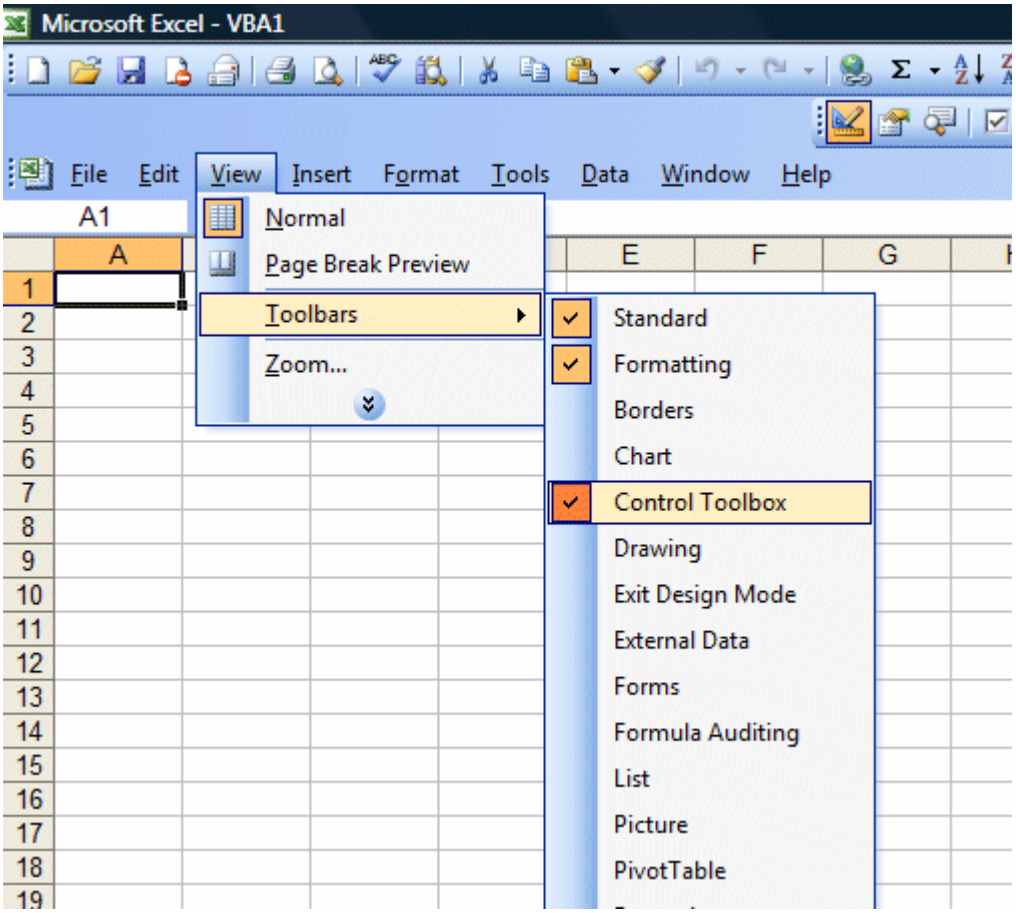

Figure 1.1: Displaying Control Toolbox in MS Excel.

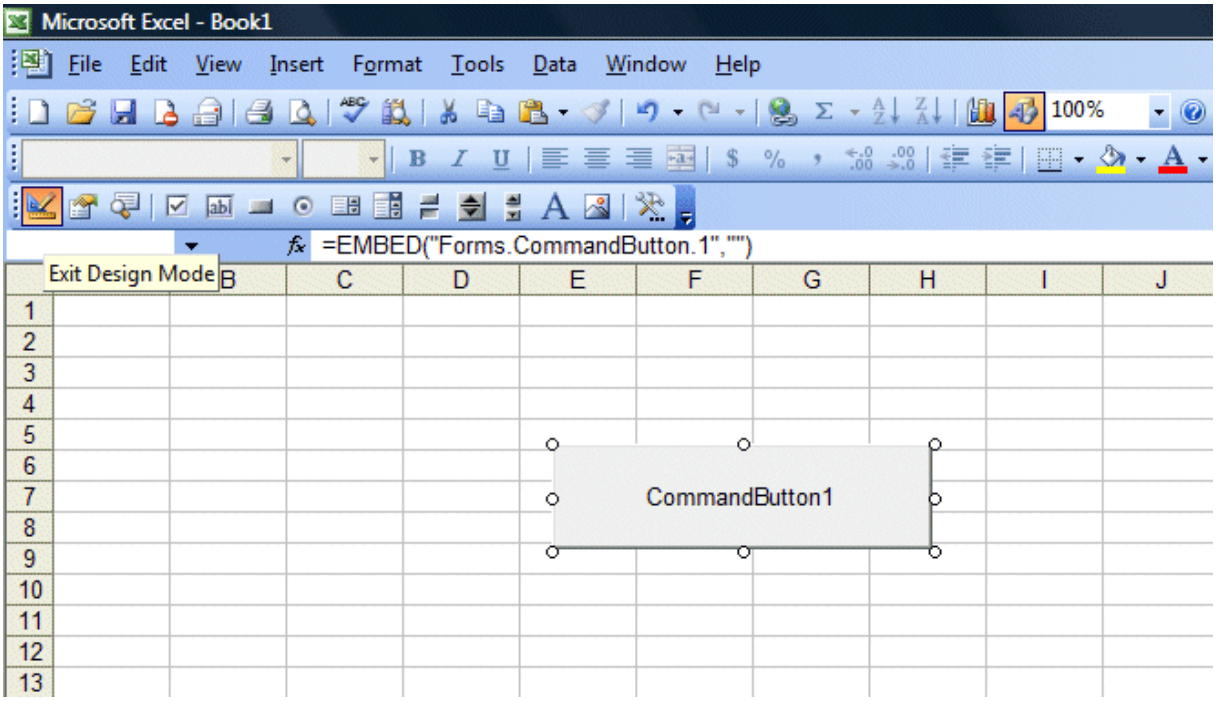

Figure 1.2: The Command Button in Design Mode

Now you select the command button and make sure the design button on the far left of the control toolbox is depressed. Next, click on the command button to launch the Visual Basic Editor. Enter the statements as shown in figure 1.3. Let's write out the code here:

Example 1.1

```
Private Sub CommandButton1_Click ()

    Range ("A1:A10).Value="Visual Basic "

    Range ("C11").Value=Range ("A11").Value +Range ("B11").Value

End Sub
```

The first statement will fill up cell A1 to cell A10 with the phrase "Visual Basic" while the second statement add the values in cell A11 and cell B11 and then show the sum in cell C11. To run the program, you need to exit the Visual Basic

Editor by click the Excel button on the far left corner of the tool bar. When you are in the MS Excel environment, you can exit the design mode by clicking the design button, then click on the command button.

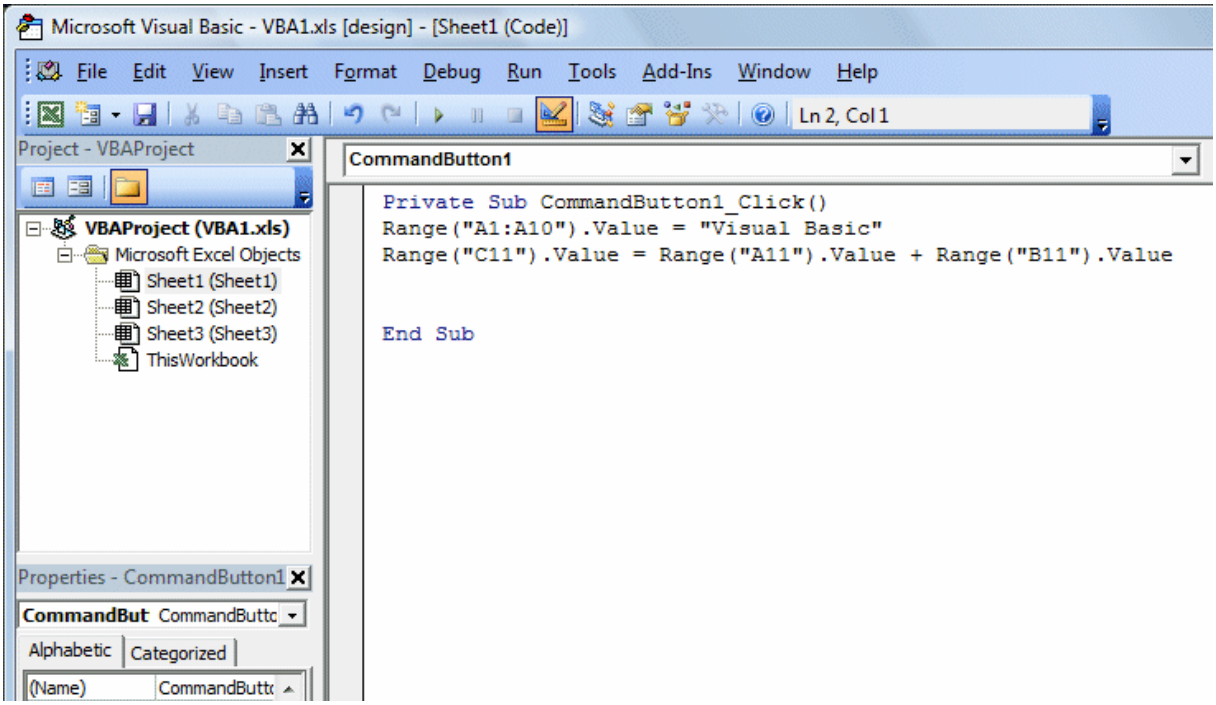

Figure 1.3: The Visual Basic Editor IDE in MS Excel

Running the above VBA will give you the following output.

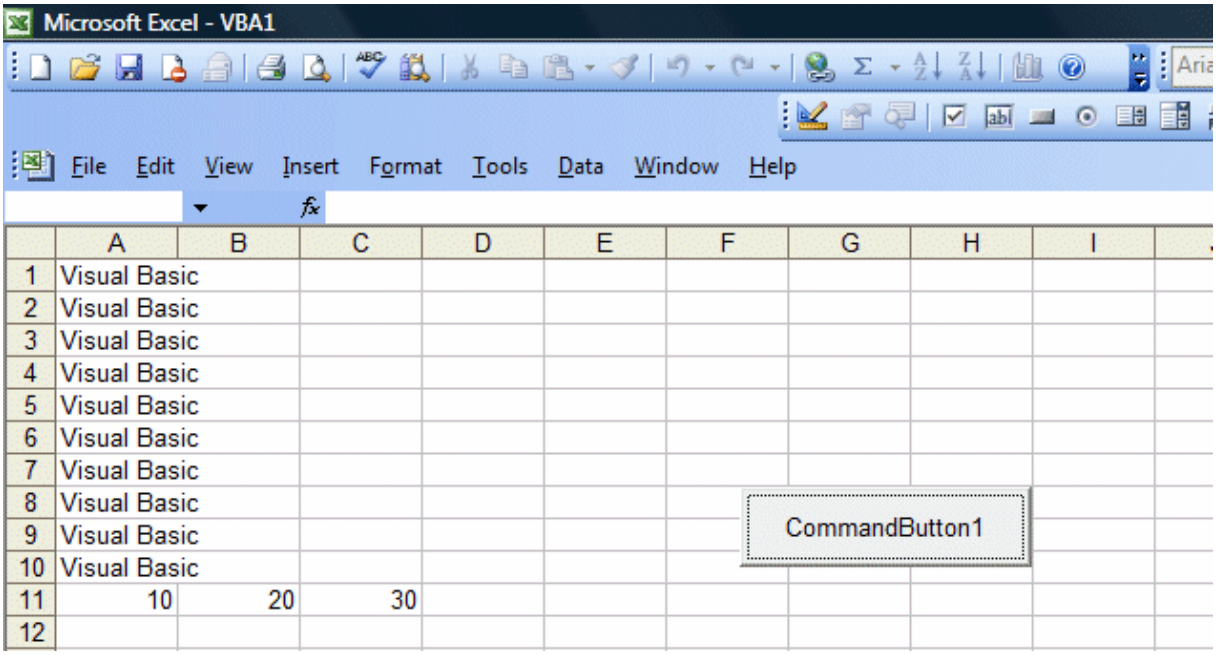

Figure 1.4:

1.3 The Excel VBA Code

Writing Excel VBA code is almost exactly the same as writing code in Visual Basic, which means you have to use syntax similar to Visual Basic. However, there are codes specially designed for use in MS Excel, such as the use of the object or function called **Range**. It is the function that specifies the value of a cell or a range of cells in MS Excel spreadsheet. The format of using Range is as follows:

Range ("cell Name").Value=K or Range ("Range of Cells").Value=K

Where Value is the property of Range and k can be a numeric value or a string

Example 1.2

Private Sub CommandButton1_Click ()

```
    Range ("A1").Value= "VBA"

End Sub
```

The above example will enter the text "VBA" into cell A1 of the MS Excel spreadsheet when the user presses the command button. You can also use Range without the Value property, as shown in Example 1.3:

Example 1.3

```
Private Sub CommandButton1_Click ()

    Range ("A1") = 100

End Sub
```

In the above example, clicking the command button with enter the value of 100 into cell A1 of the MS Excel spreadsheet.

The follow example demonstrates how to input values into a range of cells:

Example 1.4

```
'Input the value of 100 from cell A1 to cell A10

Private Sub CommandButton1_Click ()

    Range ("A1:A10") = 100

End Sub
```

Chapter 2

Managing Data in Excel VBA

- ❖ Getting to know various data types in Excel VBA
- ❖ Learn how to declare variables and assigning values to them
- ❖ Getting to know various arithmetic operators in Excel VBA
- ❖ Learn how to write code that perform arithmetic operations in Excel VBA

In our daily life, we come across many types of data. For example, we need to handle data such as names, addresses, money, dates, stock quotes, statistics and more everyday. Similarly, in Excel VBA, we have to deal with all sorts of data; some are numeric in nature while some are in the form of text or other forms. Excel VBA divides data into different types so that it is easier to manage when we need to write the code involving those data.

2.1 Data Types in Excel VBA

Excel VBA classifies the information mentioned above into two major data types; namely the numeric data types and the non-numeric data types.

2.1.1 Numeric Data Types

Numeric data types are types of data that consist of numbers, which you can compute them mathematically with various standard operators such as, add, minus, multiply, divide and so on. Examples of numeric data types are your examination marks, your height and your weight, the number of students in a class, share values, price of goods, monthly bills, fees and more. In Excel VBA, we divide numeric data into seven types, depending on the range of values they can store. Calculations that only involve round figures or data that do not need precision can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types, we also call them floating-point numbers. For currency calculation, you can use the currency data types. Lastly, if even more precision is requires which involve many decimal points, we can use the decimal data types. We summarized the data types in Table 2.1

Table 2.1: Numeric Data Types

Type	Storage	Range of Values

Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

2.1.2 Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .We summarized them in Table 2.2

Table 2.2: Nonnumeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False

Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

2.1.3 Suffixes for Literals

Literals are values that you assign to a data. In some cases, we need to add a suffix behind a literal so that VB2010 can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data. Some of the suffixes are displayed in Table 2.3.

Table 2.3: Suffixes for Literals

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

memberName="Turban, John."
TelNumber="1800-900-888-777"
LastDay=#31-Dec-00#
ExpTime=#12:00 am#

2.1.4 User-Defined Types

Besides the above data types, you too can create your own data type. You define a new data type by combining the built-in data types mention above. The format to define a data type is

Type variable_name

Statements

End Type

You need to define the user-defined type in the general declaration area of a module. Let look at Example 2.1.

Example 2.1

```
Type Student
  StuName As String
  StuID As String
  StuFee As Currency
  StuYear As String
End Type
```

Now you have created a new data type called Student which carries information such as name, student ID, fee and Year.

To make use of the above data, you can use the Dim keyword and the format is as follows:

```
Dim group As Student
group.StuName = "James"
group.StuID = "1007"
group.StuYear = "Final Year"
group.StuFee = "$20,000"
Cells(2, 2) = group.StuName
Cells(3, 2) = group.StuID
Cells(4, 2) = group.StuYear
```

```
Cells(5, 2) = group.StuFee
```

To test the above code, open a new workbook and enter the above code in the VBE. Running the VBA will enter the student information into the cells specified in the code, as shown in Figure 2.1

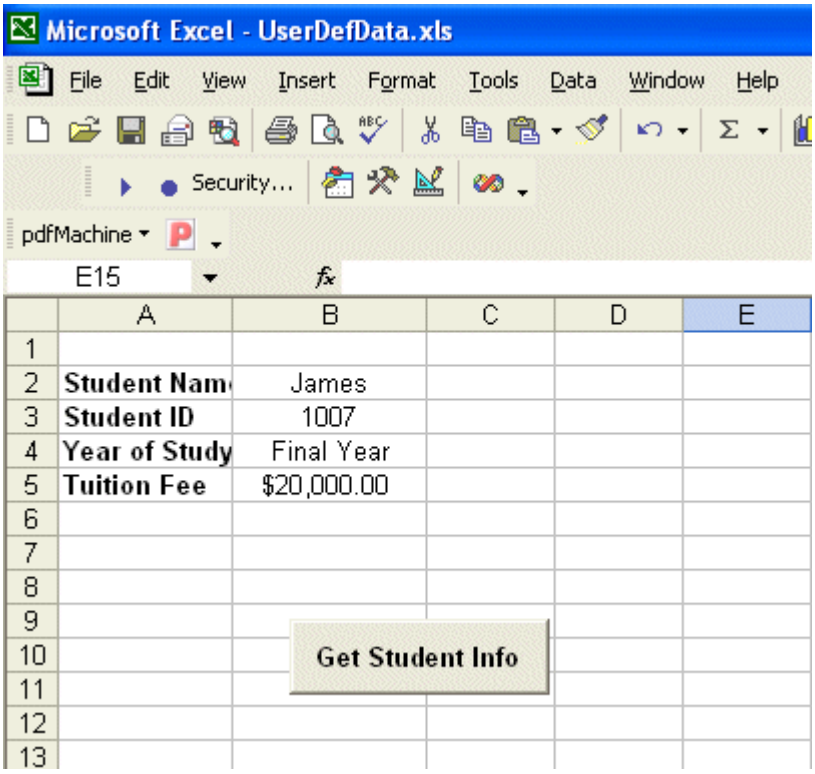

Figure 2.1

2.2 Managing Variables

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In Excel VBA, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Excel VBA, you have to follow a set of rules, as follows:

2.2.1 Variable Names

The following are the rules when naming the variables in Excel VBA

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted

Examples of valid and invalid variable names are displayed in Table 2.4

Table 2.4: Examples of valid and invalid variable names

Valid Name	Invalid Name	
My_Car	My.Car	
ThisYear	1NewBoy	
Long_Name_Can_beUSE	He&HisFather	*& is not acceptable
Group88	Student ID	* Space not allowed

2.2.2 Declaring Variables

In Excel VBA, we need to declare the variables before using them by assigning names and data types. You can declare the variables implicitly or explicitly. For example, sum=text1.text means that the variable sum is declared implicitly and ready to receive the input in textbox1. Other examples of implicit declaration are volume=8 and label="Welcome". On the other hand, for explicit declaration, variables are normally declared in the general section of the code window using the Dim statement.
The format is as follows:

```
Dim variableName as DataType
```

Example 2.2

```
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Dim total As Integer
Dim BirthDay as Date
```

You may also combine them in one line, separating each variable with a comma, as follows:

```
Dim password As String, yourName As String, firstnum As Integer.
```

If the data type is not specified, VBE will automatically declare the variable as a Variant. For string declaration, there are two possible formats, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as Example 2.1 above.
However, for the fixed-length string, you have to use the format as shown below:

```
Dim VariableName as String * n
```

Where n defines the number of characters the string can hold. For example, Dim yourName as String * 10 mean yourName can hold no more than 10 Characters.

Example 2.2

In this example, we declared three types of variables, namely the string, date and currency.

```
Private Sub CommandButton1_Click ()
```

```
    Dim YourName As String
    Dim BirthDay As Date
    Dim Income As Currency
    YourName = "Alex"
    BirthDay = "1 April 1980"
    Income = 1000
    Range("A1") = YourName
    Range ("A2") = BirthDay
    Range ("A3") = Income
End Sub
```

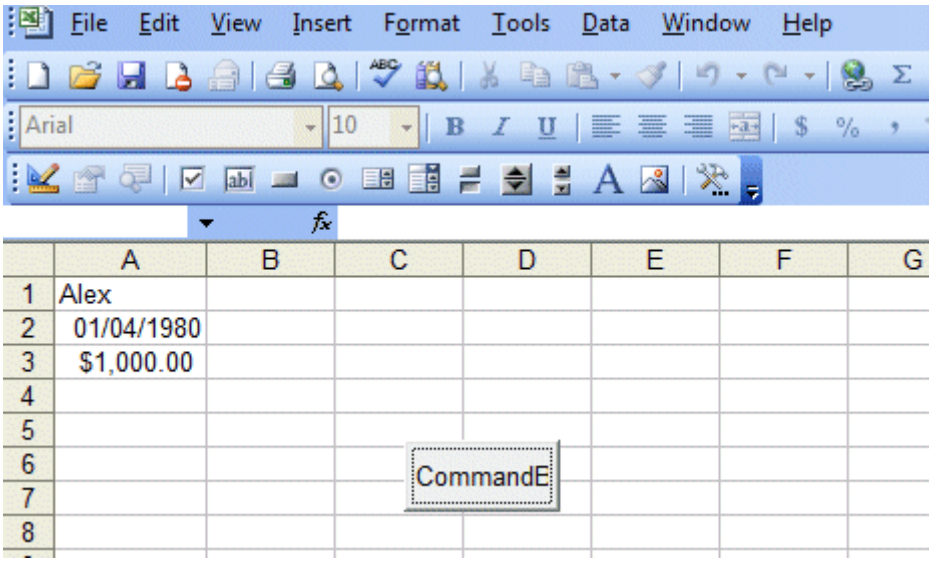

Figure 2.2: Output screen for Example 2.2

2.3 The use of Option Explicit

The use of Option Explicit is to help us to track errors in the usage of variable names within a program code. For example, if we commit a typo, the VBE will pop up an error message “Variable not defined”. Indeed, Option Explicit forces the programmer to declare all the variables using the Dim keyword. It is a good practice to use Option Explicit because it will prevent us from using incorrect

variable names due to typing errors, especially when the program gets larger. With the usage of Option Explicit, it will save us time in debugging our programs.

When Option Explicit is included in the program code, all variables need to be declared using the Dim keyword. Any variable not declared or wrongly typed will cause the program to popup the “Variable not defined” error message. The error needs to be corrected before the program can continue to run.

Example 2.3

This example uses the Option Explicit keyword and it demonstrates how a typo is being tracked.

```
Option Explicit
    Private Sub CommandButton1_Click ()
    Dim YourName As String
    Dim password As String
    YourName = "John"
    password = 12345
    Cells(1, 2) = YourNam
    Cells (1, 3) = password
End Sub
```

The typo is *YourNam* and so the error message ‘variable not defined” will be displayed and the program is suspended, as shown in Figure 2.3.

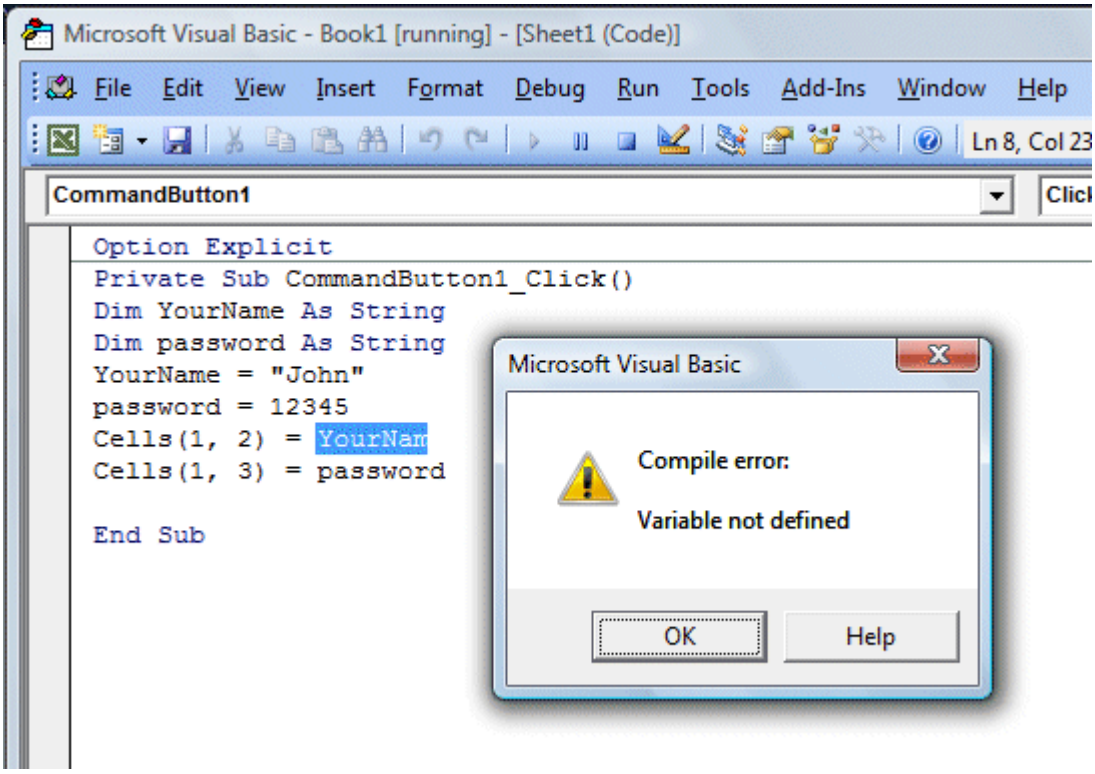

Figure 2.3: Error message due to typo error

2.4 Assigning Values to the Variables

After declaring various variables using the Dim statements, we can assign values to those variables. The general format of an assignment is

Variable=Expression

The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and more. The following are some examples:

firstNumber=100

secondNumber=firstNumber-99

userName="John Lyan"

userpass.Text = password

Label1.Visible = True

```
Command1.Visible = false
ThirdNumber = Val(usernum1.Text)
total = firstNumber + secondNumber+ThirdNumber
```

2.5 Performing Arithmetic Operations in Excel VBA

In order to compute input from the user and to generate results in Excel VBA, we can use various mathematical operators. In Excel VBA, except for + and -, the symbols for the operators are different from normal mathematical operators, as shown in Table 2.5.

Table 2.5: Arithmetic Operators

Operator	Mathematical function	Example
^	Exponential	2^4=16
*	Multiplication	4*3=12
/	Division	12/4=3
Mod	Modulus (return the remainder from an integer division)	15 Mod 4=3
\	Integer Division (discards the decimal places)	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

Example 2.4

```
Option Explicit
    Private Sub CommandButton1_Click ()
    Dim number1, number2, number3 as Single
    Dim total, average as Double
    number1=Cells (1, 1).Value
    number1=Cells (2, 1).Value
```

```
        number3= Cells (3, 1).Value
         Total=number1+number2+number3
        Average=Total/3
        Cells (5, 1) =Total
        Cells (6, 1) =Average
End Sub
```

In example 2.4, three variables are declared as single and another two variables are declared as variant. Variant means the variable can hold any numeric data type. The program computes the total and average of the three numbers that are entered into three cells in the Excel spreadsheet.

Example 2.5

```
        Option Explicit
        Private Sub CommandButton1_Click ()
        Dim secondName As String
        Dim yourName As String
        firstName = Cells (1, 1).Value
        secondName = Cells (2, 1).Value
        yourName = firstName + "  " + secondName
                Cells(3,1) = yourName
        End Sub
```

In this example, three variables are declared as string. The variable firstName and the variable secondName will receive their data entered by the user into Cells (1, 1) and cells (2, 1) respectively. The variable yourName will be assigned the data by combining the first two variables. Finally, yourName is displayed on Cells (3, 1). You will notice that performing arithmetic operation on strings will

result in concatenation of the strings, as shown in figure 2.4 below. Names in A1 and A2 are joined up and displayed in A3.

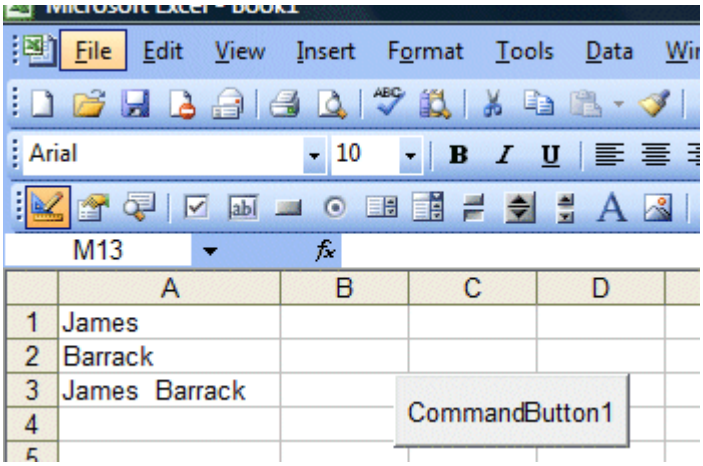

Figure 2.4: Concatenation of Strings