A SunCam online continuing education course

Computer Programming in Excel (VBA)

Part 3: Functions, Procedures, and String Variables

by

Kwabena Ofosu, Ph.D., P.E., PTOE

Abstract

This course is the third of a four-part series on computer programming in *Excel Visual Basic for Applications* (VBA), tailored to practicing engineers. In this course, topics on functions and procedures, and working with string variables are covered. A tour of the VBA object library is also presented. Several examples relevant to engineering are used to illustrate and demonstrate the concepts and methods learned in this class. Two mini-projects are used to demonstrate the programming concepts and methods in situations encountered by practicing engineers.

Computer Programming in Visual Basic (VBA) – Part 1 and Part 2 are not required pre-requisites to this course. It would however be helpful to understand the basic principles of computer programming as well as the fundamentals of the Excel VBA language as presented in Part 1 and Part 2 of this series.

TABLE OF CONTENTS

List of Tables

1. PROCEDURES AND FUNCTIONS

1.1 Procedures

A procedure is a block of code enclosed within a declaration statement and a corresponding *End* declaration. In VBA, all executable statements must be written within some procedure. There are many types of procedures such as **Sub procedures**, **Event procedures**, and **Function procedures**. An advantage of procedures is that once set up they may be used in other parts of the code of an application by **calling** them. Procedures can be used to split large and complex programs into smaller discrete units that are more manageable, and makes it easier to isolate and identify bugs and errors.

1.2 Sub Procedures

A sub procedure is a procedure that performs some sequence of statements, actions or tasks. The statements are enclosed by the ***Sub*** and ***End Sub*** statements. The general structure of a VBA sub procedure is as follows

```
AccessModifier Sub Nameofprocedure( )

        'type code to executed, here

End Sub
```

The access modifier controls access to the sub procedure. An access modifier may be ***Private***, or ***Public***. A private procedure can only be called by other procedures in the same module whereas a public procedure can be called by procedures in the same module as well as procedures of other modules in the Excel workbook.

For example consider a sub procedure that performs some calculation.

```
Private Sub CalculationCode( )

        'type code to perform the calculations

End Sub
```

Consider that now the programmer would like to call this sub procedure within a larger sub procedure that conducts the calculation and sends the results to be displayed on a spreadsheet.

```
Private Sub ProcessToSpreadsheet( )

    'call sub procedure to perform calculations              calling the
        CalculationCode  ←————————————————————————           pre-existing
    'type code to send calculation results to a spreadsheet  sub procedure
        ……….

End Sub
```

1.3 Naming a Procedure

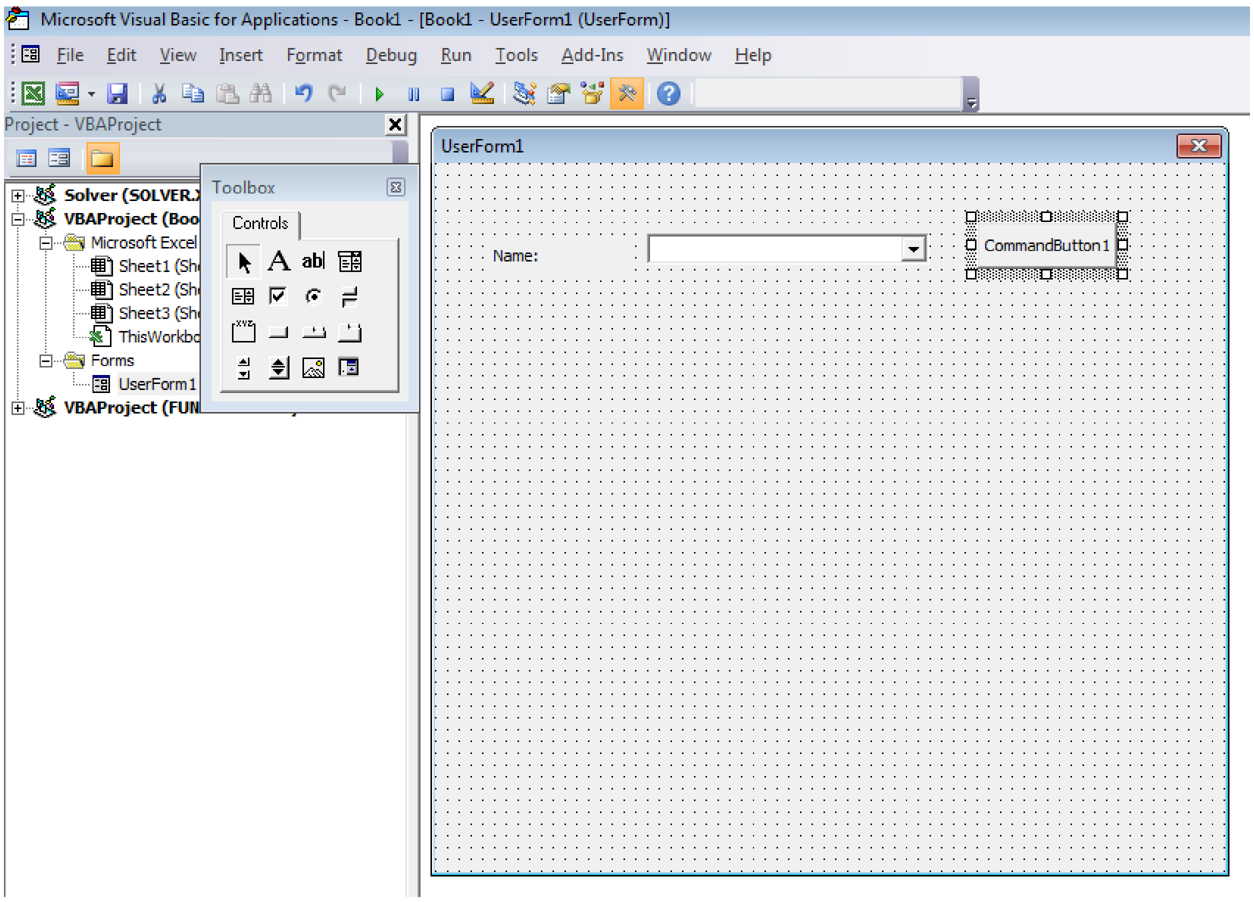
The naming of a procedure follows the same rules as naming a variable. Also, naming conventions can be applied to the name of the procedure. Examples of common elements of naming conventions used include:

1. If the procedure involves some action that can be described by a verb, use that verb as the name or a prefix to the name of the procedure. For example S*how*, *Close*, *Calculate*, etc.
2. Starting the name with an upper case letter is a common practice.
3. If a combination of words is used, start each word with an uppercase letter.
4. Use names that clearly identify the main purpose of the procedure. For example *CalculateAverages( )*.

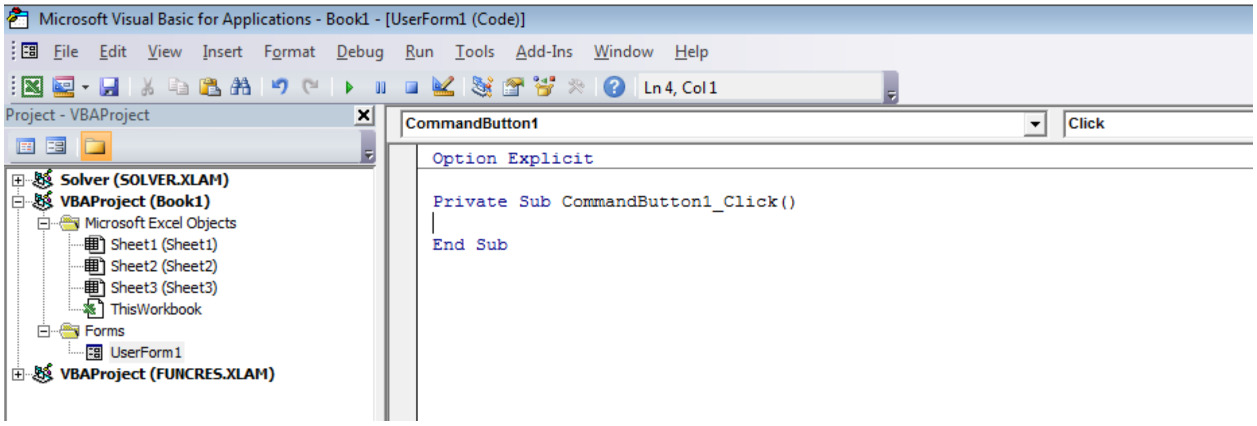
1.4 Event Procedures

An event procedure (or event-handling procedure) is a sub procedure that executes a set of instructions in response to an event, typically a user action, on an object. For example clicking on a command button, or double clicking on label, or changing the value of a combo box, etc. VBA provides event handling procedures for the various objects and their events. Once opened the programmer then develops the code that is to execute once that event is invoked by the user.

Open a VBA session.
Insert a new UserForm.
Drag a label, a combo box, and a command button onto the form, from the Toolbox.

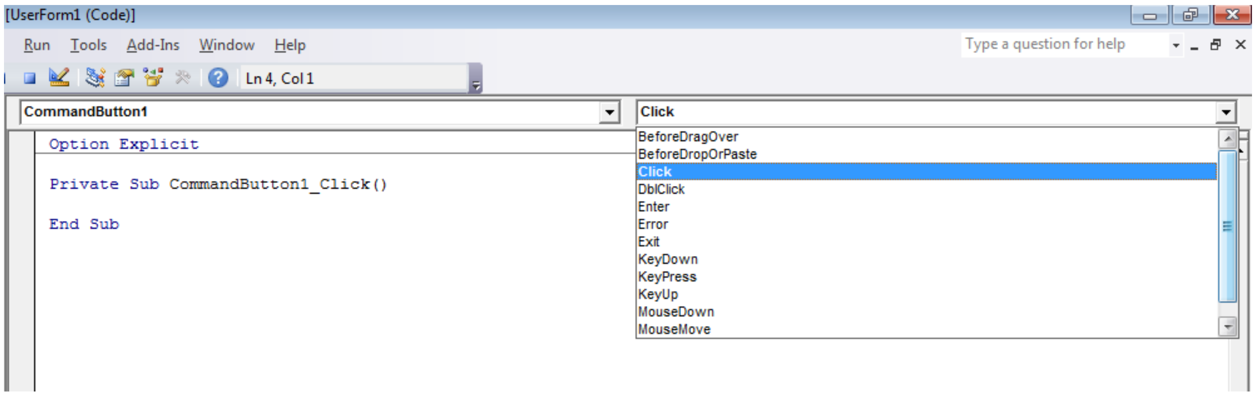


Double click on the command button to open the code window.



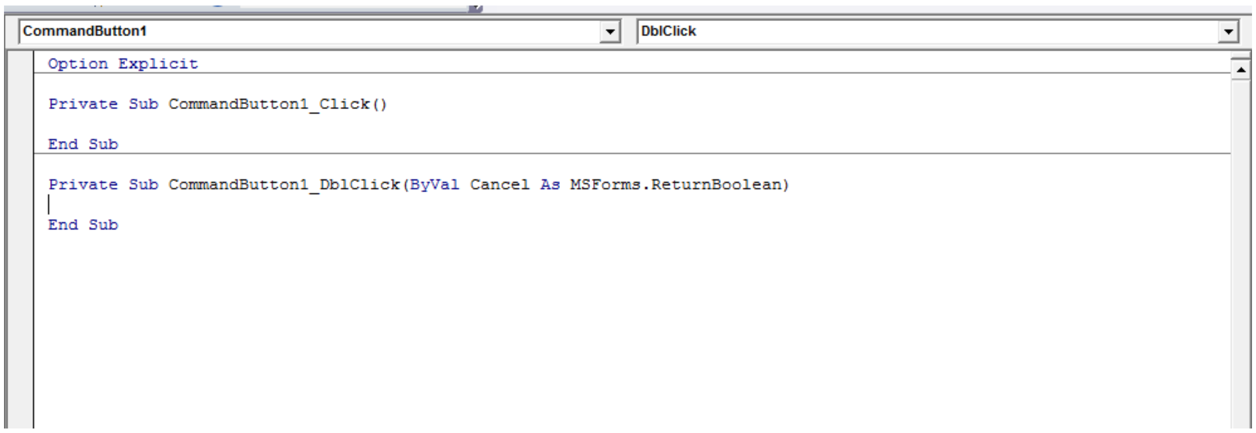
```
Option Explicit

Private Sub CommandButton1_Click()

End Sub
```

The *CommandButton1_Click( )* event procedure opens, and the cursor is in the **main body** of the procedure. By typing instructions in the main body of this event procedure, the programmer can set up what will happen when this button in clicked on in run time by a user.
Click on the event drop down to see other event procedures available for the *CommandButton1*.



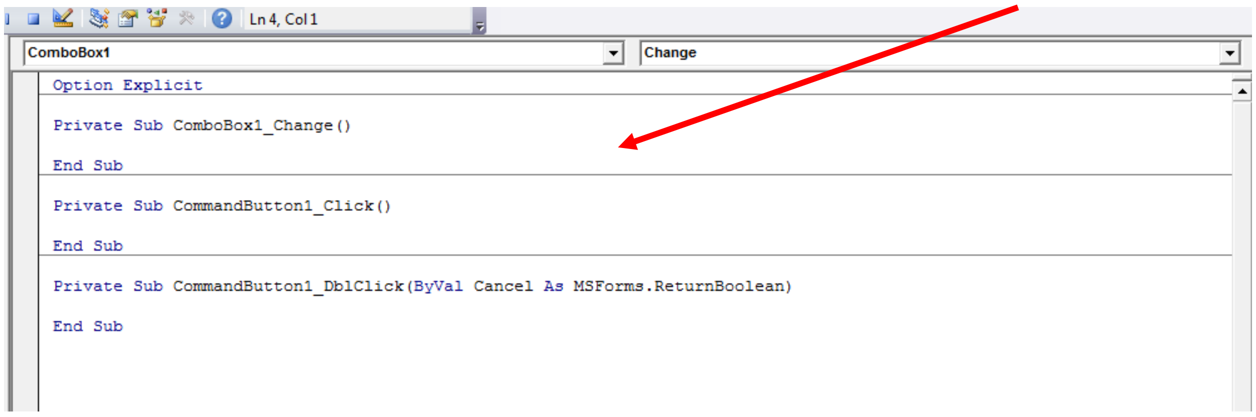
```
Option Explicit

Private Sub CommandButton1_Click()

End Sub
```

Select the *DblClick* event
A double-click sub procedure opens, and the programmer may develop the code that will be executed if a user were to double click on this command button in run time.



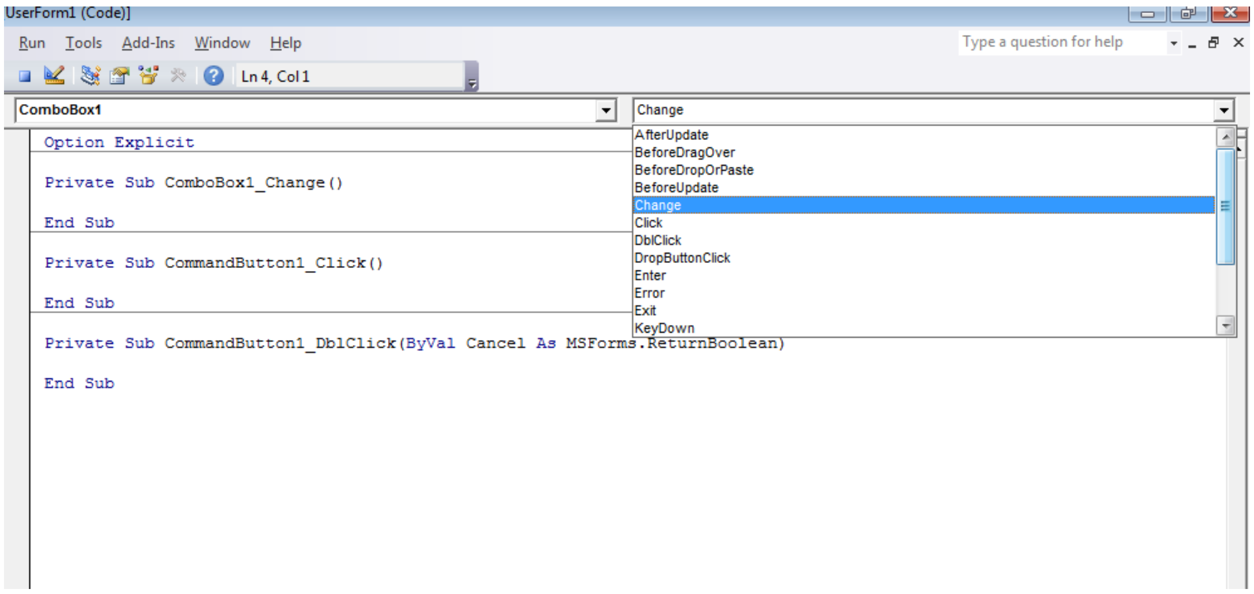
```
Option Explicit

Private Sub CommandButton1_Click()

End Sub

Private Sub CommandButton1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)

End Sub
```

Return to Object View
Select the combo box, and double click on it to see the events that can be associated with it.



```
Option Explicit

Private Sub ComboBox1_Change()

End Sub

Private Sub CommandButton1_Click()

End Sub

Private Sub CommandButton1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)

End Sub
```

The *Change* procedure for the combo box opens. The programmer can enter code that will execute if the selected value of the combo box is changed in run time.
Click on the event drop down to review other events supported for a combo box.



```
Option Explicit

Private Sub ComboBox1_Change()

End Sub

Private Sub CommandButton1_Click()

End Sub

Private Sub CommandButton1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)

End Sub
```

Note that in all cases so far, the event procedure name always consists of the control name, an underscore, and the procedure's event name. If the intent is to write code for the click event of *ComboBox1*, rather than the *Change* event, then the click event will be selected from the event drop down list to create the procedure and the user will then add the relevant code to the main body of the procedure. Alternately, the *ComboBox1_Change ( )* can be typed over and amended to read *ComboBox1_Click ( )*, and the relevant code then added to the main body.

1.5 Function Procedures

A function procedure (or **function**), just like a sub procedure, performs some sequence of statements, actions or tasks, but in addition, it always returns a value. The value will be stored in a variable that bears the same name as the function. Therefore, somewhere in the main body of the function, there must always be an assignment of some value or variable to the function name variable which will then be returned to the user at the location in the code where the function is called.

Function names follow the same rules for variables as well as the same conventions for procedures. The access modifiers work the same as with sub procedures.

The basic syntax for setting up a function in VBA is

```
AccessModifier Function NameoftheFunction( ) As DataType

        'type the code for the calculation

        'assign some result to the variable NameoftheFunction
        'to be returned to the user

End Function
```

where *DataType* is the data type of the output (the returned value).

A function may need some external input value(s) in order to perform the calculations. These inputs are called **arguments.** Arguments are essentially variables and hence they must be declared. They are declared within the function declaration statement. For example

AccessModifier Function NameoftheFunction(argument1 As datatype1) As DataType

'type the code for the calculation

'assign result(s) to the variable NameoftheFunction

End Function

When an argument is **passed** to a function, it may be passed by value (***ByVal***) or passed by reference (***ByRef***). When an argument is passed by value, a copy of the argument's value is sent to the function for the calculations to proceed, and the function cannot alter the original value of the argument. When an argument is passed by reference, a reference (or a path, or address) to the argument stored in memory is sent to the function, and the function will have access to the argument in its memory location, and may therefore change the original value of the argument. A full blown function procedure will therefore be of the form

AccessModifier Function NameoftheFunction(ByVal argument1 As datatype1) As DataType

'type the code for the calculation

'assign result(s) to the variable NameoftheFunction

End Function

For example

Private Function MyBonus(ByVal curAnnualSales As Currency) As Currency

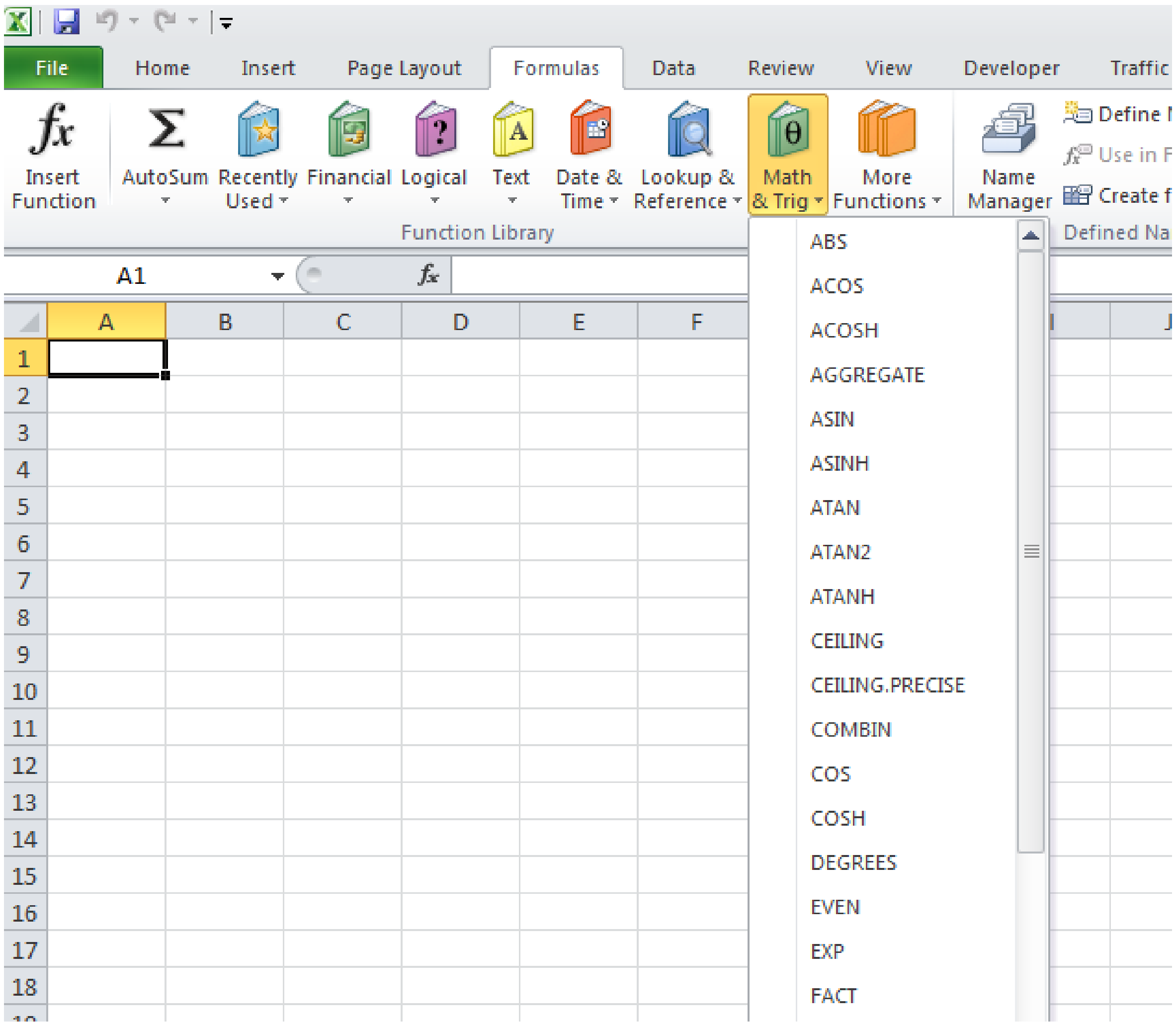
'type the code to calculate the bonus amount

'assign result(s) to the variable MyBonus

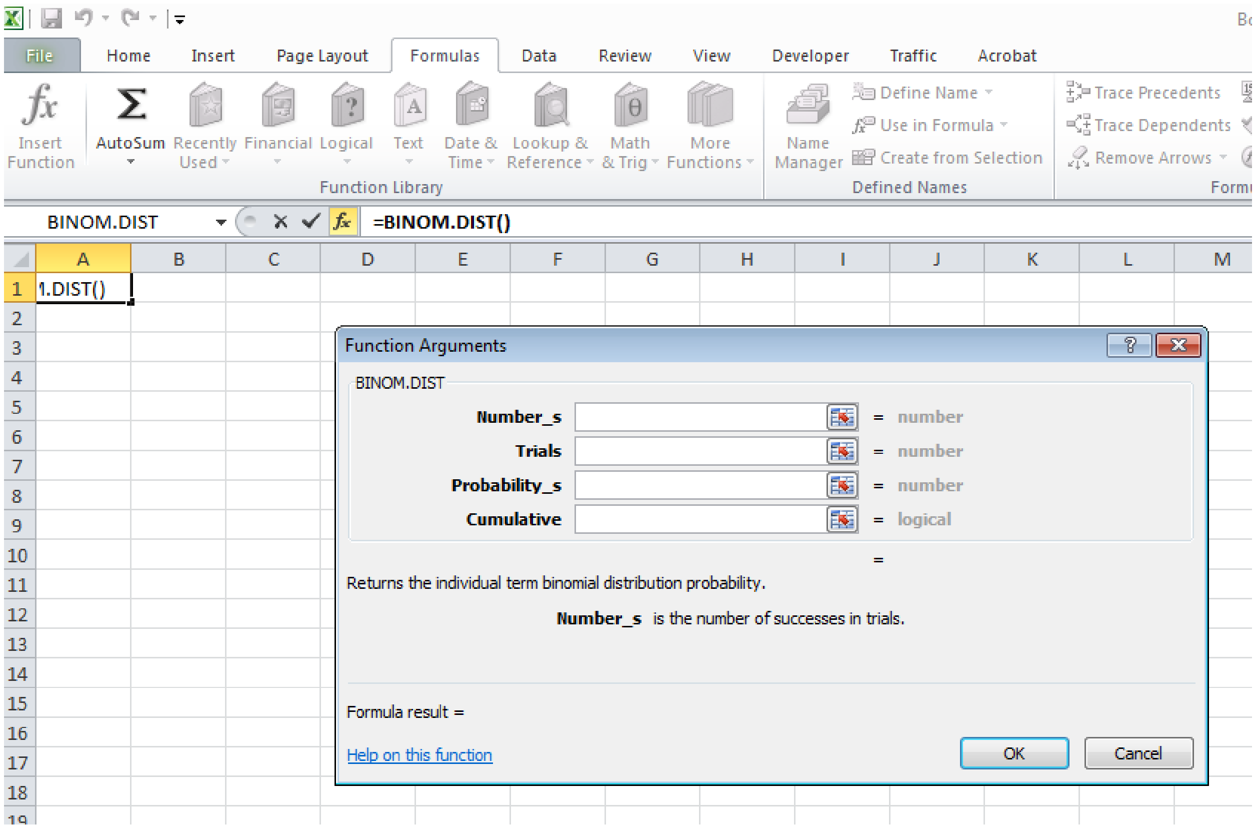
End Function

1.6 Built-in Functions

Excel comes with an extensive library of functions. It is advantageous to check for a built-in function before creating custom built functions. To use a built-in function in the code, the programmer must call it and pass the required or relevant arguments. A preview of available built-in functions can be seen from the Excel Formulas tab which has functions subdivided by area of specialty such as Statistical Formulas, Math and Trigonometry, Financial, and many other areas.



By selecting a function the inputs required (the arguments) can be seen from the formula window. For example, calculation of probabilities from a Binomial Distribution.



Function Arguments

BINOM.DIST

Number_s		= number
Trials		= number
Probability_s		= number
Cumulative		= logical

Returns the individual term binomial distribution probability.

Number_s is the number of successes in trials.

Formula result =

The typical set up for using a built-in function in VBA code is of the structure

avariable = Application.Builtinfunction(argument1, argument2, ...)

If the arguments are from cells on a spreadsheet then the *WorksheetFunction* property is appended to the *Application* class as follows

avariable = Application.WorksheetFunction.Builtinfunction(argument1, argument2, ...)

Example: To calculate an average bonus to company employees, the *Average* built-in function may be used.

```
curAverageBonus =  Application.Average(curEmployeeBonus1, curEmployeeBonus2, _
        curEmployeeBonus3, curEmployeeBonus4, curEmployeeBonus5, curEmployeeBonus6)
```

Example: To calculate the median test score using the *Median* built-in function whereby the test scores are being read from a range of cells on a spreadsheet.

```
dblMedScore =  Application.WorksheetFunction.Median(Range(Sheets("Sheet1").cells(2,2), _
                                                     Sheets("Sheet1").cells(2,20)))
```

In the VBA code window, as one types the *Application.*, or *Application.WorksheetFunction.*, the Excel VBA function library provides a tentative list of candidate function names in alphabetical order from which the programmer may select the function that is needed.

1.7 Constants

Within some functions or procedure, it may be necessary to establish some constant value to facilitate the calculations. For example, acceleration due to gravity, or the value of π, etc. A constant is created as follows

```
Const nameofconstant = valueofconstant
```

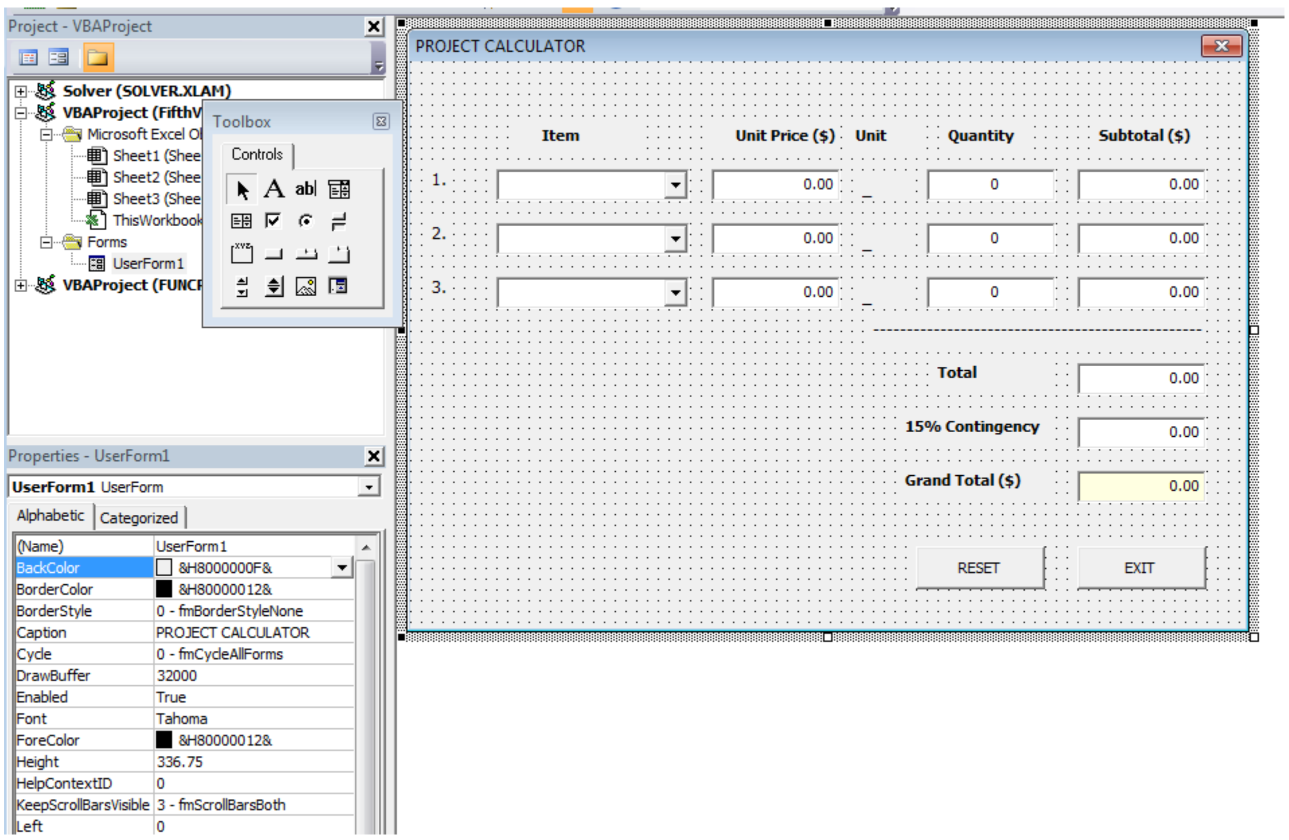
The *Const* keyword ensures the value cannot be changed. For example"

```
Const dblAccByGravity = 9.81
```

1.8 Fifth VBA Project

Problem Statement
In this project, the construction project calculator developed in the *Second VBA Project* will be modified to incorporate functions to implement the calculations. Each line item will have an *Item* which will be selected from a drop down list. On selecting the *Item* from the drop down list, the relevant *Unit Price* and *Units* (of measure) will automatically populate the *Unit Price* and *Units* controls respectively. On entering the *Quantity* on a line, the subtotal for that line item will automatically calculate and display the result in the *Subtotal* control. The calculator will have a general text box to enter a contingency amount which will be added to the total of the subtotals to produce the grand total.

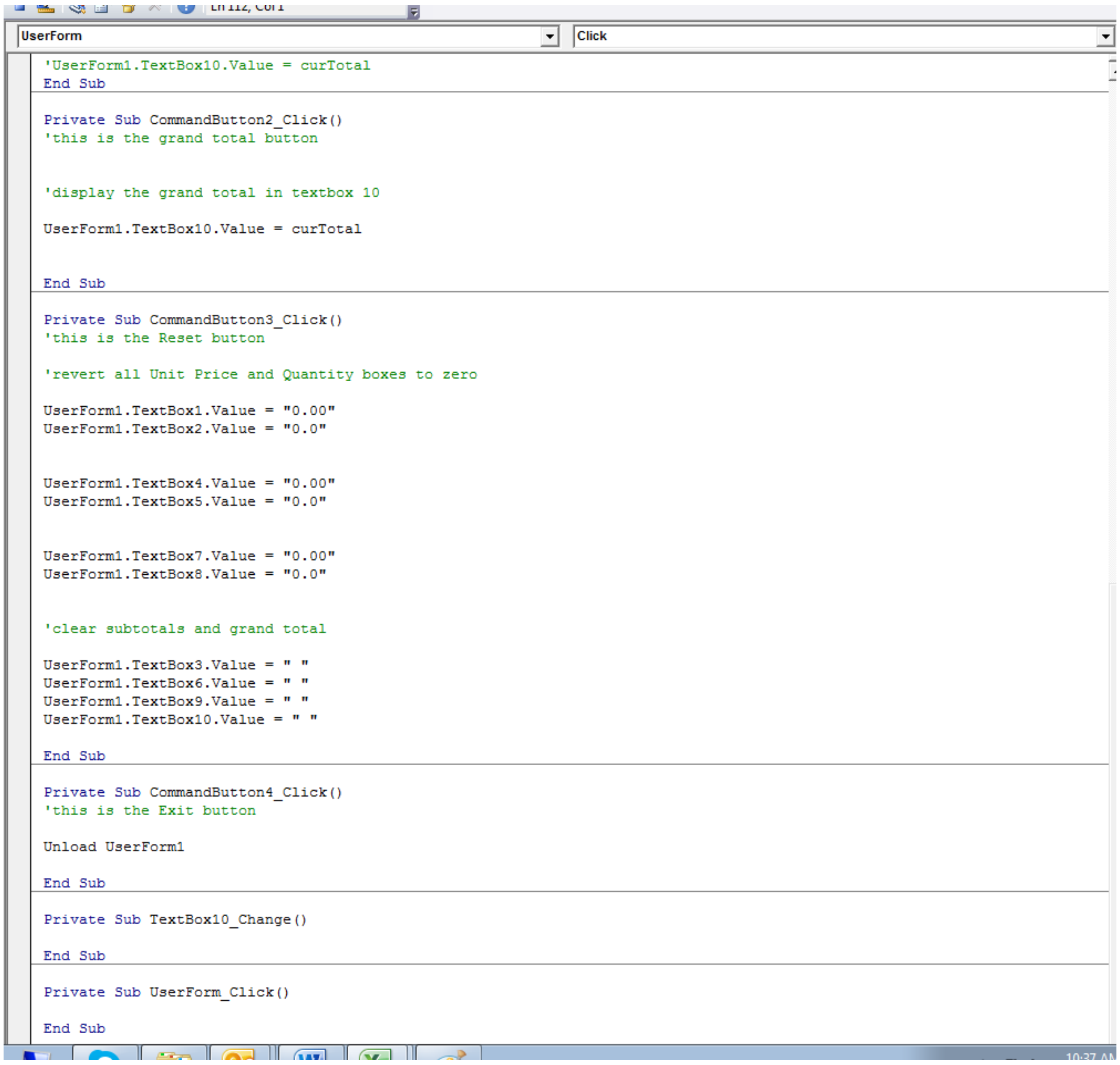
Solution
Make a copy of the *SecondVBAProject* workbook and rename it *FifthVBAProject*.
Open the application in the VBA Environment design time and modify it as follows



(Name)	UserForm1
BackColor	&H8000000F&
BorderColor	&H80000012&
BorderStyle	0 - fmBorderStyleNone
Caption	PROJECT CALCULATOR
Cycle	0 - fmCycleAllForms
DrawBuffer	32000
Enabled	True
Font	Tahoma
ForeColor	&H80000012&
Height	336.75
HelpContextID	0
KeepScrollBarsVisible	3 - fmScrollBarsBoth
Left	0

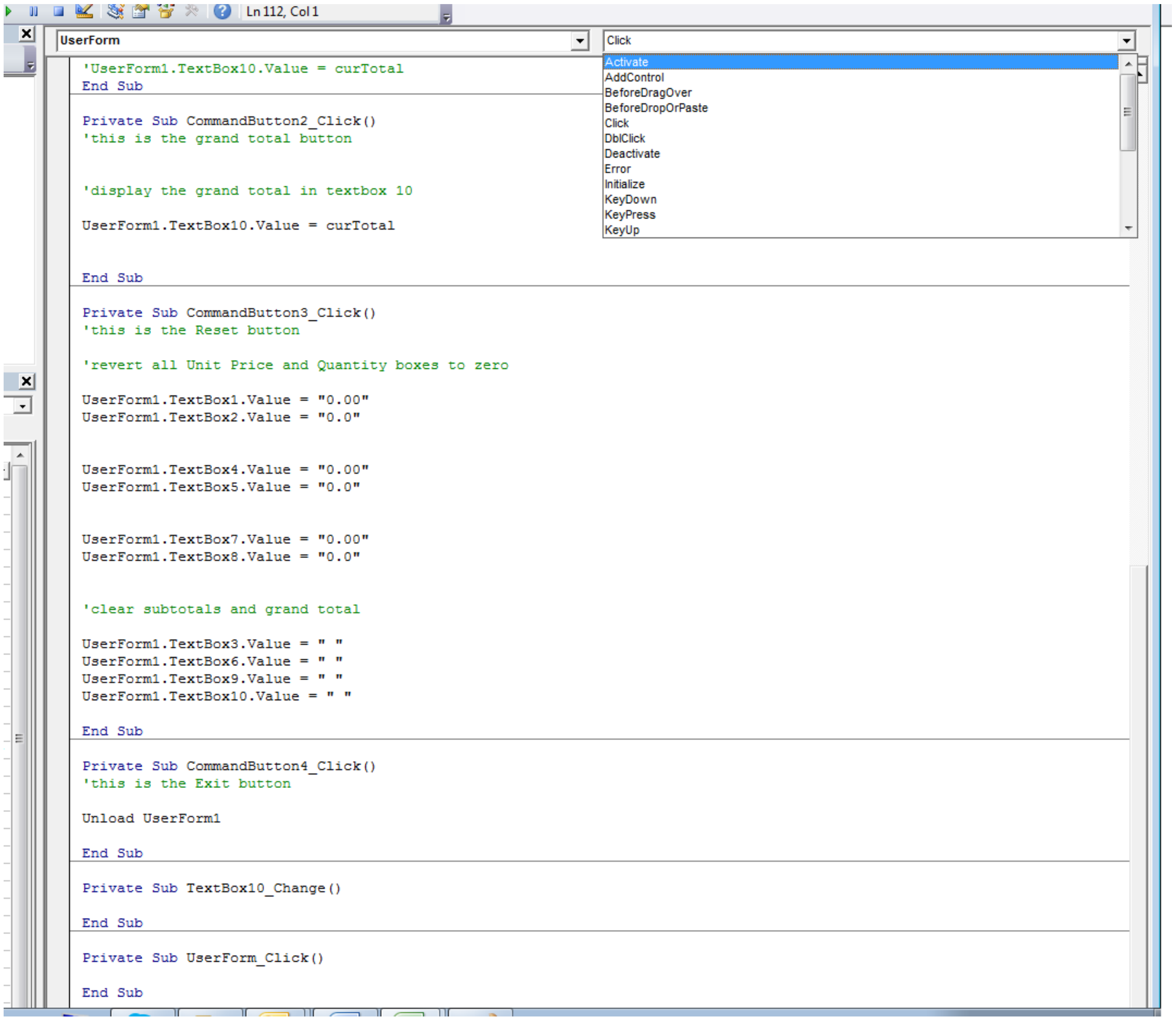
The combo boxes will list the construction work items Concrete, Asphalt, and Road Markings. The lists are to be loaded once the UserForm opens.

Double click on the UserForm to open its code window.
This pre-existing code from *Second VBA Project* will be modified for this project.



```
'UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button


'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub

Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price and Quantity boxes to zero

UserForm1.TextBox1.Value = "0.00"
UserForm1.TextBox2.Value = "0.0"


UserForm1.TextBox4.Value = "0.00"
UserForm1.TextBox5.Value = "0.0"


UserForm1.TextBox7.Value = "0.00"
UserForm1.TextBox8.Value = "0.0"


'clear subtotals and grand total

UserForm1.TextBox3.Value = " "
UserForm1.TextBox6.Value = " "
UserForm1.TextBox9.Value = " "
UserForm1.TextBox10.Value = " "

End Sub

Private Sub CommandButton4_Click()
'this is the Exit button

Unload UserForm1

End Sub

Private Sub TextBox10_Change()

End Sub

Private Sub UserForm_Click()

End Sub
```

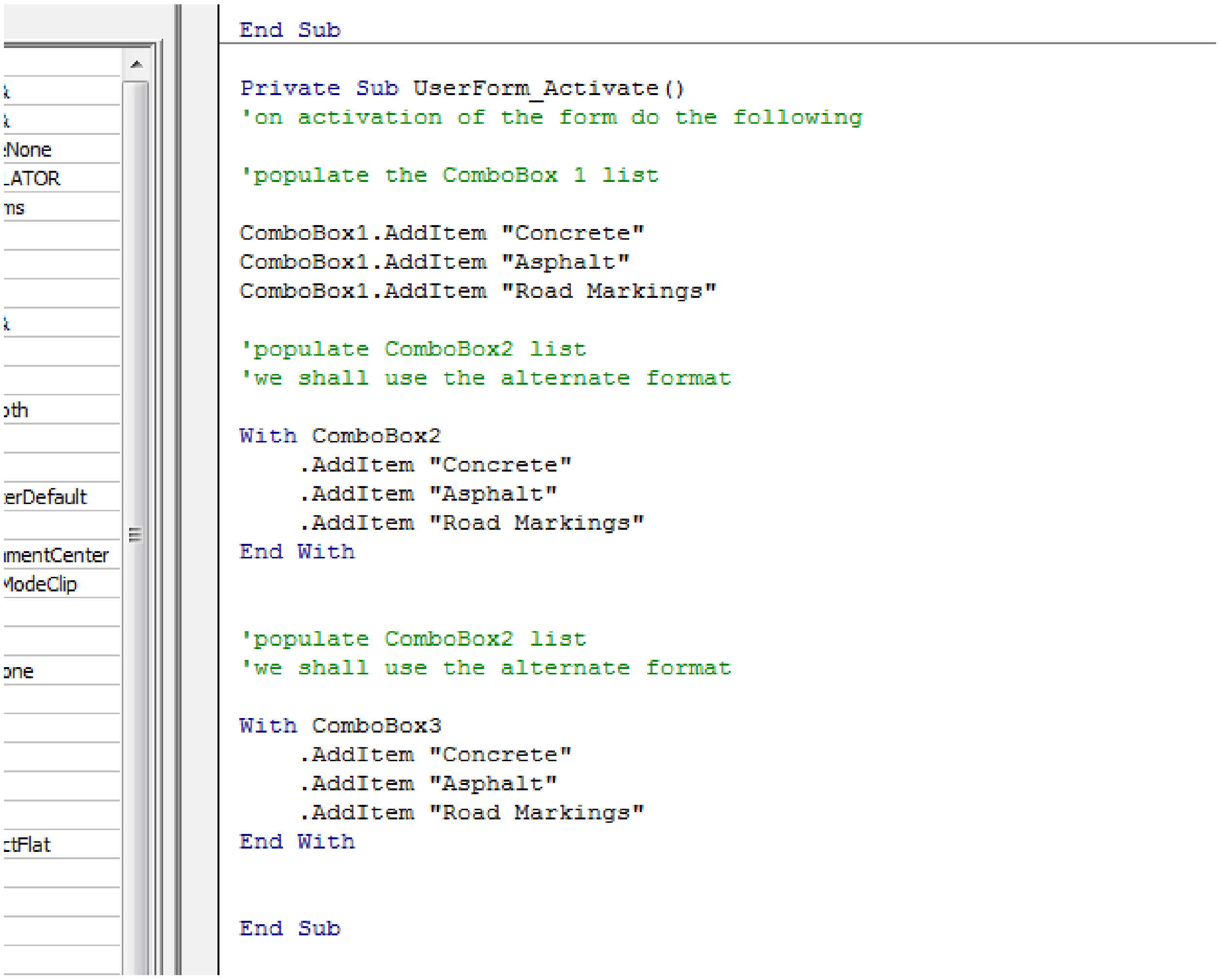
With cursor in the *UserForm1 Click* procedure, change the event in the event window to *Activate*.



```
'UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button


'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub

Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price and Quantity boxes to zero

UserForm1.TextBox1.Value = "0.00"
UserForm1.TextBox2.Value = "0.0"


UserForm1.TextBox4.Value = "0.00"
UserForm1.TextBox5.Value = "0.0"


UserForm1.TextBox7.Value = "0.00"
UserForm1.TextBox8.Value = "0.0"


'clear subtotals and grand total

UserForm1.TextBox3.Value = " "
UserForm1.TextBox6.Value = " "
UserForm1.TextBox9.Value = " "
UserForm1.TextBox10.Value = " "

End Sub

Private Sub CommandButton4_Click()
'this is the Exit button

Unload UserForm1

End Sub

Private Sub TextBox10_Change()

End Sub

Private Sub UserForm_Click()

End Sub
```

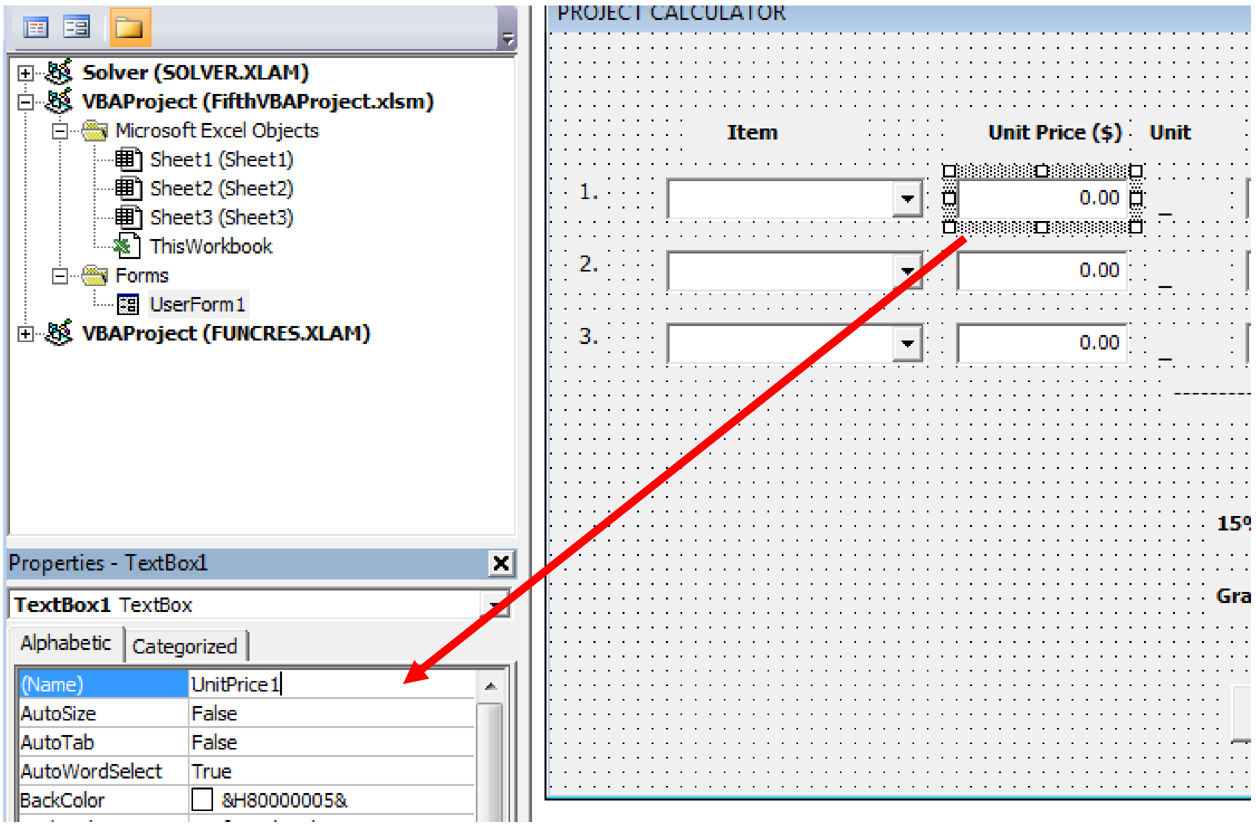
In other words when the form activates, execute the codes in that procedure.

Rename the combo boxes *ComboBox1, ComboBox2*, and *ComboBox3* respectively.
Use the *AddItems* method to populate the combo boxes' lists.



```
End Sub

Private Sub UserForm_Activate()
'on activation of the form do the following

'populate the ComboBox 1 list

ComboBox1.AddItem "Concrete"
ComboBox1.AddItem "Asphalt"
ComboBox1.AddItem "Road Markings"

'populate ComboBox2 list
'we shall use the alternate format

With ComboBox2
     .AddItem "Concrete"
     .AddItem "Asphalt"
     .AddItem "Road Markings"
End With


'populate ComboBox2 list
'we shall use the alternate format

With ComboBox3
     .AddItem "Concrete"
     .AddItem "Asphalt"
     .AddItem "Road Markings"
End With


End Sub
```

Rename the unit price text boxes *UnitPrice1*, *UnitPrice2*, and *UnitPrice3* respectively.

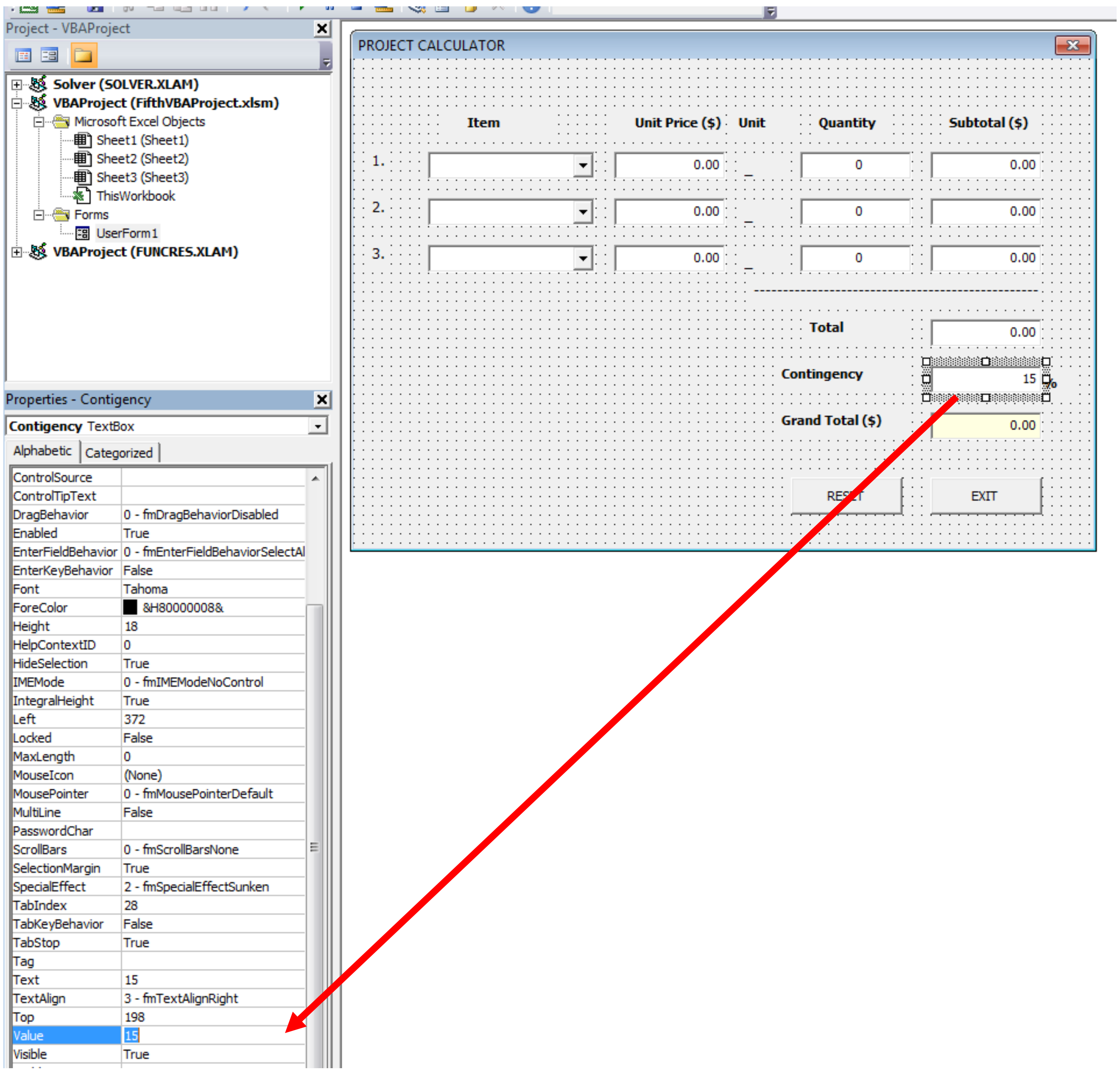


(Name)	UnitPrice1
AutoSize	False
AutoTab	False
AutoWordSelect	True
BackColor	&H80000005&

The text boxes assume the new names and must be called as such in the code from this point onwards.

Rename the quantity text boxes *Quantity1*, *Quantity2*, and *Quantity3* respectively.
Repeat the process for the three subtotal textboxes.
Rename the grand total textbox *GrandTotal*, the total textbox *Total*, and the 15% contingency textbox *Contingency*.

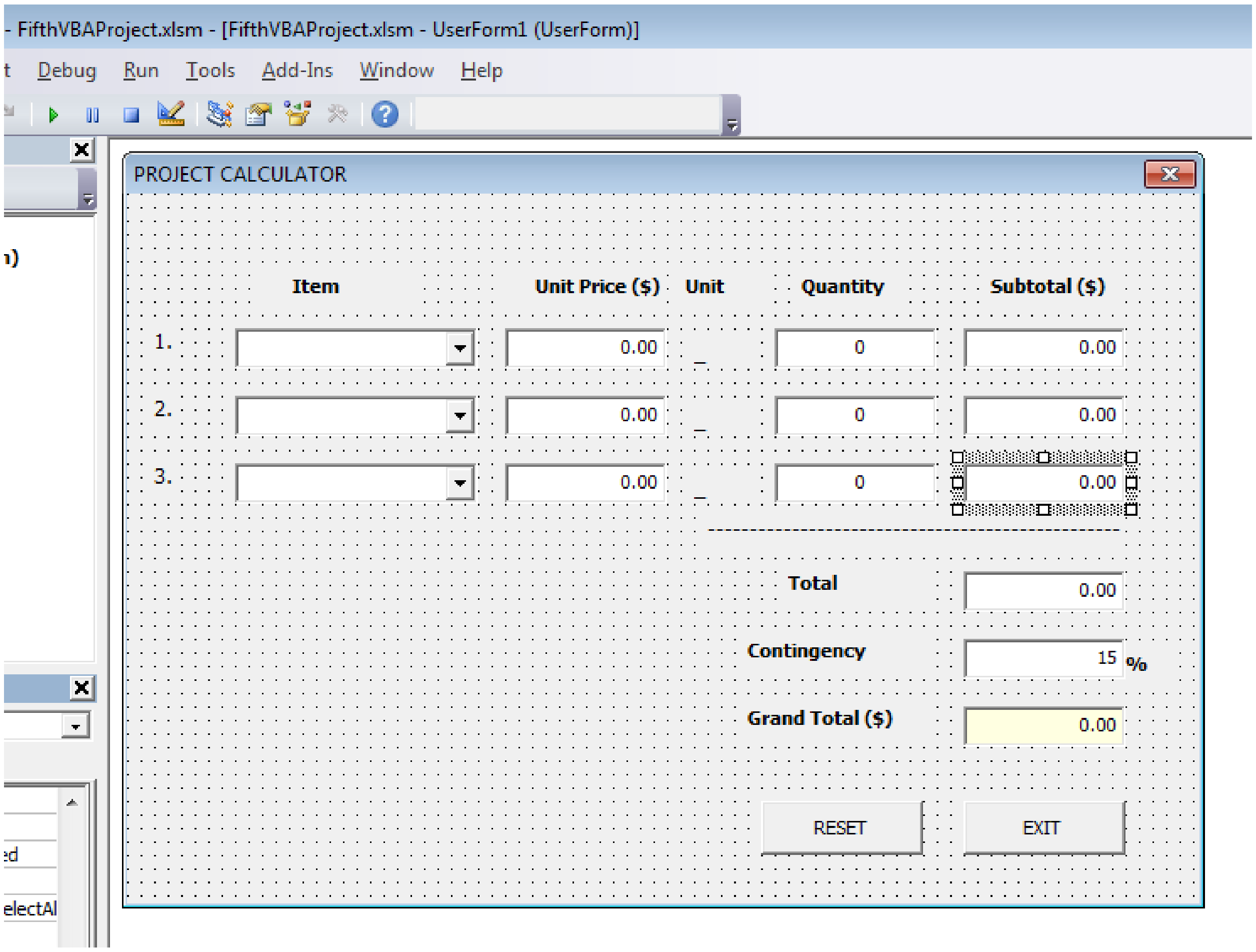
Modify the "15% Contingency" label to read "Contingency", and place a label with the "%" sign on the other side of the *Contingency* text box.
Click on the Contingency text box and change the **Value property** to "15"



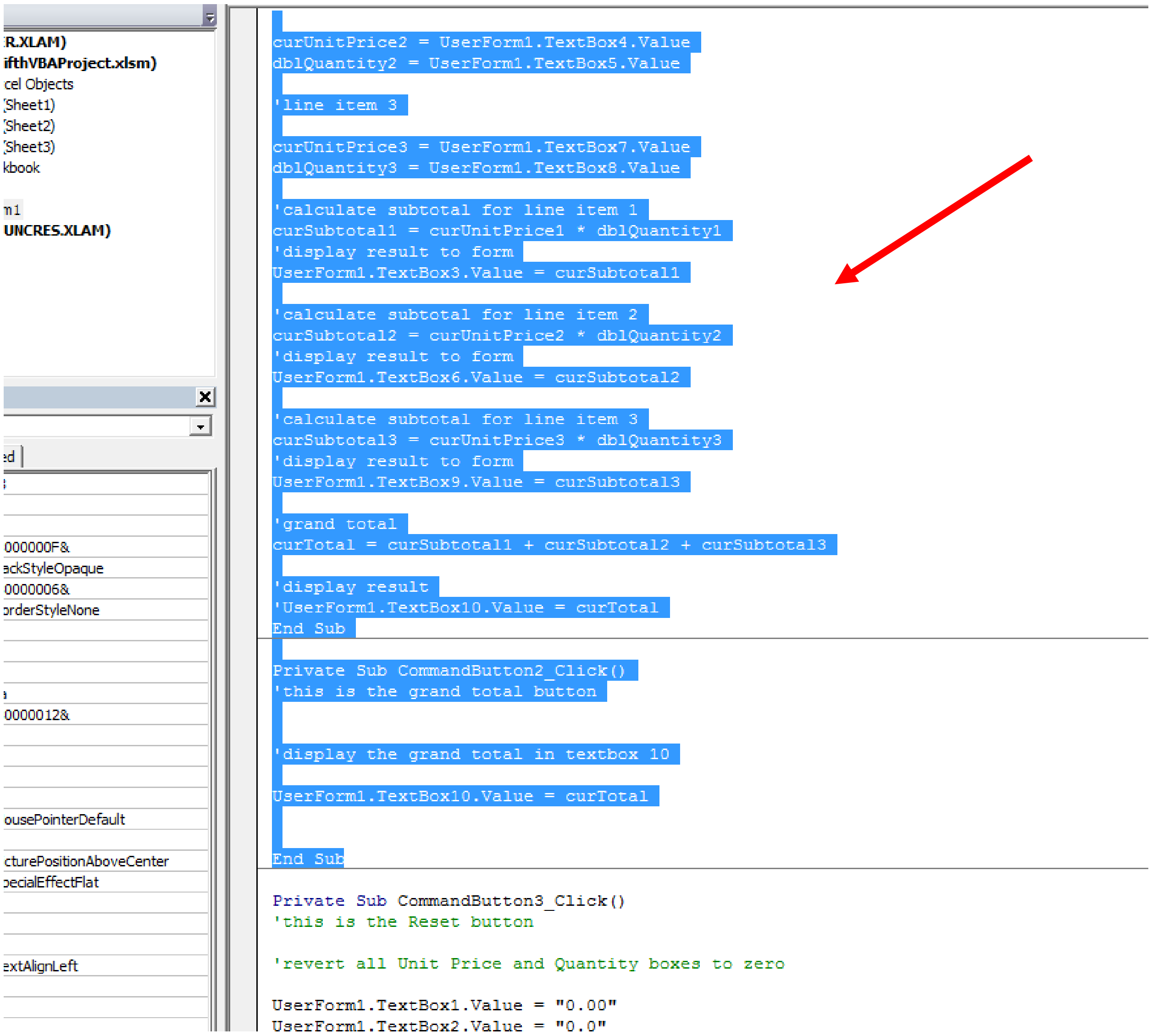
ControlSource	
ControlTipText	
DragBehavior	0 - fmDragBehaviorDisabled
Enabled	True
EnterFieldBehavior	0 - fmEnterFieldBehaviorSelectAl
EnterKeyBehavior	False
Font	Tahoma
ForeColor	■ &H80000008&
Height	18
HelpContextID	0
HideSelection	True
IMEMode	0 - fmIMEModeNoControl
IntegralHeight	True
Left	372
Locked	False
MaxLength	0
MouseIcon	(None)
MousePointer	0 - fmMousePointerDefault
MultiLine	False
PasswordChar	
ScrollBars	0 - fmScrollBarsNone
SelectionMargin	True
SpecialEffect	2 - fmSpecialEffectSunken
TabIndex	28
TabKeyBehavior	False
TabStop	True
Tag	
Text	15
TextAlign	3 - fmTextAlignRight
Top	198
Value	15
Visible	True

The *Units* labels are to show an underscore representing a blank. Later, code shall be added such that if an item is selected from a combo box, the relevant units of measure for that item will display in the corresponding *Units* label.
Rename the *Units* labels *Label31*, *Label32*, and *Label33* respectively.

At this stage the form should look like the following

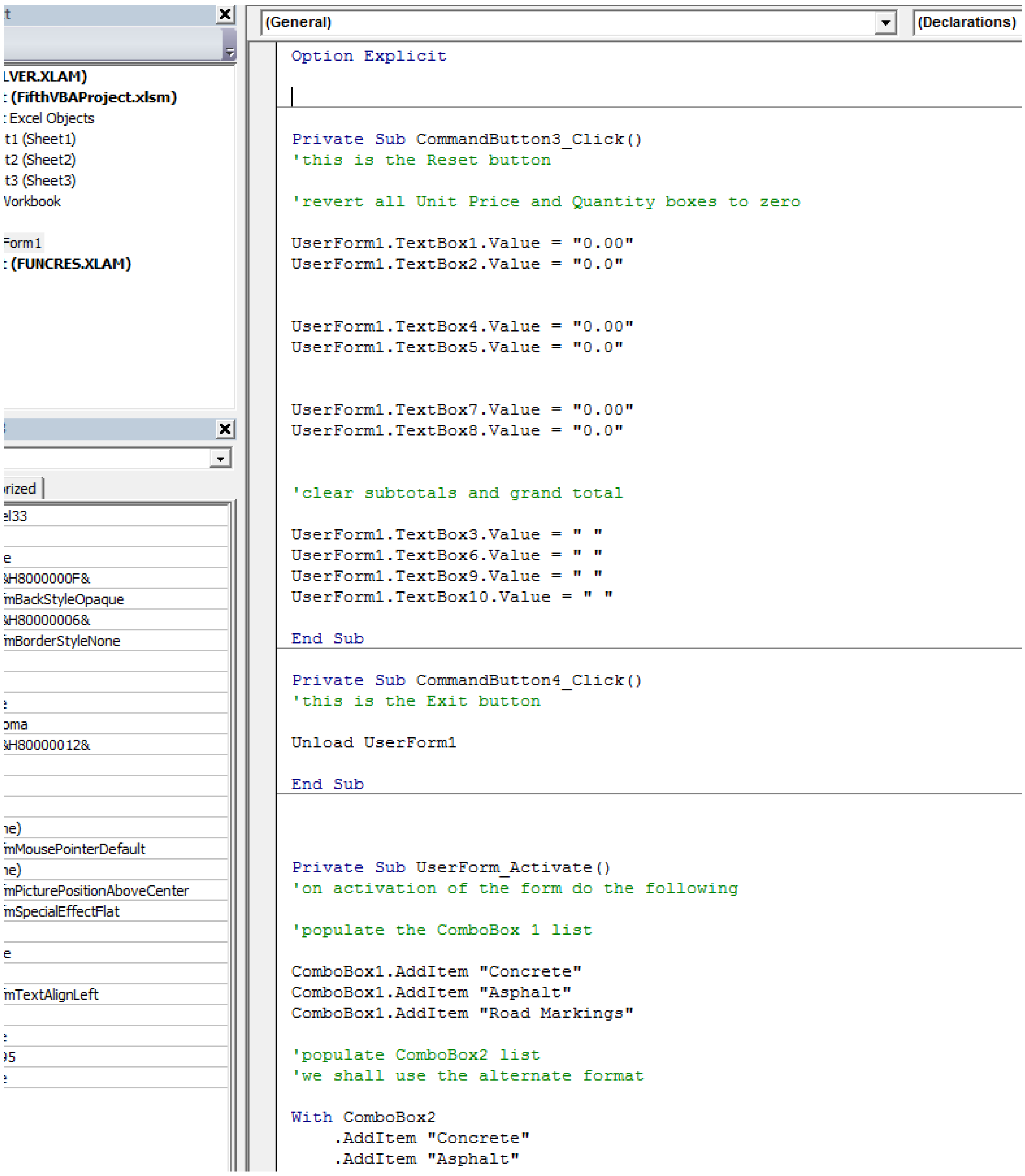


Double click on the form to open the code window.
For the new project calculator, the pre-existing *CommandButton1* and *CommandButton2* procedures will not be used as these controls will be deleted.
Select the procedures for *CommandButton1* and *CommandButton2* and delete them.



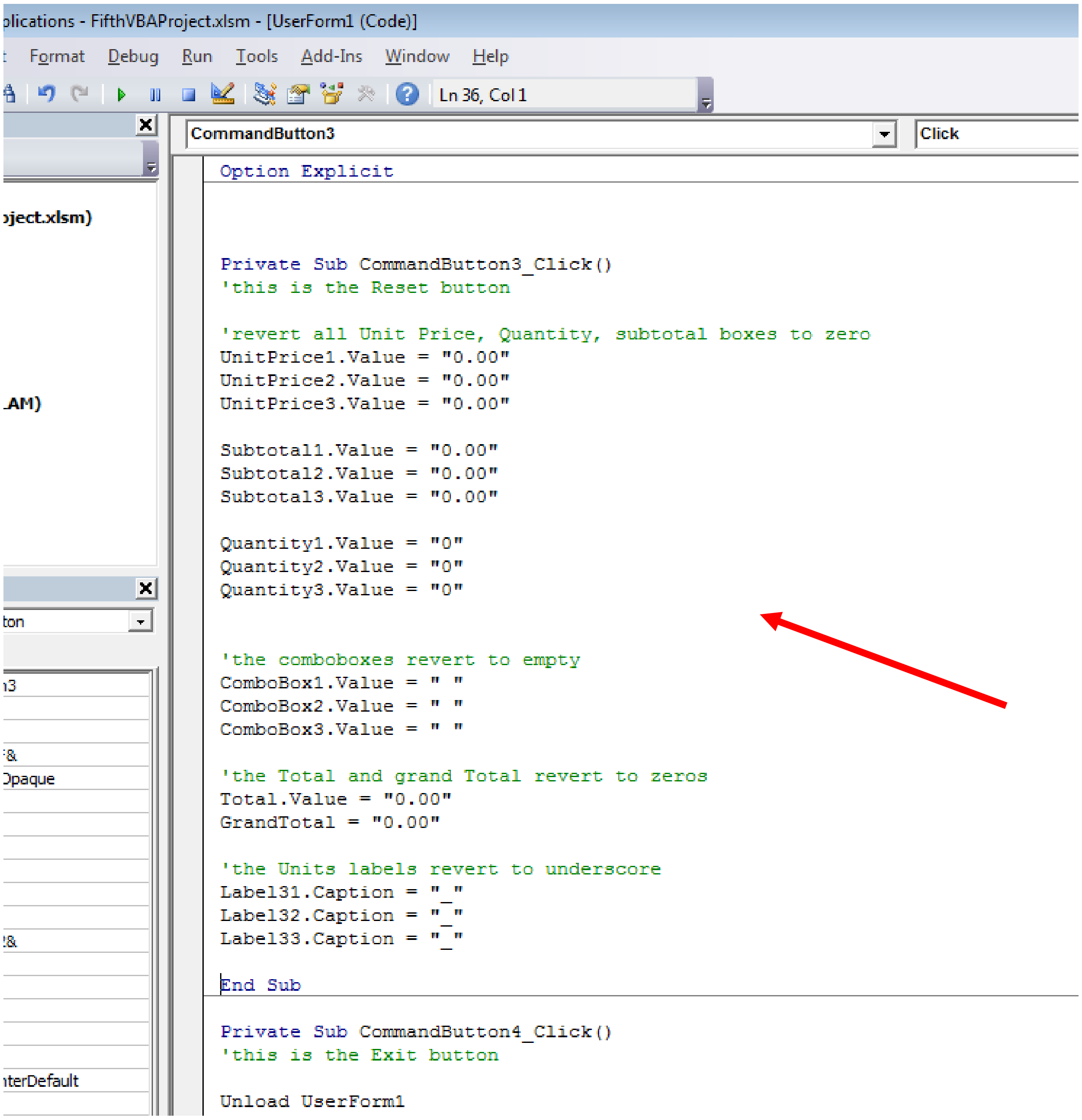
```
curUnitPrice2 = UserForm1.TextBox4.Value
dblQuantity2 = UserForm1.TextBox5.Value

'line item 3

curUnitPrice3 = UserForm1.TextBox7.Value
dblQuantity3 = UserForm1.TextBox8.Value

'calculate subtotal for line item 1
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result to form
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
'UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button


'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub

Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price and Quantity boxes to zero

UserForm1.TextBox1.Value = "0.00"
UserForm1.TextBox2.Value = "0.0"
```

The code window (truncated view) is now



```
Option Explicit


Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price and Quantity boxes to zero

UserForm1.TextBox1.Value = "0.00"
UserForm1.TextBox2.Value = "0.0"


UserForm1.TextBox4.Value = "0.00"
UserForm1.TextBox5.Value = "0.0"


UserForm1.TextBox7.Value = "0.00"
UserForm1.TextBox8.Value = "0.0"


'clear subtotals and grand total

UserForm1.TextBox3.Value = " "
UserForm1.TextBox6.Value = " "
UserForm1.TextBox9.Value = " "
UserForm1.TextBox10.Value = " "

End Sub

Private Sub CommandButton4_Click()
'this is the Exit button

Unload UserForm1

End Sub



Private Sub UserForm_Activate()
'on activation of the form do the following

'populate the ComboBox 1 list

ComboBox1.AddItem "Concrete"
ComboBox1.AddItem "Asphalt"
ComboBox1.AddItem "Road Markings"

'populate ComboBox2 list
'we shall use the alternate format

With ComboBox2
    .AddItem "Concrete"
    .AddItem "Asphalt"
```

The *CommandButton3* procedure (the *Reset* button) needs to be modified to call the new names of the controls on the form.


plications - FifthVBAProject.xlsm - [UserForm1 (Code)]
Format Debug Run Tools Add-Ins Window Help
Ln 36, Col 1
CommandButton3
Click
oject.xlsm)
AM)
ton
n3
&
Opaque
&
nterDefault

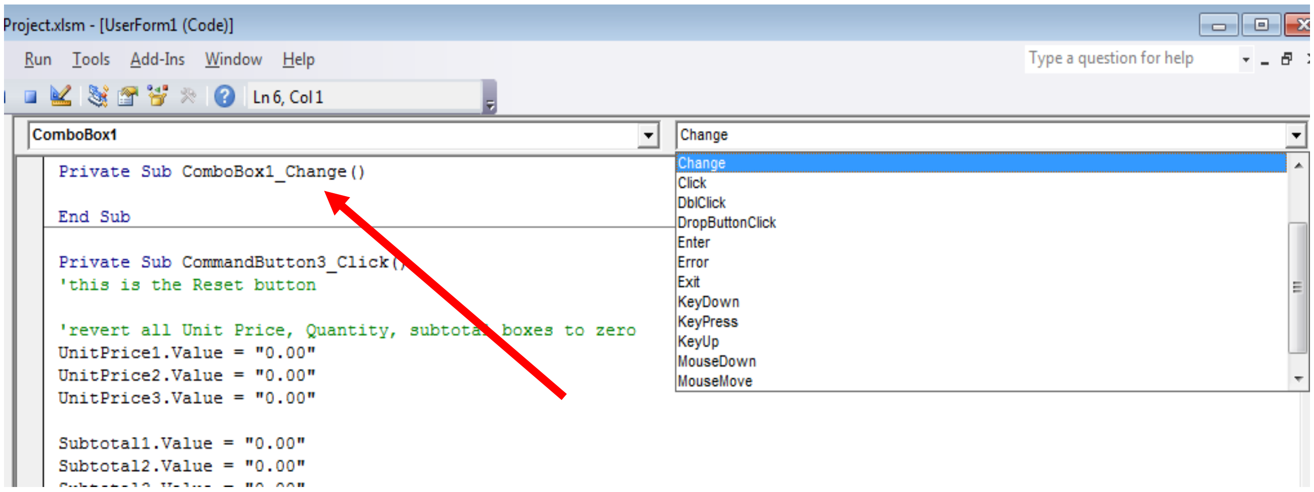
```
Option Explicit


Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price, Quantity, subtotal boxes to zero
UnitPrice1.Value = "0.00"
UnitPrice2.Value = "0.00"
UnitPrice3.Value = "0.00"

Subtotal1.Value = "0.00"
Subtotal2.Value = "0.00"
Subtotal3.Value = "0.00"

Quantity1.Value = "0"
Quantity2.Value = "0"
Quantity3.Value = "0"


'the comboboxes revert to empty
ComboBox1.Value = " "
ComboBox2.Value = " "
ComboBox3.Value = " "

'the Total and grand Total revert to zeros
Total.Value = "0.00"
GrandTotal = "0.00"

'the Units labels revert to underscore
Label31.Caption = "_"
Label32.Caption = "_"
Label33.Caption = "_"

End Sub

Private Sub CommandButton4_Click()
'this is the Exit button

Unload UserForm1
```

The code for the various calculations shall now be added.

Consider *ComboBox1*. If an item is selected, the correct unit price must automatically populate the *UnitPrice1* text box. In ordinary language "if the *ComboBox1* value is changed, put in the applicable unit price for line 1". A procedure can thus be set for the object *ComboBox1* with procedure "Change"

In the object drop down select *ComboBox1*.
In the procedure drop down select **Change**

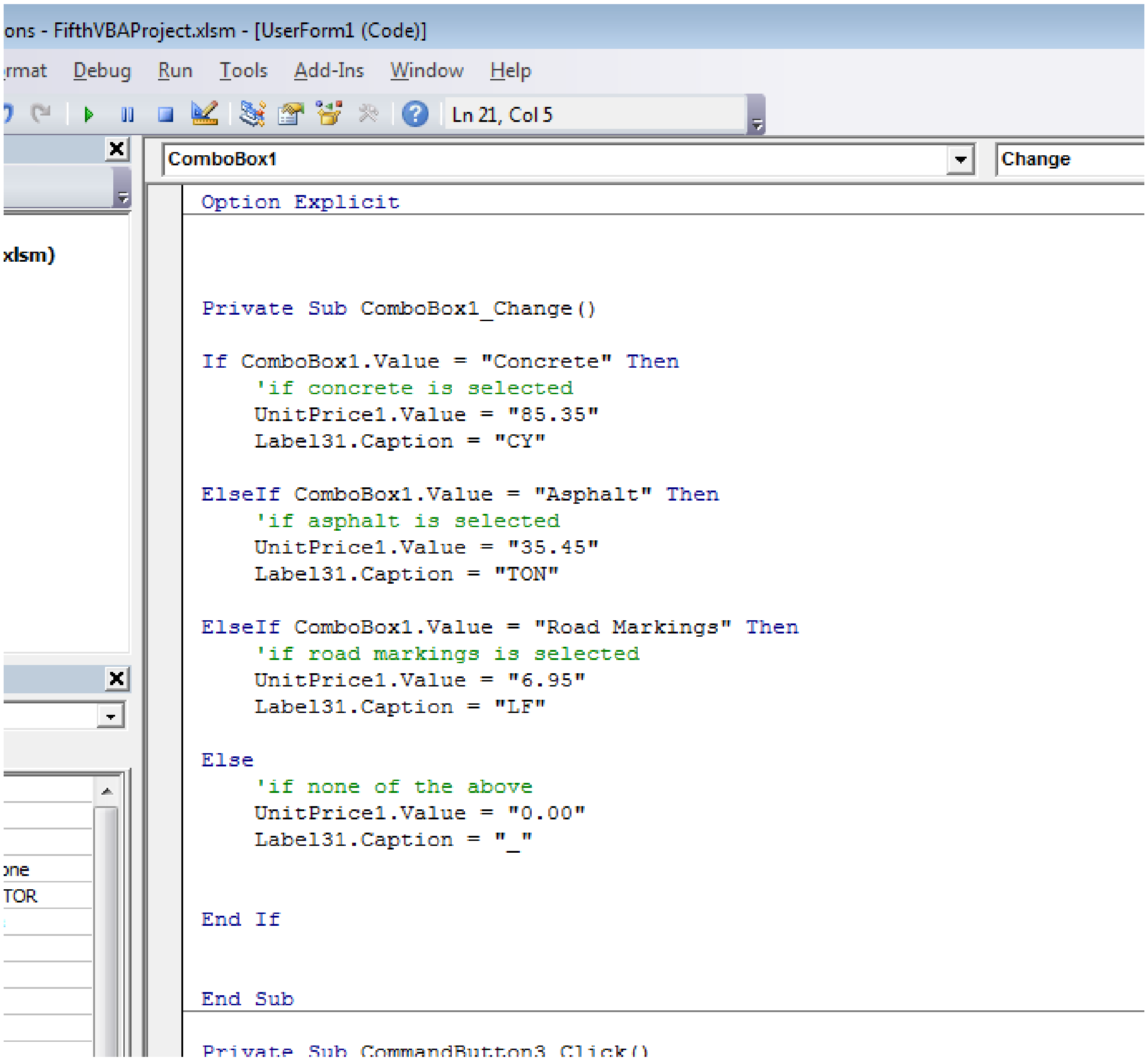
The sub procedure is created. Code must now be added that will execute any time the *ComboBox1* value is changed.



```
Private Sub ComboBox1_Change()

End Sub

Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price, Quantity, subtotal boxes to zero
UnitPrice1.Value = "0.00"
UnitPrice2.Value = "0.00"
UnitPrice3.Value = "0.00"

Subtotal1.Value = "0.00"
Subtotal2.Value = "0.00"
```

Consider the market price of concrete is $85.32 per cubic yard (CY), asphalt is $35.45 per ton (TON), and road marking is $6.95 per linear foot (LF), an *If* statement is needed to display the unit price and the unit of measure based on the item selected in *ComboBox1*.

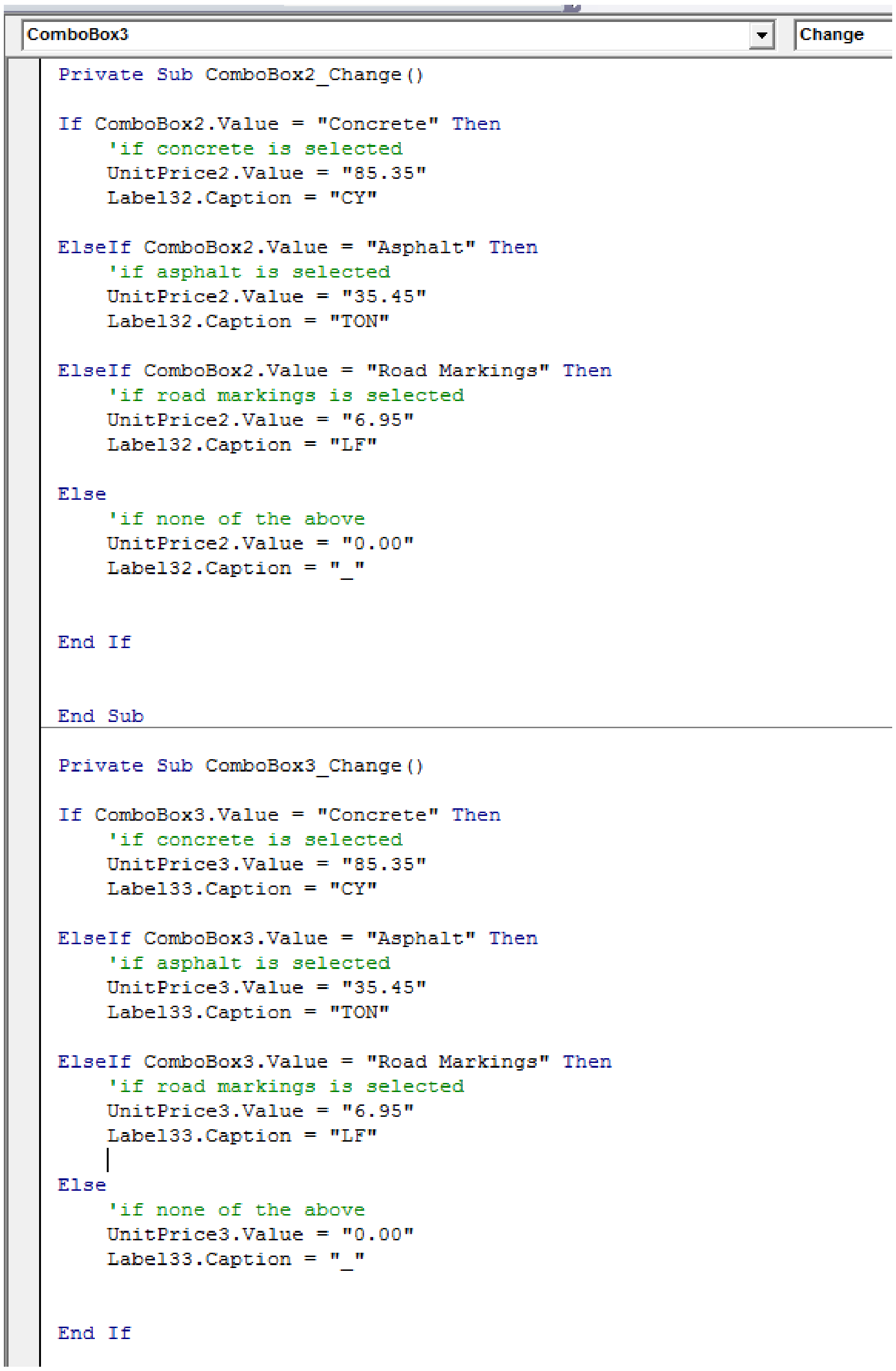
The *If* statement is as follows


ons - FifthVBAProject.xlsm - [UserForm1 (Code)]
rmat Debug Run Tools Add-Ins Window Help
Ln 21, Col 5
ComboBox1
Change
xlsm)
one
TOR

```
Option Explicit


Private Sub ComboBox1_Change()

If ComboBox1.Value = "Concrete" Then
    'if concrete is selected
    UnitPrice1.Value = "85.35"
    Label31.Caption = "CY"

ElseIf ComboBox1.Value = "Asphalt" Then
    'if asphalt is selected
    UnitPrice1.Value = "35.45"
    Label31.Caption = "TON"

ElseIf ComboBox1.Value = "Road Markings" Then
    'if road markings is selected
    UnitPrice1.Value = "6.95"
    Label31.Caption = "LF"

Else
    'if none of the above
    UnitPrice1.Value = "0.00"
    Label31.Caption = "_"


End If


End Sub

Private Sub CommandButton3_Click()
```

Save the workbook.
Click on Debug. Select Compile VBAProject
Correct any errors.
Resave.

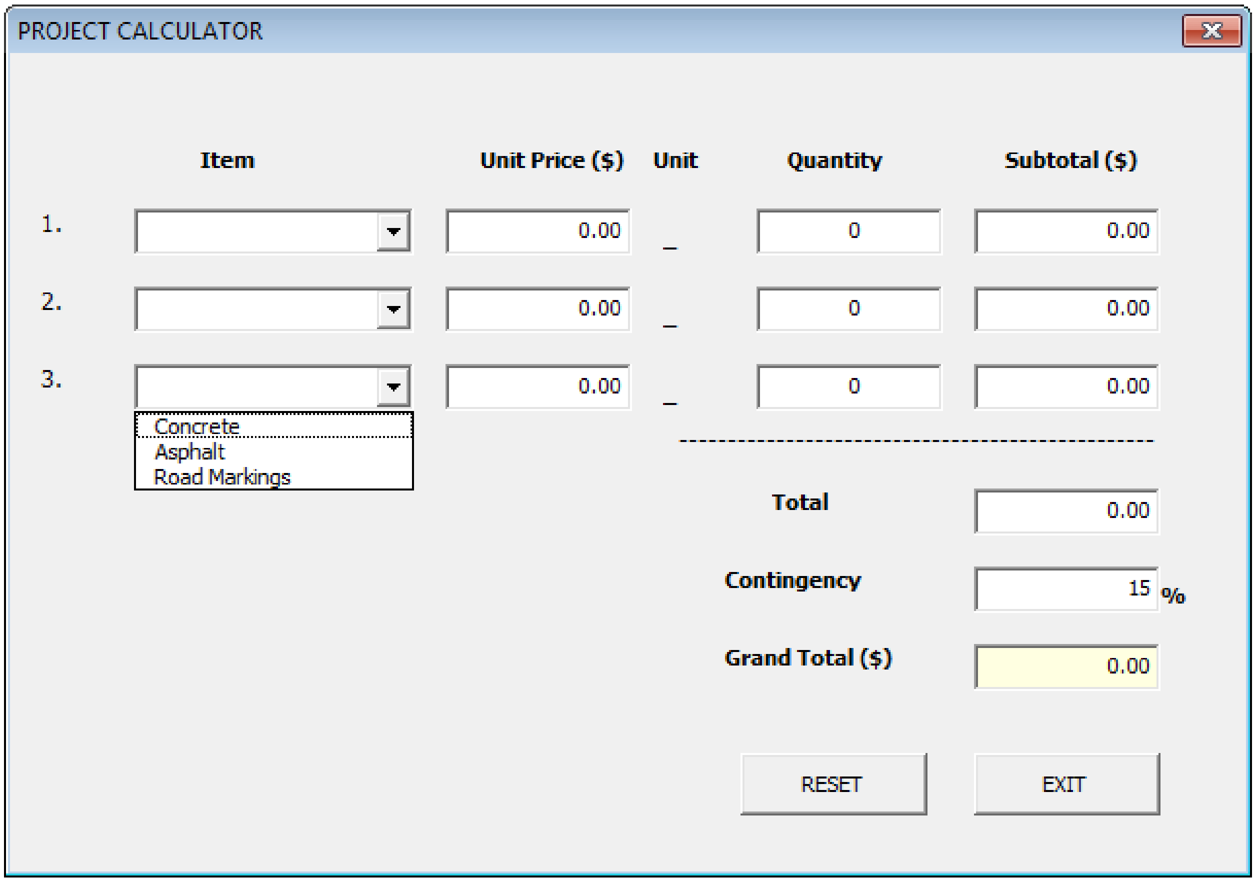
If there are no errors, repeat the process for the code for *ComboBox2* and *ComboBox3*, as follows



```
Private Sub ComboBox2_Change()

If ComboBox2.Value = "Concrete" Then
    'if concrete is selected
    UnitPrice2.Value = "85.35"
    Label32.Caption = "CY"

ElseIf ComboBox2.Value = "Asphalt" Then
    'if asphalt is selected
    UnitPrice2.Value = "35.45"
    Label32.Caption = "TON"

ElseIf ComboBox2.Value = "Road Markings" Then
    'if road markings is selected
    UnitPrice2.Value = "6.95"
    Label32.Caption = "LF"

Else
    'if none of the above
    UnitPrice2.Value = "0.00"
    Label32.Caption = "_"


End If


End Sub

Private Sub ComboBox3_Change()

If ComboBox3.Value = "Concrete" Then
    'if concrete is selected
    UnitPrice3.Value = "85.35"
    Label33.Caption = "CY"

ElseIf ComboBox3.Value = "Asphalt" Then
    'if asphalt is selected
    UnitPrice3.Value = "35.45"
    Label33.Caption = "TON"

ElseIf ComboBox3.Value = "Road Markings" Then
    'if road markings is selected
    UnitPrice3.Value = "6.95"
    Label33.Caption = "LF"

Else
    'if none of the above
    UnitPrice3.Value = "0.00"
    Label33.Caption = "_"


End If
```

Save the workbook.
Click on Debug. Select Compile VBAProject.
Correct any errors.
If there are no errors, resave.
Test the application.

Open the application.
Select an item from the each dropdown list.



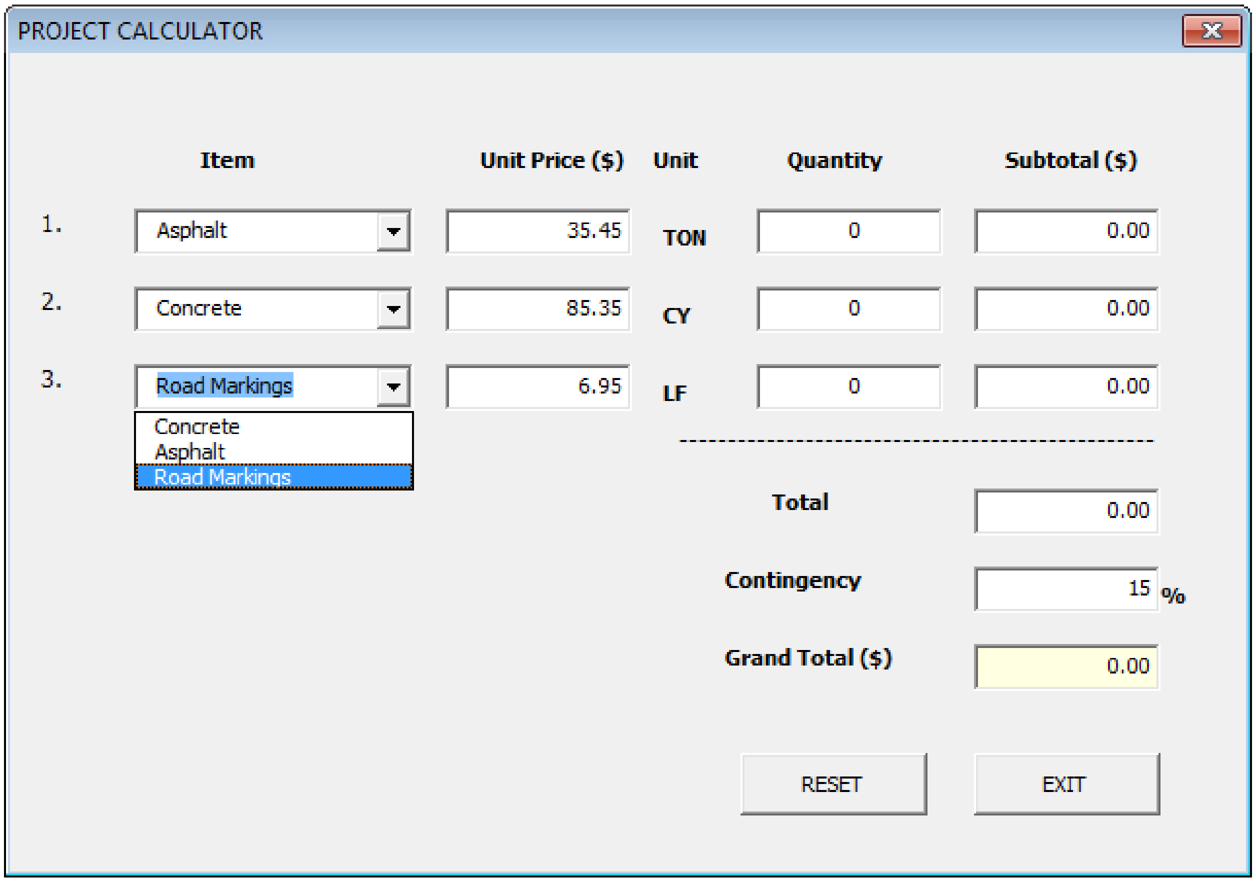
PROJECT CALCULATOR

	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.		0.00	_	0	0.00
2.		0.00	_	0	0.00
3.		0.00	_	0	0.00

Concrete
Asphalt
Road Markings

Total	0.00
Contingency	15 %
Grand Total ($)	0.00

Confirm that the unit prices and units populate correctly as intended.


PROJECT CALCULATOR

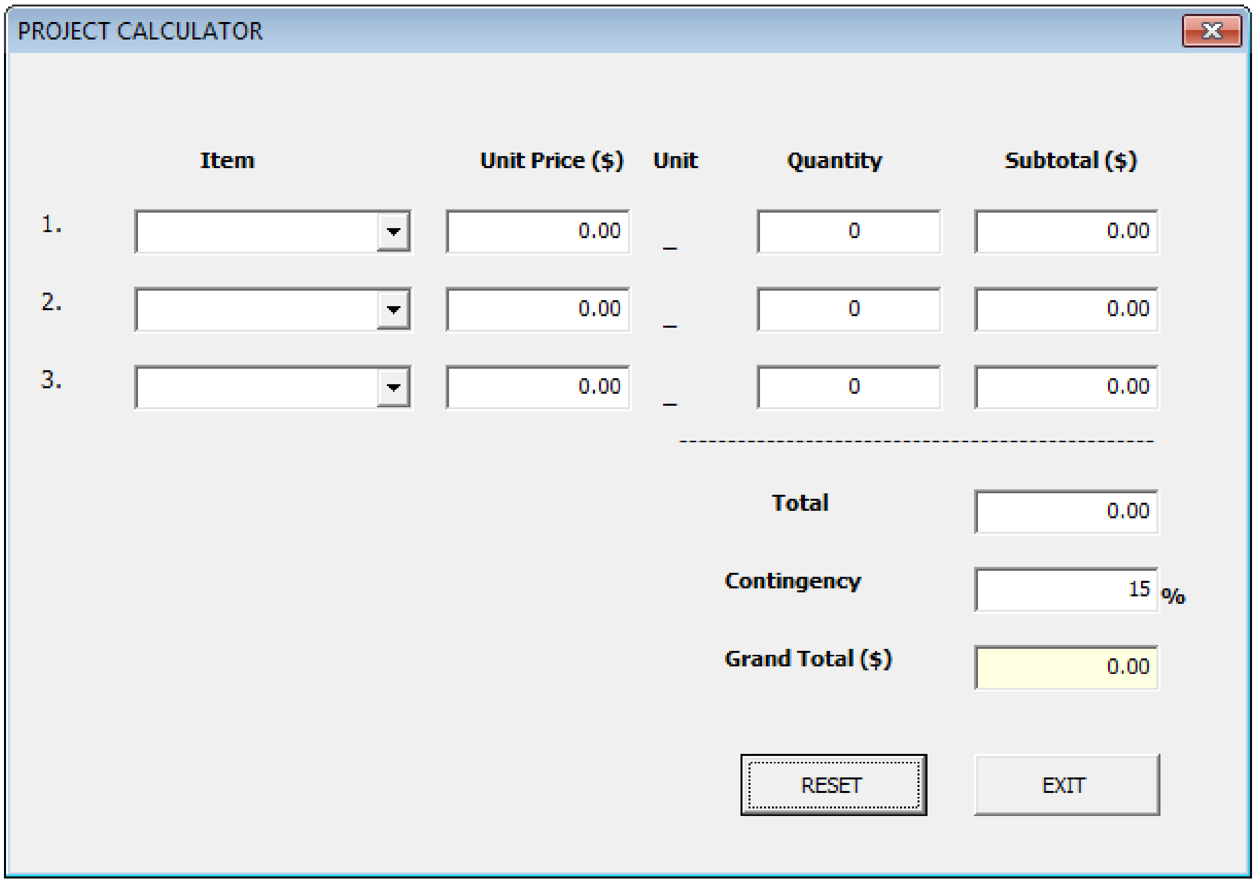
	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.	Asphalt	35.45	TON	0	0.00
2.	Concrete	85.35	CY	0	0.00
3.	Road Markings	6.95	LF	0	0.00

Concrete
Asphalt
Road Markings

Total	0.00
Contingency	15 %
Grand Total ($)	0.00

RESET
EXIT

Click on the *Reset* button to clear the form.
The form reverts to its original entries.


PROJECT CALCULATOR

	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.		0.00	_	0	0.00
2.		0.00	_	0	0.00
3.		0.00	_	0	0.00

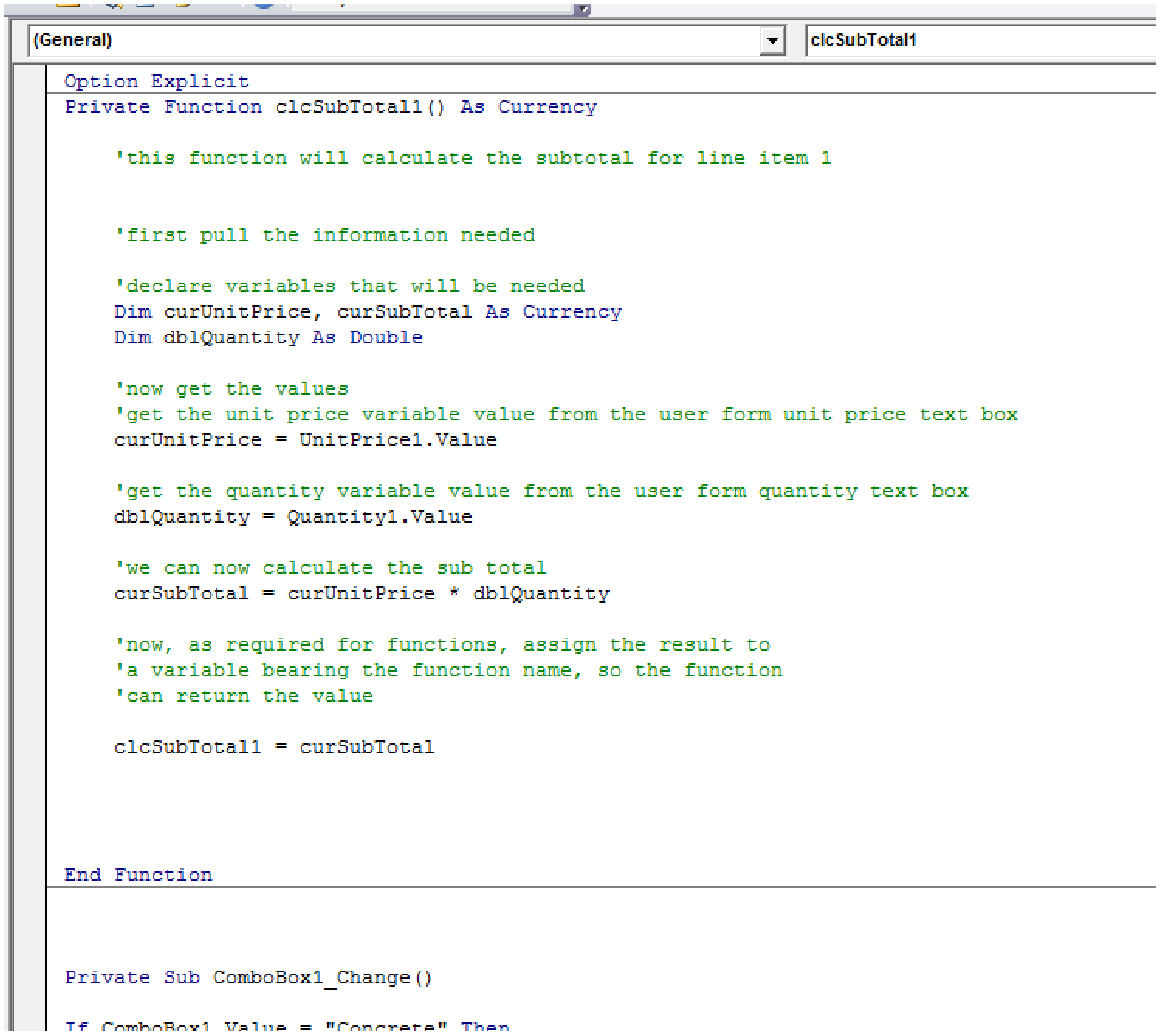
Total 0.00

Contingency 15 %

Grand Total ($) 0.00

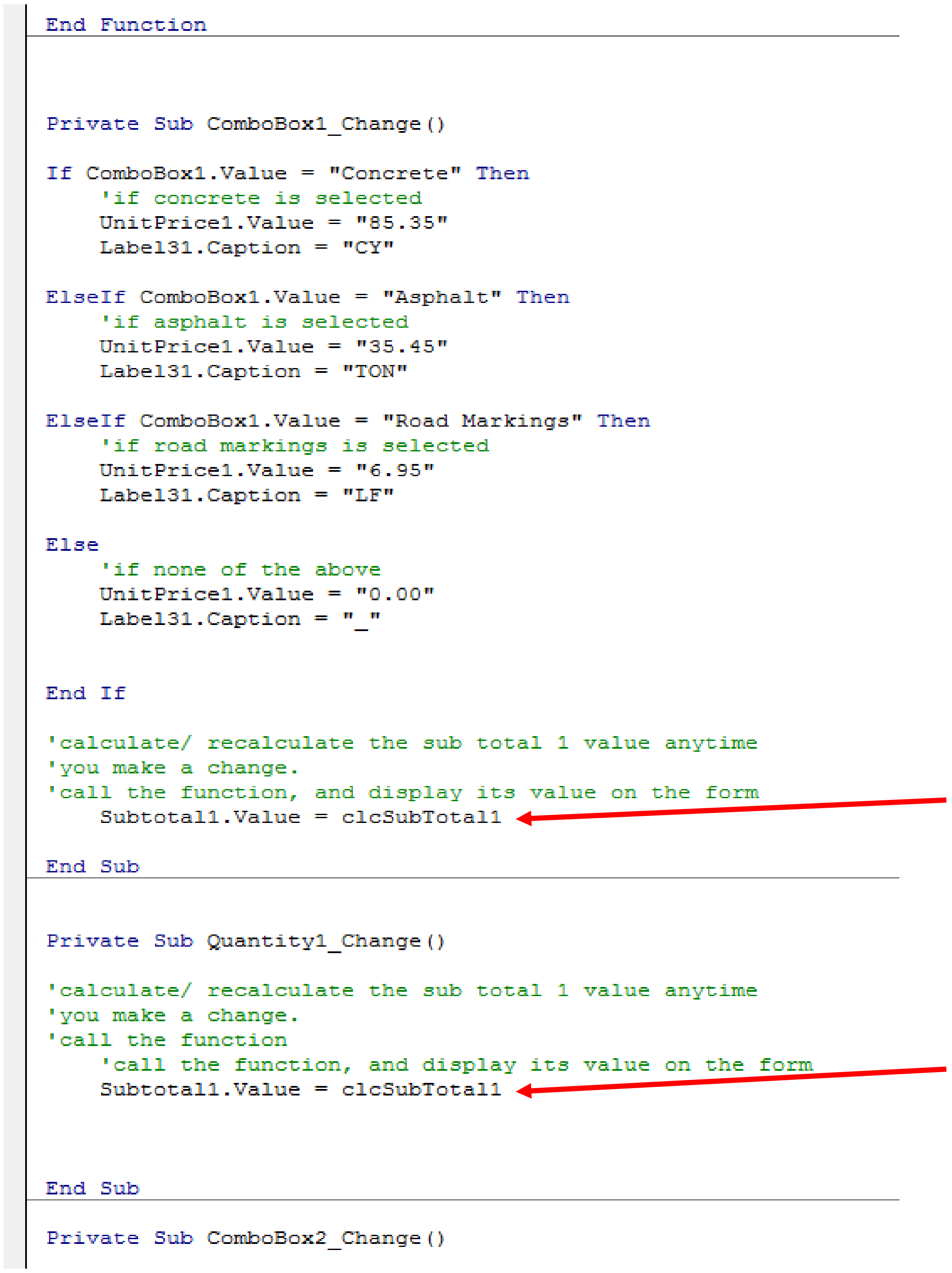
Click on *Exit* to close the application and return to design mode.

Next, calculate the subtotal for the first line item. The subtotal is the unit price multiplied by the quantity. A function shall be developed to do this calculation.



```
Option Explicit
Private Function clcSubTotal1() As Currency

    'this function will calculate the subtotal for line item 1


    'first pull the information needed

    'declare variables that will be needed
    Dim curUnitPrice, curSubTotal As Currency
    Dim dblQuantity As Double

    'now get the values
    'get the unit price variable value from the user form unit price text box
    curUnitPrice = UnitPrice1.Value

    'get the quantity variable value from the user form quantity text box
    dblQuantity = Quantity1.Value

    'we can now calculate the sub total
    curSubTotal = curUnitPrice * dblQuantity

    'now, as required for functions, assign the result to
    'a variable bearing the function name, so the function
    'can return the value

    clcSubTotal1 = curSubTotal




End Function



Private Sub ComboBox1_Change()

If ComboBox1.Value = "Concrete" Then
```

The subtotal calculation function shall "fire' anytime the *UnitPrice1* control is changed and anytime the *Quantity1* value is changed. Insert the function call within these procedures.

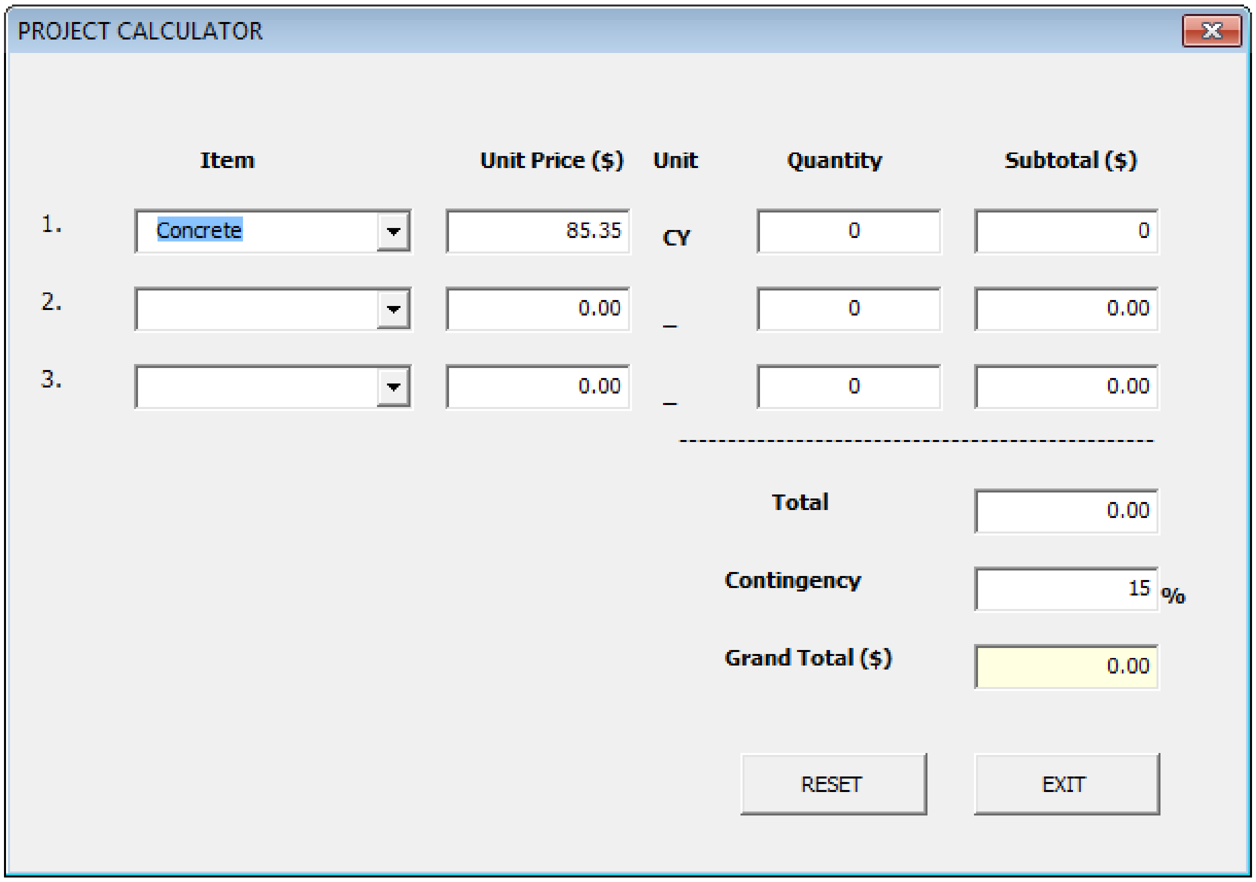


```
End Function


Private Sub ComboBox1_Change()

If ComboBox1.Value = "Concrete" Then
    'if concrete is selected
    UnitPrice1.Value = "85.35"
    Label31.Caption = "CY"

ElseIf ComboBox1.Value = "Asphalt" Then
    'if asphalt is selected
    UnitPrice1.Value = "35.45"
    Label31.Caption = "TON"

ElseIf ComboBox1.Value = "Road Markings" Then
    'if road markings is selected
    UnitPrice1.Value = "6.95"
    Label31.Caption = "LF"

Else
    'if none of the above
    UnitPrice1.Value = "0.00"
    Label31.Caption = "_"


End If

'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function, and display its value on the form
    Subtotal1.Value = clcSubTotal1

End Sub


Private Sub Quantity1_Change()

'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function
    'call the function, and display its value on the form
    Subtotal1.Value = clcSubTotal1



End Sub

Private Sub ComboBox2_Change()
```

Note that there was no declaration for *clcSubTotal1* in either *Quantity1_Change( )* or *ComboBox1_Change( )*. The compiler recognizes it as a function procedure and "goes to look for it" in the code. Once found, it executes it and the assigns the value calculated therein to a variable of the same name. This variable is now available for use anywhere in the module by simply calling it.

Save the workbook.
Click on Debug. Select Compile VBAProject
Correct any errors.
Resave.
If there are no errors, test the application.

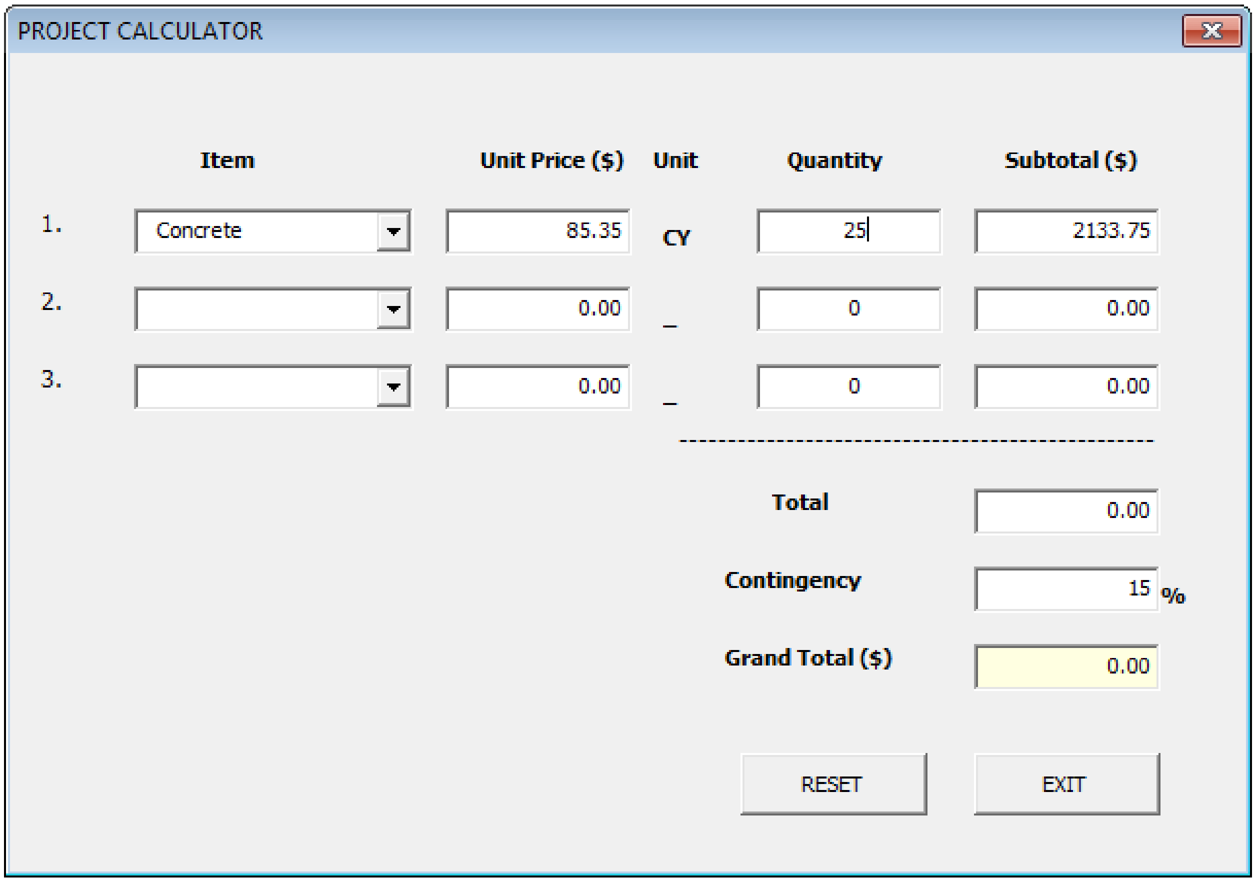
Open the application and select an item on line 1.



	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.	Concrete	85.35	CY	0	0
2.		0.00	_	0	0.00
3.		0.00	_	0	0.00

Total 0.00
Contingency 15 %
Grand Total ($) 0.00

Enter a quantity.

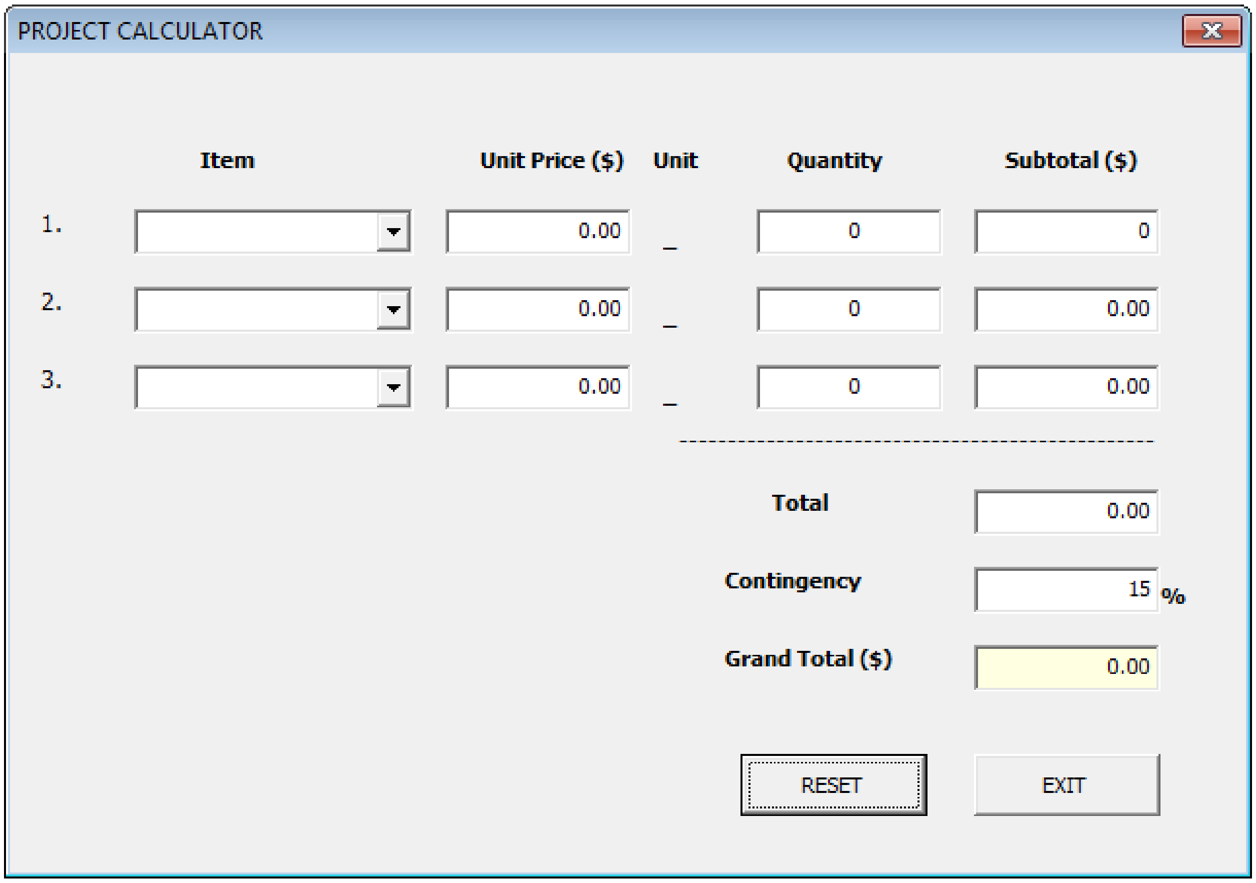

PROJECT CALCULATOR

	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.	Concrete	85.35	CY	25	2133.75
2.		0.00	_	0	0.00
3.		0.00	_	0	0.00

Total	0.00
Contingency	15 %
Grand Total ($)	0.00

The subtotal is automatically calculated as expected.

Click on *Reset* to check that all controls will clear as intended.



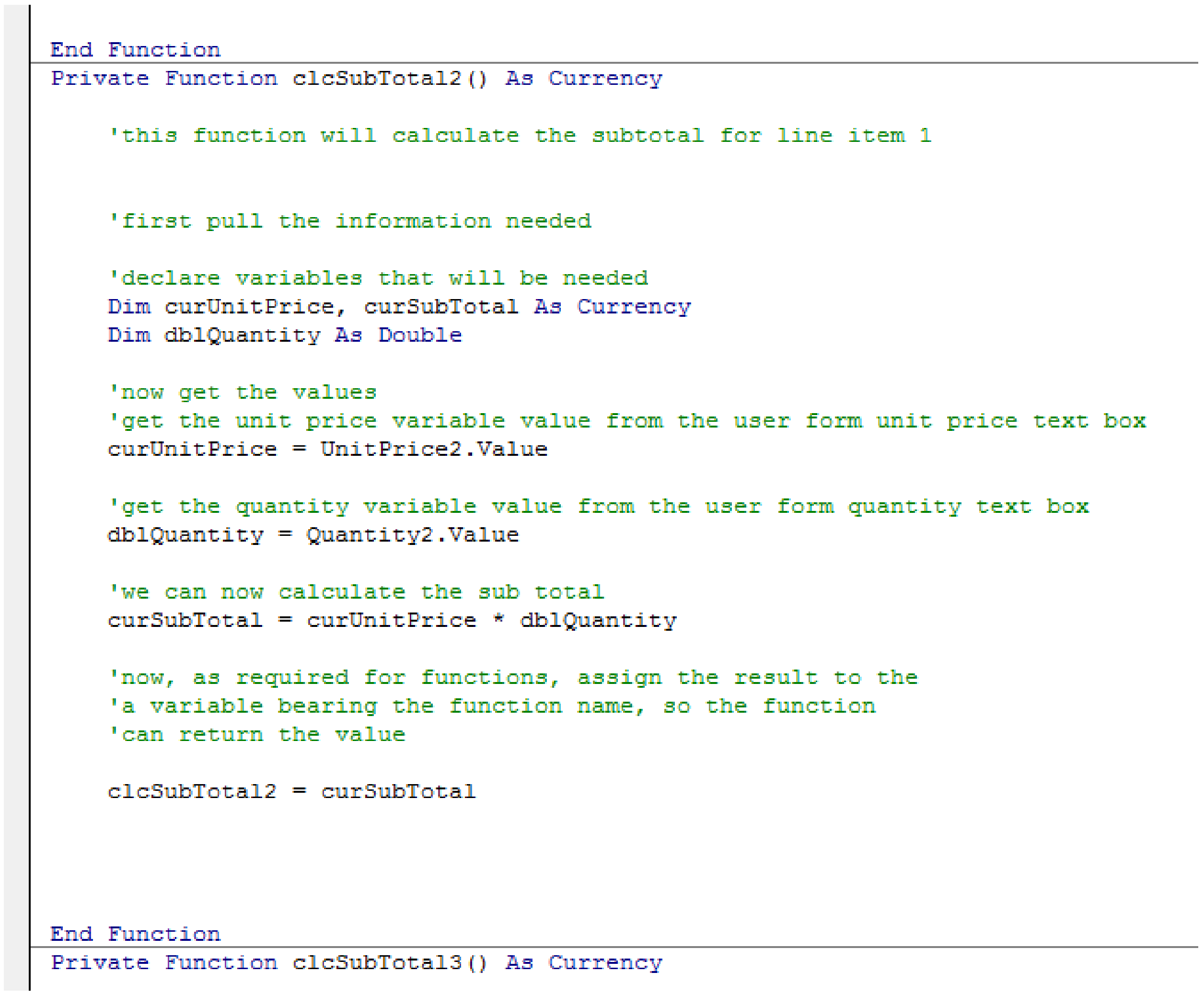
	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.		0.00	_	0	0
2.		0.00	_	0	0.00
3.		0.00	_	0	0.00

Total	0.00
Contingency	15 %
Grand Total ($)	0.00

The test is a success.

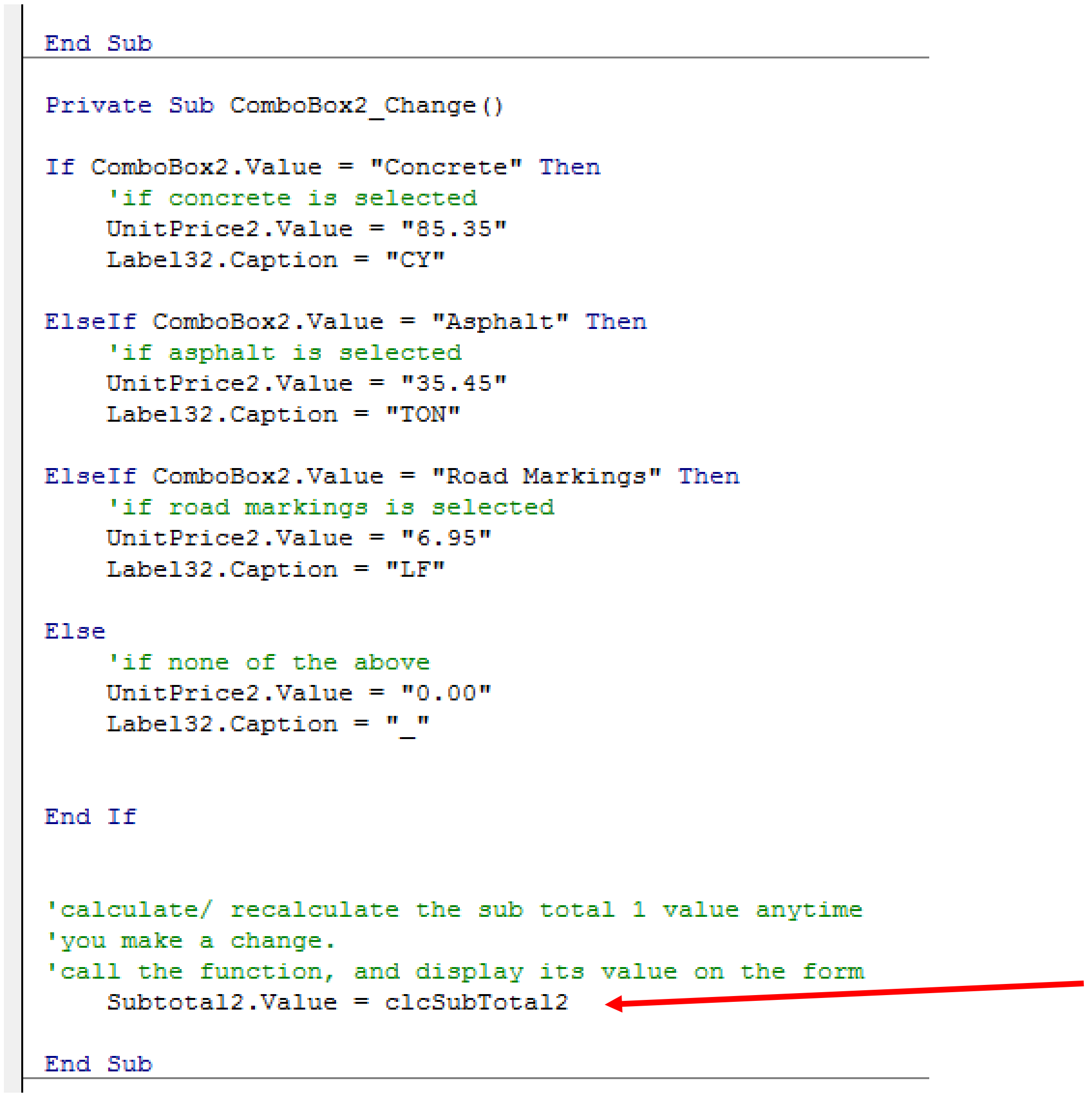
Add functions for the subtotals of the other line items.

The function for the calculation of the sub total for the second line item is as follows



```
End Function
Private Function clcSubTotal2() As Currency

    'this function will calculate the subtotal for line item 1


    'first pull the information needed

    'declare variables that will be needed
    Dim curUnitPrice, curSubTotal As Currency
    Dim dblQuantity As Double

    'now get the values
    'get the unit price variable value from the user form unit price text box
    curUnitPrice = UnitPrice2.Value

    'get the quantity variable value from the user form quantity text box
    dblQuantity = Quantity2.Value

    'we can now calculate the sub total
    curSubTotal = curUnitPrice * dblQuantity

    'now, as required for functions, assign the result to the
    'a variable bearing the function name, so the function
    'can return the value

    clcSubTotal2 = curSubTotal



End Function
Private Function clcSubTotal3() As Currency
```

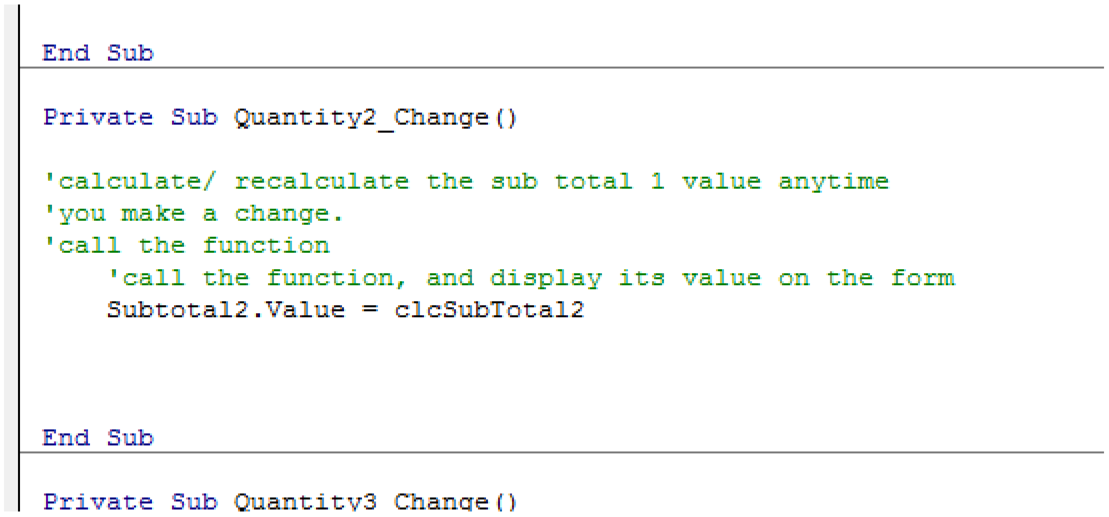
The function to calculate the subtotal for the second line item is embedded in the quantity and unit rate change procedures for the second line item as follows



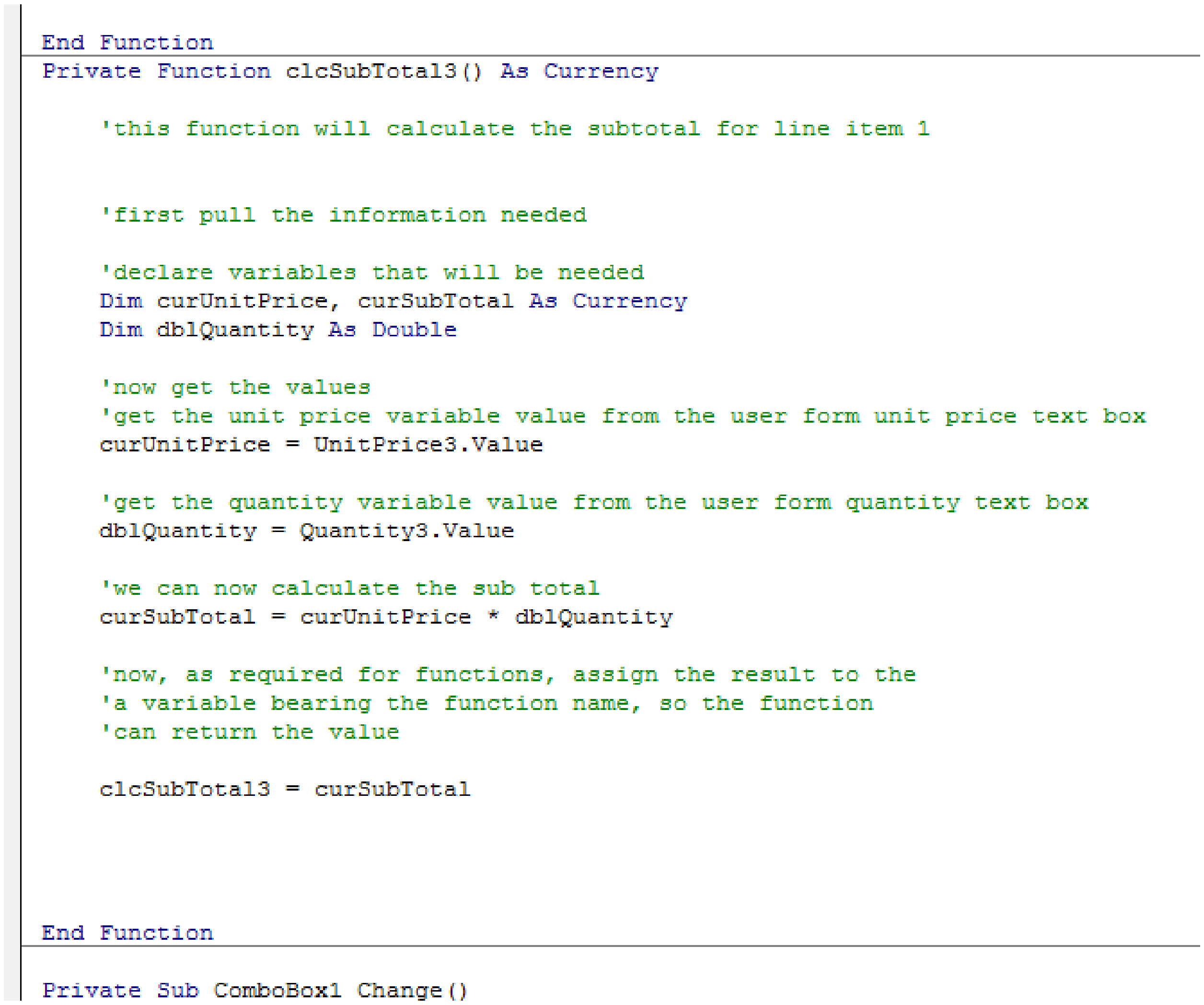
```
End Sub

Private Sub ComboBox2_Change()

If ComboBox2.Value = "Concrete" Then
    'if concrete is selected
    UnitPrice2.Value = "85.35"
    Label32.Caption = "CY"

ElseIf ComboBox2.Value = "Asphalt" Then
    'if asphalt is selected
    UnitPrice2.Value = "35.45"
    Label32.Caption = "TON"

ElseIf ComboBox2.Value = "Road Markings" Then
    'if road markings is selected
    UnitPrice2.Value = "6.95"
    Label32.Caption = "LF"

Else
    'if none of the above
    UnitPrice2.Value = "0.00"
    Label32.Caption = "_"


End If


'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function, and display its value on the form
    Subtotal2.Value = clcSubTotal2

End Sub
```

.

and

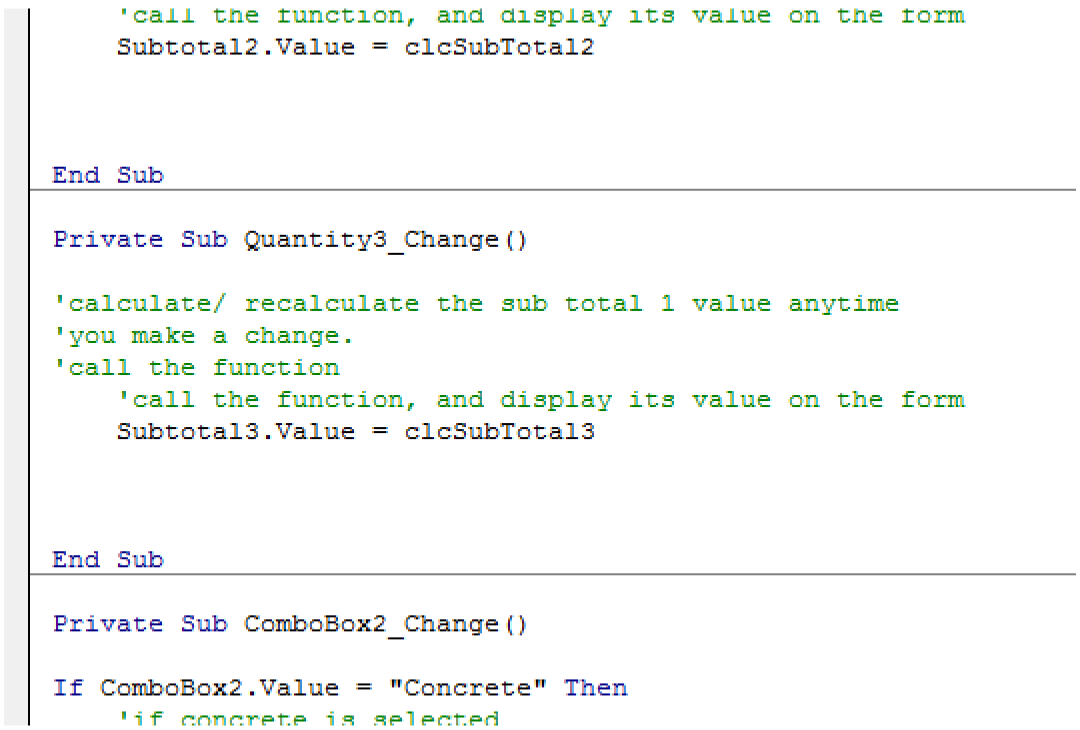

```
End Sub

Private Sub Quantity2_Change()

'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function
    'call the function, and display its value on the form
    Subtotal2.Value = clcSubTotal2



End Sub

Private Sub Quantity3_Change()
```

The function to calculate the subtotal for the third line item is embedded in the quantity and unit price change procedures for the third line item as follows



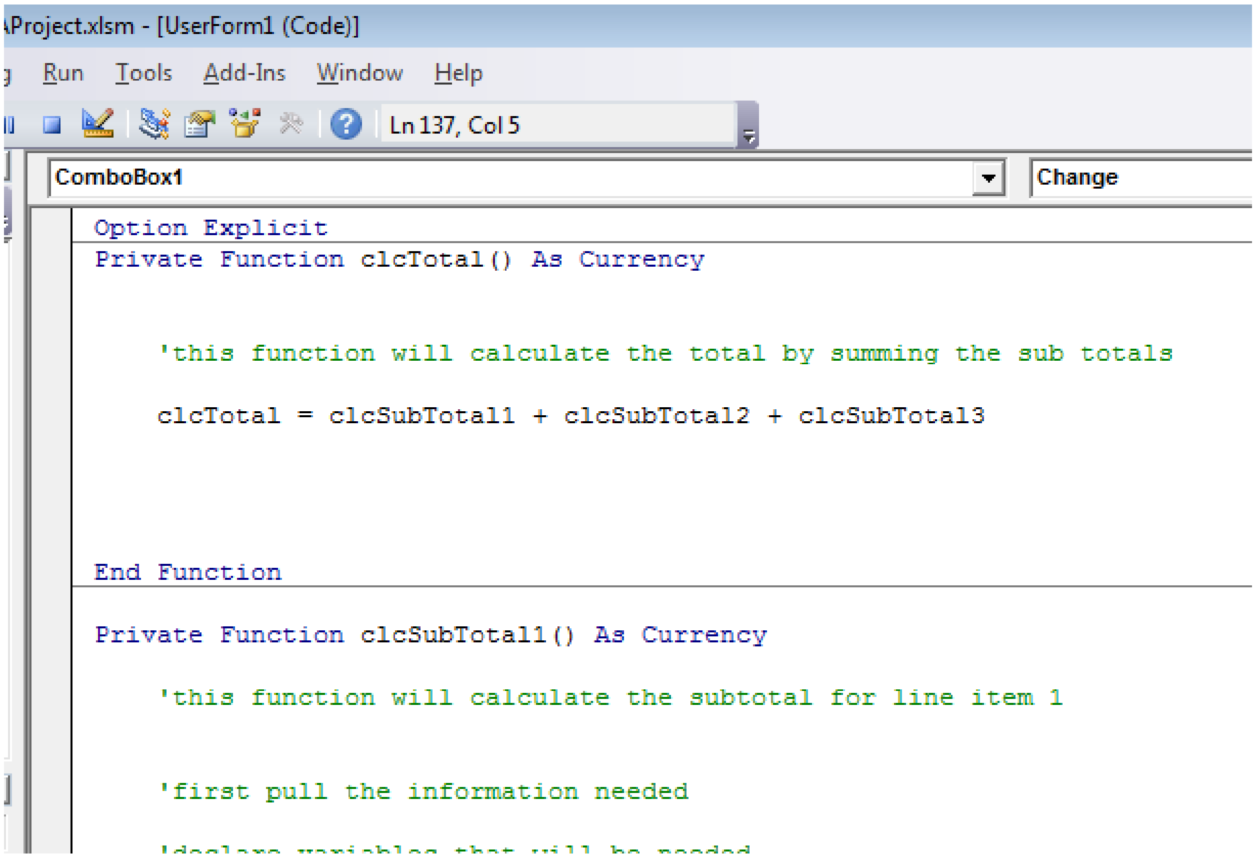
```
End Function
Private Function clcSubTotal3() As Currency

    'this function will calculate the subtotal for line item 1


    'first pull the information needed

    'declare variables that will be needed
    Dim curUnitPrice, curSubTotal As Currency
    Dim dblQuantity As Double

    'now get the values
    'get the unit price variable value from the user form unit price text box
    curUnitPrice = UnitPrice3.Value

    'get the quantity variable value from the user form quantity text box
    dblQuantity = Quantity3.Value

    'we can now calculate the sub total
    curSubTotal = curUnitPrice * dblQuantity

    'now, as required for functions, assign the result to the
    'a variable bearing the function name, so the function
    'can return the value

    clcSubTotal3 = curSubTotal




End Function

Private Sub ComboBox1_Change()
```

and



```
    'call the function, and display its value on the form
    Subtotal2.Value = clcSubTotal2



End Sub
________________________________________________________________

Private Sub Quantity3_Change()

'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function
    'call the function, and display its value on the form
    Subtotal3.Value = clcSubTotal3



End Sub
________________________________________________________________

Private Sub ComboBox2_Change()

If ComboBox2.Value = "Concrete" Then
    'if concrete is selected
```

The total, being the summation of the subtotals will be calculated using a function.


Project.xlsm - [UserForm1 (Code)]
Run Tools Add-Ins Window Help
Ln 137, Col 5
ComboBox1
Change

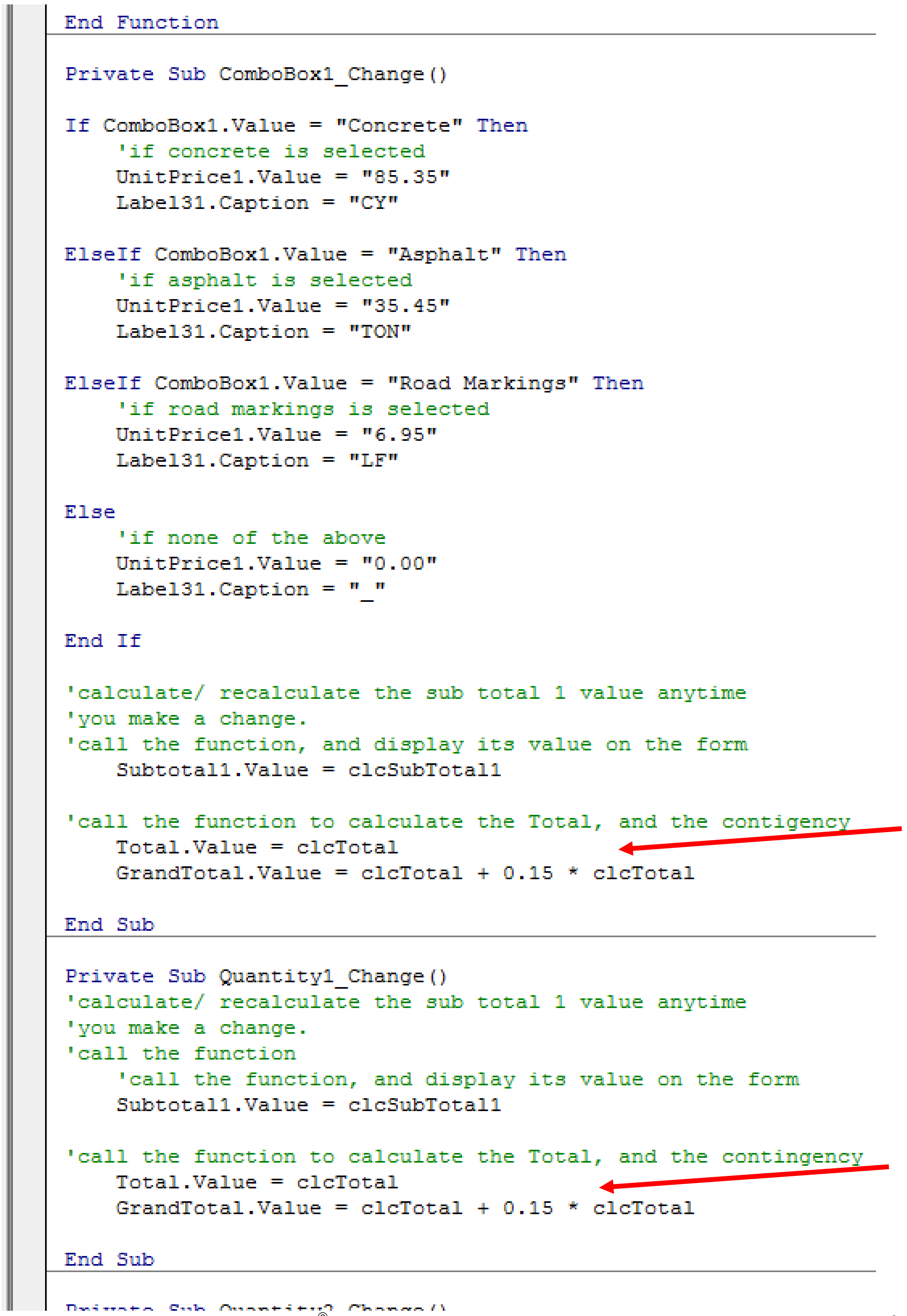
```
Option Explicit
Private Function clcTotal() As Currency


    'this function will calculate the total by summing the sub totals

    clcTotal = clcSubTotal1 + clcSubTotal2 + clcSubTotal3




End Function

Private Function clcSubTotal1() As Currency

    'this function will calculate the subtotal for line item 1


    'first pull the information needed
```

This function will be called to update the total anytime a unit price or quantity for any line item is changed. This means the *clcTotal* calculation function will be embedded in all six change procedures.

The contingency is 15% of the total amount. It may be calculated by calling the *clcTotal* function within each unit price and quantity change procedure and multiplying it by the percentage rate.

The grand total is the sum of total and contingency. It may be calculated by calling the *clcTotal* function within each unit price and quantity change procedure and adding the contingency amount.

The calls for the first line item are as follows



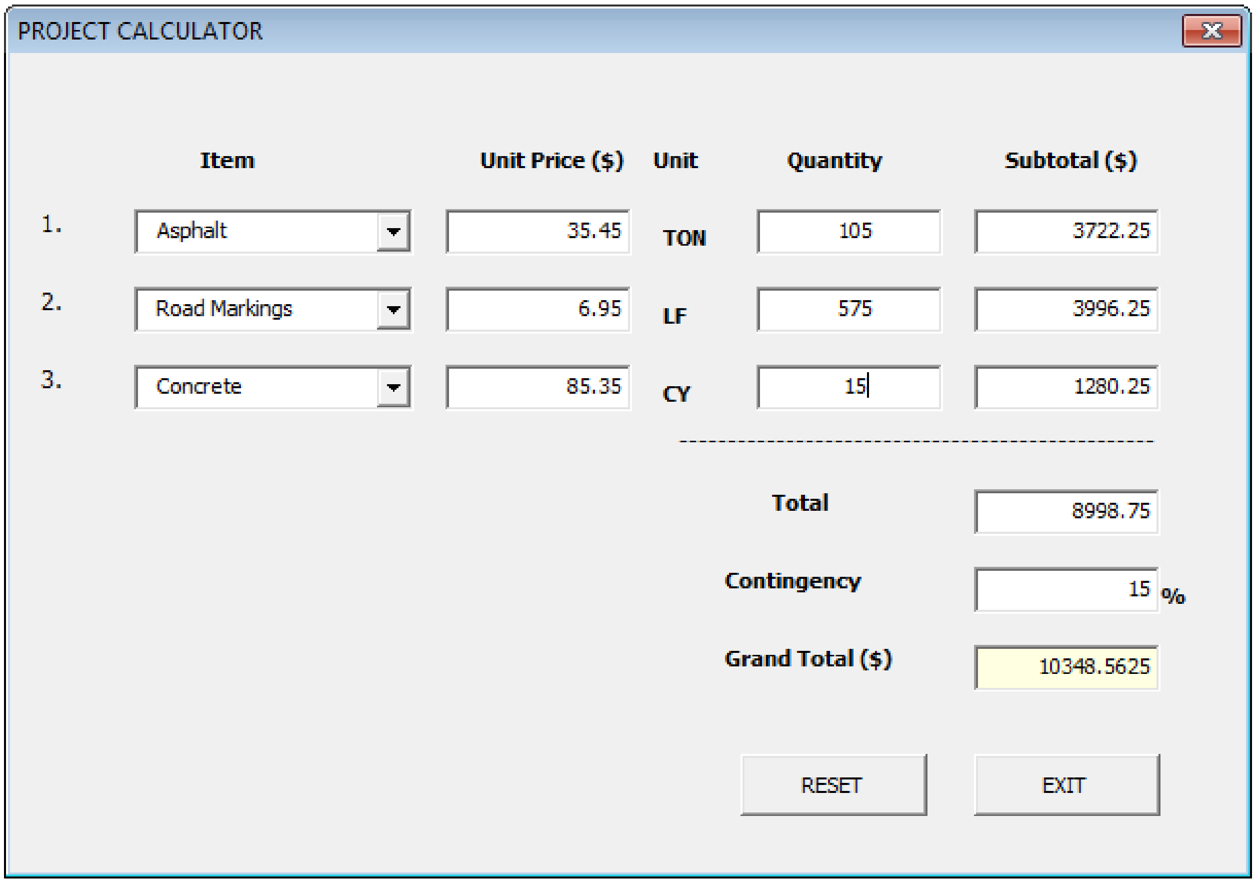
```
End Function

Private Sub ComboBox1_Change()

If ComboBox1.Value = "Concrete" Then
    'if concrete is selected
    UnitPrice1.Value = "85.35"
    Label31.Caption = "CY"

ElseIf ComboBox1.Value = "Asphalt" Then
    'if asphalt is selected
    UnitPrice1.Value = "35.45"
    Label31.Caption = "TON"

ElseIf ComboBox1.Value = "Road Markings" Then
    'if road markings is selected
    UnitPrice1.Value = "6.95"
    Label31.Caption = "LF"

Else
    'if none of the above
    UnitPrice1.Value = "0.00"
    Label31.Caption = "_"

End If

'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function, and display its value on the form
    Subtotal1.Value = clcSubTotal1

'call the function to calculate the Total, and the contigency
    Total.Value = clcTotal
    GrandTotal.Value = clcTotal + 0.15 * clcTotal

End Sub

Private Sub Quantity1_Change()
'calculate/ recalculate the sub total 1 value anytime
'you make a change.
'call the function
    'call the function, and display its value on the form
    Subtotal1.Value = clcSubTotal1

'call the function to calculate the Total, and the contingency
    Total.Value = clcTotal
    GrandTotal.Value = clcTotal + 0.15 * clcTotal

End Sub

Private Sub Quantity2_Change()
```

Repeat these *clcTotal* function calls for the second and third line items.

Save the workbook.
Click on Debug. Select Compile VBAProject
Correct any errors.
Resave.

If there are no errors, test the application.

Open the application and fill out the line items.


PROJECT CALCULATOR

	Item	Unit Price ($)	Unit	Quantity	Subtotal ($)
1.	Asphalt	35.45	TON	105	3722.25
2.	Road Markings	6.95	LF	575	3996.25
3.	Concrete	85.35	CY	15	1280.25

Total	8998.75
Contingency	15 %
Grand Total ($)	10348.5625

The subtotals, total and grand total are calculated as expected.

The test is a success.
Close out of the application.
Save the workbook.
Close out of Excel VBA.

2. WORKING WITH STRINGS AND DATES

A **string** is a series of characters. The string data type is used to store and manipulate text. The **date** data type is used to support dates and times, as well as date and time-based operations. VBA and the Excel library provide a wide variety of functions for strings and dates. A selection of string and date manipulations is presented in this chapter.

2.1 String Variables

To assign a value to a string variable, the value is put in double quotation marks. For example

```
Private Sub StringExample( )

        Dim strFirstname, strLastname As String

        strFirstname = “Sam”
        strLastname = “Johnson”

End Sub
```

2.2 Concatenation

String concatenation is the operation of “welding” strings together. The operator used is the plus sign (+), or the ampersand sign (&). For example

```
Private Sub StringExample( )

        Dim strFirstname, strLastname, strFullname As String

        strFirstname = “Sam”
        strLastname = “Johnson”
        strFullname = strFirstname & strLastname

End Sub
```

The above code will yield the full name as *"SamJohnson"*. To add a space between the first name and the last name, a string with the space character needs to be added. For example

```
Private Sub StringExample( )

        Dim strFirstname, strLastname, strFullname As String

        strFirstname = "Sam"
        strLastname = "Johnson"

        strFullname = strFirstname & " " & strLastname

End Sub
```

This code would yield the full name as "*Sam Johnson*". In many official documents, the last name is written first with a comma and a space separating the first name. This can be implemented as follows

```
Private Sub StringExample( )

        Dim strFirstname, strLastname, strFullname As String

        strFirstname = "Sam"
        strLastname = "Johnson"

        strFullname = strLasttname & ", " & strFirstname

End Sub
```

The above code will yield the full name as *"Johnson, Sam"*.

2.3 String Length

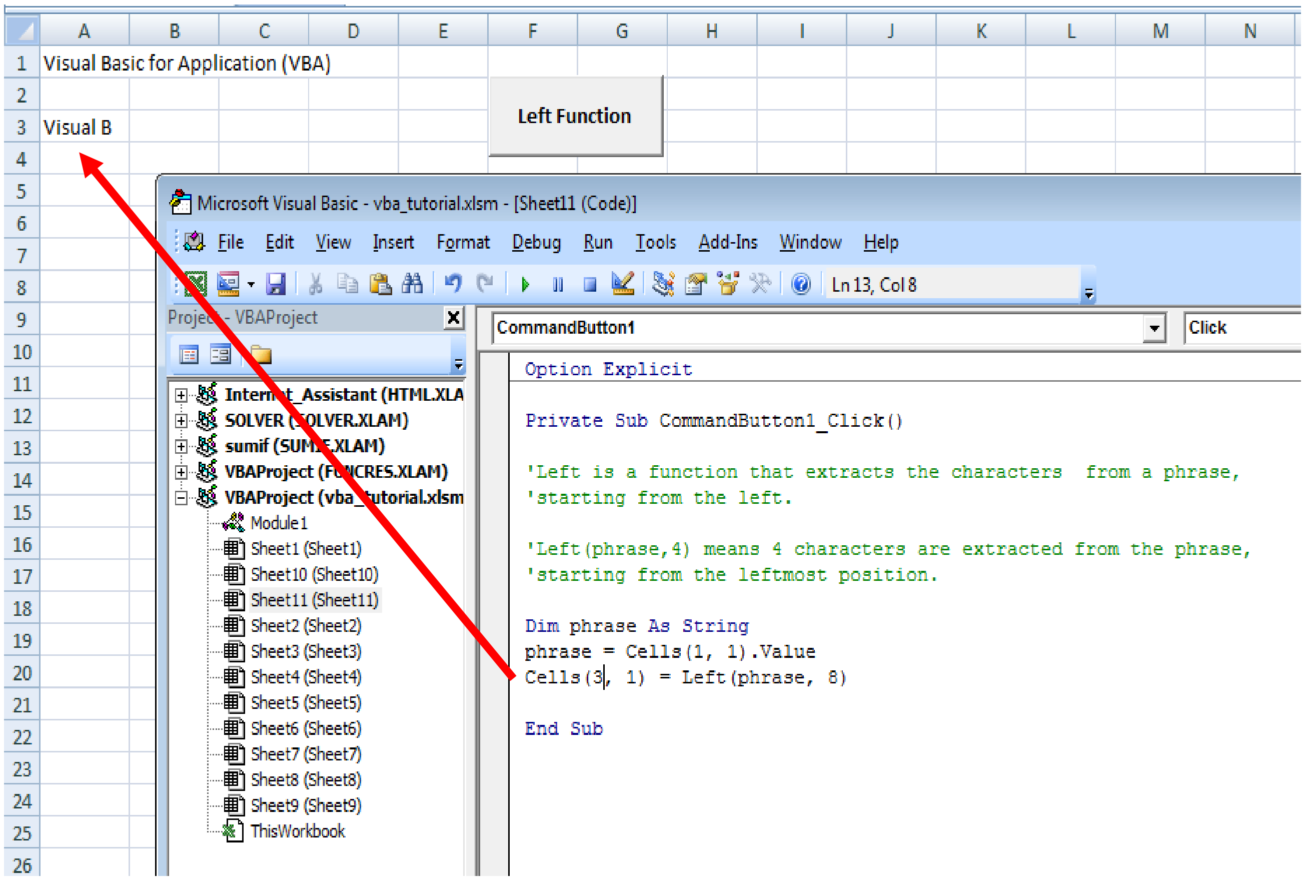
The length of a string is the number of characters it has. The VBA built-in function *Len* can be used to find the length of a string. For example

```
Private Sub StringExample( )

        Dim strFirstname, strLastname, strFullname As String
        Dim intStrinLeng As Integer

        strFirstname = "Sam"
        strLastname = "Johnson"

        strFullname = strLasttname & ", " & strFirstname

        intStrinLeng = Len(strFullname)
        'or intStrinLeng = Len("Johnson, Sam")

End Sub
```

The value of the *intStrinLeng* integer variable will be 12, of which 7 came from the last name, 3 from the first name, plus 1 comma character, and 1 space character in the last-name-first-name separator.

2.4 Left Function

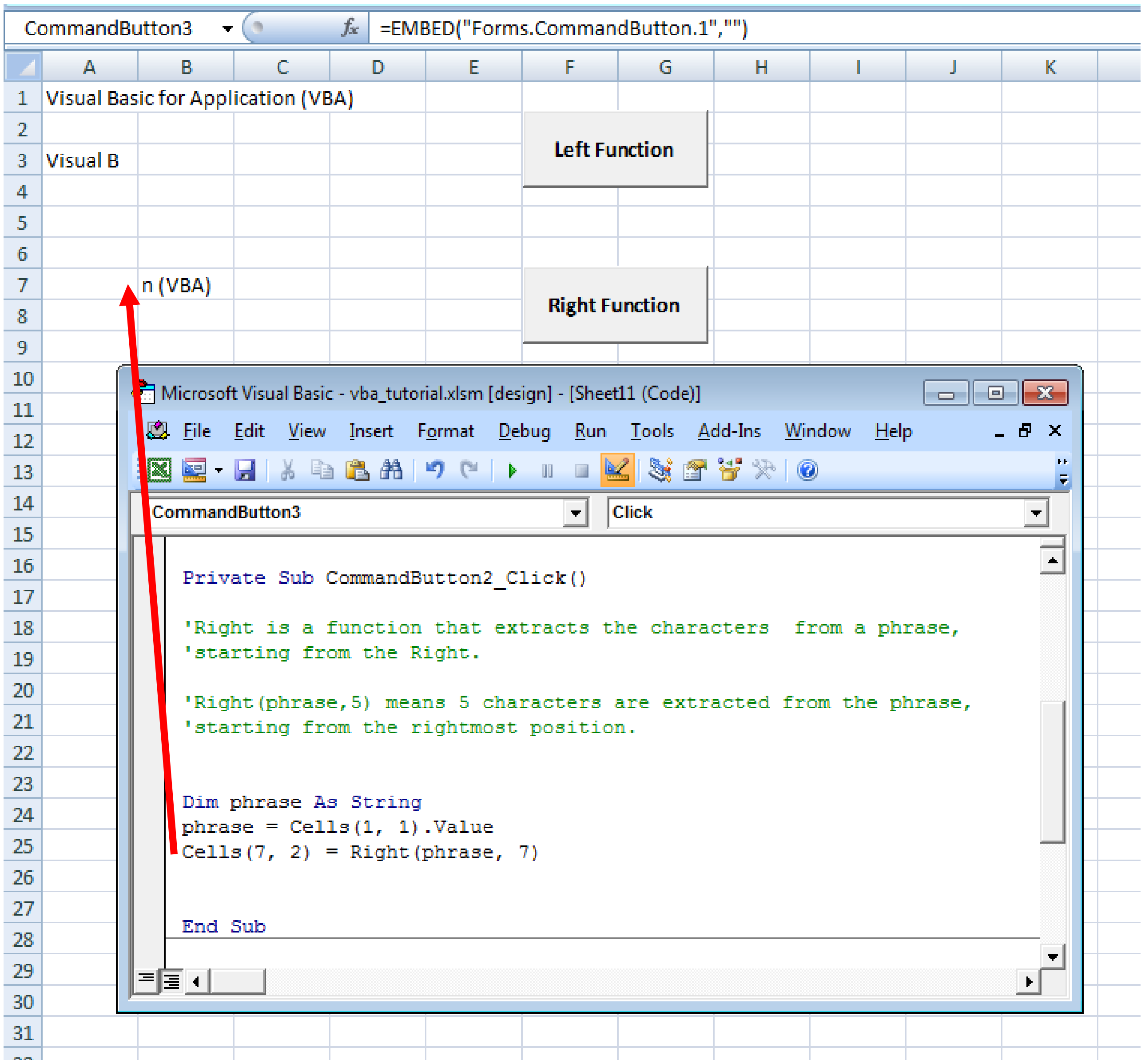
The *Left* function extracts a specified number of characters from a string starting from the left. Consider the following example. A string value in a cell on a spreadsheet is stored as a string variable. The *Left* function is then performed on the string variable to extract the first eight characters, and the result displayed in another cell on the spreadsheet.



```
Option Explicit

Private Sub CommandButton1_Click()

'Left is a function that extracts the characters  from a phrase,
'starting from the left.

'Left(phrase,4) means 4 characters are extracted from the phrase,
'starting from the leftmost position.

Dim phrase As String
phrase = Cells(1, 1).Value
Cells(3, 1) = Left(phrase, 8)

End Sub
```

2.5 Right Function

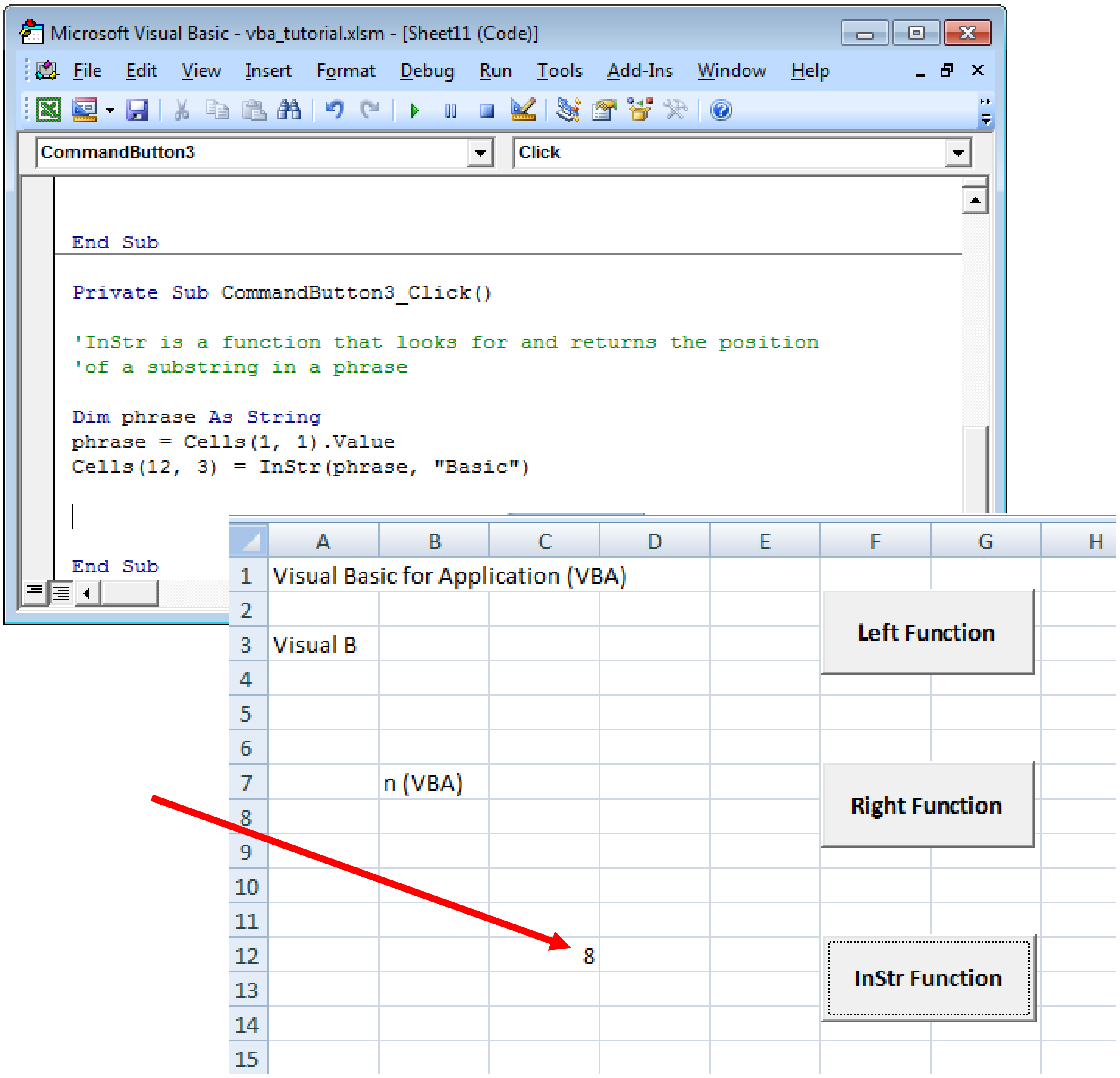
The *Right* function extracts a specified number of characters from a string starting from the right. Consider the following example.


CommandButton3
=EMBED("Forms.CommandButton.1","")
Visual Basic for Application (VBA)
Visual B
n (VBA)
Left Function
Right Function
Microsoft Visual Basic - vba_tutorial.xlsm [design] - [Sheet11 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
CommandButton3
Click

```
Private Sub CommandButton2_Click()

'Right is a function that extracts the characters  from a phrase,
'starting from the Right.

'Right(phrase,5) means 5 characters are extracted from the phrase,
'starting from the rightmost position.


Dim phrase As String
phrase = Cells(1, 1).Value
Cells(7, 2) = Right(phrase, 7)


End Sub
```

2.6 InStr Function

The *InStr* function finds and returns the starting position (from the left) of a specified substring within a string. In the following example, the position of the start of the substring “Basic” is found from the original string “Visual Basic for Application (VBA)”.

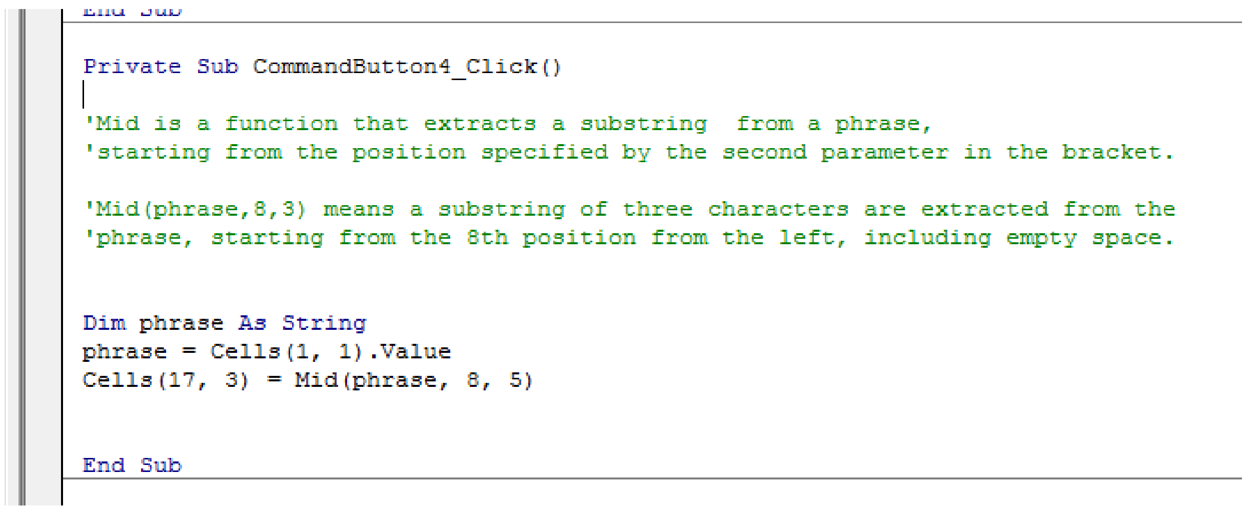

Microsoft Visual Basic - vba_tutorial.xlsm - [Sheet11 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
CommandButton3
Click
A B C D E F G H
Left Function
Right Function
InStr Function

```
End Sub

Private Sub CommandButton3_Click()

'InStr is a function that looks for and returns the position
'of a substring in a phrase

Dim phrase As String
phrase = Cells(1, 1).Value
Cells(12, 3) = InStr(phrase, "Basic")

End Sub
```

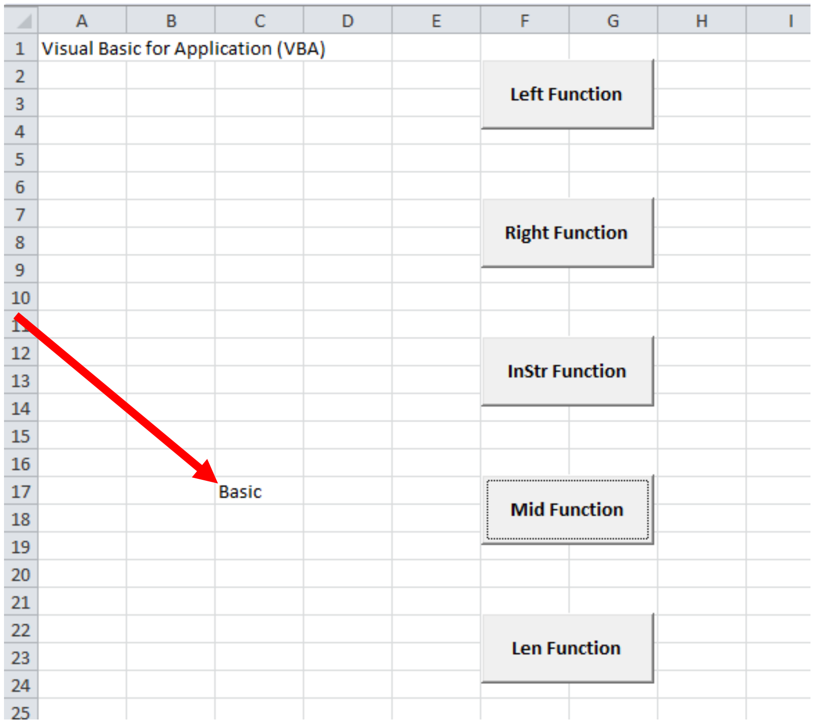
	A	B	C	D
1	Visual Basic for Application (VBA)			
2				
3	Visual B			
4				
5				
6				
7		n (VBA)		
8				
9				
10				
11				
12			8	
13				
14				
15				

2.7 Mid Function

The *Mid* function extracts a specified number of characters from a string starting at a specified position (from the left), of the original string. For example



```
End Sub

Private Sub CommandButton4_Click()

'Mid is a function that extracts a substring  from a phrase,
'starting from the position specified by the second parameter in the bracket.

'Mid(phrase,8,3) means a substring of three characters are extracted from the
'phrase, starting from the 8th position from the left, including empty space.


Dim phrase As String
phrase = Cells(1, 1).Value
Cells(17, 3) = Mid(phrase, 8, 5)


End Sub
```



	A	B	C	D	E	F	G	H	I
1	Visual Basic for Application (VBA)								
17			Basic						

2.8 Reversing a String

The *StrReverse* function will reverse the order of the characters in the original string. For example

```
Private Sub StringExample( )

        Dim strFirstname, strLastname, strFullname, strNameScramble As String
        Dim intStrinLeng As Integer

        strFirstname = “Sam”
        strLastname = “Johnson”

        strFullname = strLasttname & “, “ & strFirstname

        strNameScramble = StrReverse(strFullname)



End Sub
```

The value of the string variable *strNameScramble* at the end of the procedure will therefore be *“maS ,nosnhoJ”*

2.9 Upper Case and Lower Case Conversion

A string can be converted to all upper case or all lower case by using the case conversion functions in VBA, *UCase* and *LCase* respectively. For example

```
Private Sub StringExample( )

        Dim strFirstname, strLastname, strCaseFirstname, strCaseLastname As String

        strFirstname = “Sam”
        strLastname = “Johnson”

        strCaseFirstname = LCase(strFirstname)

        strCaseLastname = Ucase(strLasttname)



End Sub
```

The above procedure will yield values for the first name variable and the last name variable of *“sam”* and *“JOHNSON”* respectively.

2.10 Basics of Dates

A variable of the date data type is declared and assigned a value as follows

```
Dim datevariable as Date
```

and

```
datevariable = #the date#
```

The components of a date are separated by a “/” or a “-“.The date value must always be wrapped with the “#” characters for it to be correctly assigned and saved as such. Alternately, the date may be assigned by passing a string type through the *DateValue* built-in function, as follows

```
datevariable = DateValue(“the date”)
```

For example

```
Private Sub DateExample( )

        Dim dteStartDate As Date

        dteStartDate = #09/03/2013#


End Sub
```

or

```
Private Sub DateExample( )

        Dim dteStartDate As Date

        dteStartDate = DateValue("09/03/2013")


End Sub
```

or

```
Private Sub DateExample( )

        Dim dteStartDate As Date
        Dim strDateInput As String

        strDateInput = "09/03/2013"

        dteStartDate = DateValue(strDateInput)


End Sub
```

A date variable is called just like any other variable learned thus far.

For example the following code will display the start date value above, on a message box.

```
MsgBox "The project started on:  " & dteStartDate
```

2.11 Date Formats

The date formats used include

- mm-dd-yy
- mm-dd-yyyy

where

mm is the two digit month
dd is the two digit day
yy is the two digit year
yyyy is the four digit year

In countries where it is customary to quote the day first, followed by the month, and then the year, acceptable formats include

- dd-mmm-yy
- dd mmm yyyy
- dd-mmmm-yyyy
- dd mmmm yyyy

where

mmm is the short name of the month (first three letters starting with an upper case letter)
mmmm is the full name of the month (starting with an upper case letter)

Short and full names of months may also be used in the United States style of dates as follows

- mmm dd, yy
- mmm dd, yyyy
- mmmm dd, yy
- mmmm dd, yyyy

where

mmm is the short name of the month (first three letters starting with an upper case letter)
mmmm is the full name of the month (starting with an upper case letter)

2.12 Current Date

The current date can be obtained by the built-in date function which simply calls the keyword *Date*, as follows

dteStartDate = Date

To display the current date on a message box, for instance, the code would be

MsgBox Date

2.13 Components of a Date

The month, day, and year components may be extracted from a date and stored elsewhere. This can be done using the *DatePart* function with syntax

DatePart("yyyy", the date)

DatePart("m", the date)

DatePart("d", the date)

For example

```
Private Sub DateExample( )

        Dim dteStartDate As Date
        Dim intMonth, intDay as Integer

        dteStartDate = DateValue("09/03/2013")

        intMonth = DatePart("m", dteStartDate)


End Sub
```

The value of the variable *intMonth* would therefore be 9.

Another set of functions that can perform extraction of the date components are as follows

Day(the date)

Month(the date)

Year(the date)

For example, for the date 09/03/2013,

intDay = Day(dteStartDate)

will return a value of 3.

2.14 Subtracting Dates

Dates may be subtracted to determine the time elapsed between two dates. This can be done with the *DateDiff* function. The syntax is

```
DateDiff("x",  earlier data, later date)
```

where *x* is *d* for days, or *m* for months, or *yyyy* for years, based on what scale is selected to calculate the elapsed time.

For example, if a project began on September 9, 2013, and was completed on December 13, 2013, calculate the actual project duration in days.

```
Private Sub DateExample( )

        Dim dteStartDate, dteEndDate As Date
        Dim intProjectDuration As Integer

        dteStartDate = #09/03/2013#
        dteEndDate = #12/13/2013#

        intProjectDuration = DateDiff("d", dteStartDate, dteEndDate)

End Sub
```

The project duration value will be 101 days.

2.15 Time

Time is a date data type in VBA. The difference being the "**:**" separating the components instead of the "/" or the "-" used in dates. The declaration, extraction of components, and arithmetic operations of time values are identical in structure to those of dates.

2.16 Type Conversion Functions

A type conversion function converts an input expression to a specific data type. In VBA, type conversion functions begin with the "C" followed by the name of or a suffix representing the name of the data type the input will be converted to.

Example: The *CInt* function converts an expression to an integer type. Consider the following

```
Dim dblBalance As Double

    dblBalance = 2.23
    intEntry = CInt(dblBalance)

    UserForm1.TextBox1.Value = intEntry
```

At the end of this code the value displayed in the text box on the user form will be a value of 2.

Example: The *CDate* function will convert an input string type into a date type. Consider the following

```
strFirst = "1/25/2013"

strLast = "2/15/2013"

intDurationDays = DateDiff("d", CDate(strFirst), CDate(strLast) )
```

The *DateDiff* function requires date type inputs to compute the period elapsed between the input dates. In this case the dates are originally string types and will yield a run time error if inserted directly into the *DateDiff* function. The *CDate* function was called to convert the strings inline to date types which the *DateDiff* can now use to make the evaluation, resulting in a duration of 21 days.

A selection of type conversion functions, the requirements of their input expressions, and the type of the returned value are presented in Table 5.

Table 1: Type conversion functions

Type Conversion Function	Input	Output Data Type
CStr	Date, time, numeric expressions	String.
CBool	Char, string, or numeric expression.	Boolean
CDate	A representation of a date or time, as a string	Date
CInt	Values ranging from -2,147,483,648 through 2,147,483,647	Integer
CDbl	Values ranging from 1.79769313486231570E+308 through -4.94065645841246544E-324 and 4.94065645841246544E-324 through 1.79769313486231570E+308	Double
CByte	0 through 255	Byte

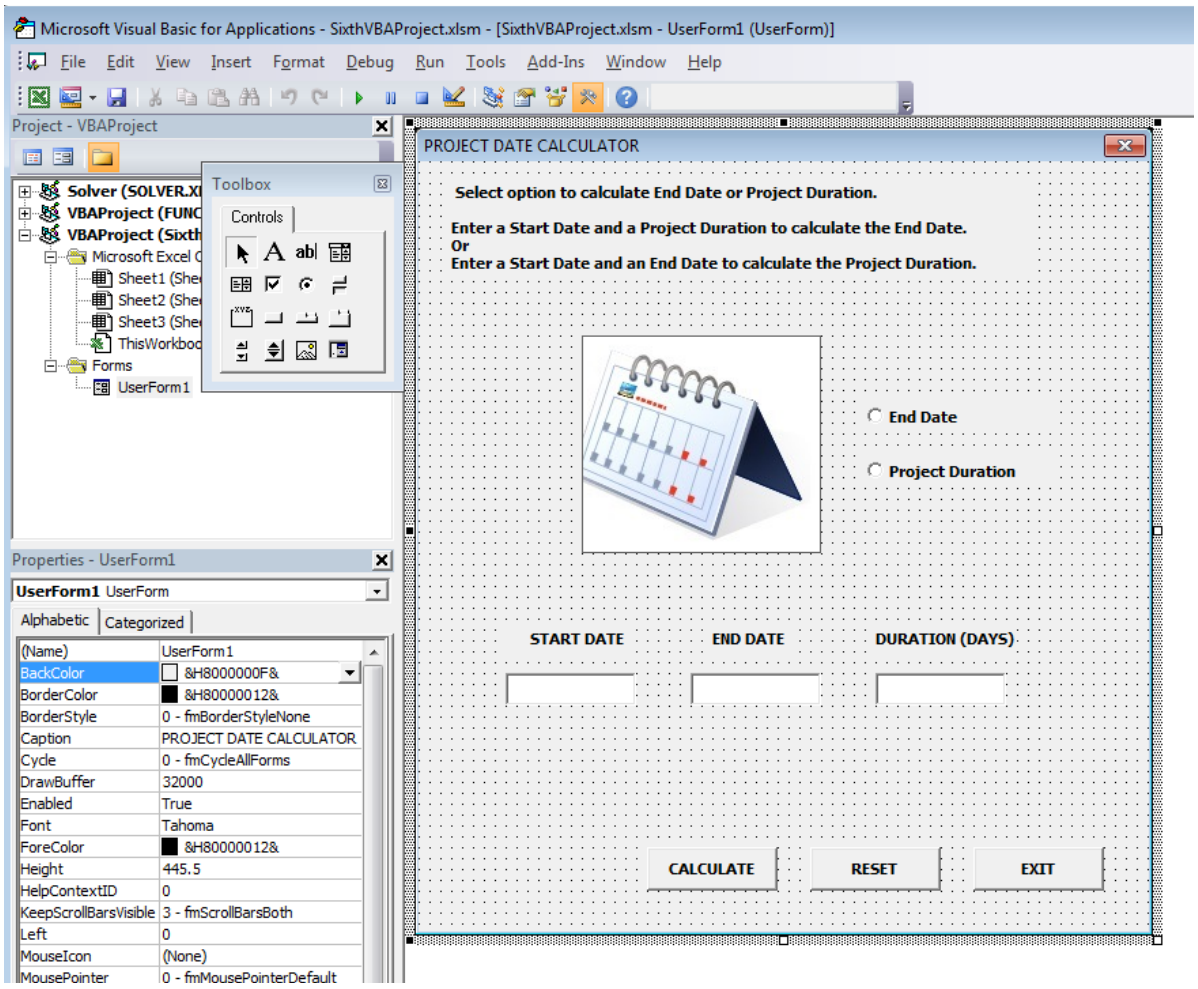
2.17 Sixth VBA Project

Problem Statement

A simple date calculator will be developed. The date calculator will calculate the number of days between two valid dates, and it will calculate the project completion date based on a start date and a project duration.

Solution

The proposed framework is as follows



(Name)	UserForm1
BackColor	&H8000000F&
BorderColor	&H80000012&
BorderStyle	0 - fmBorderStyleNone
Caption	PROJECT DATE CALCULATOR
Cycle	0 - fmCycleAllForms
DrawBuffer	32000
Enabled	True
Font	Tahoma
ForeColor	&H80000012&
Height	445.5
HelpContextID	0
KeepScrollBarsVisible	3 - fmScrollBarsBoth
Left	0
MouseIcon	(None)
MousePointer	0 - fmMousePointerDefault

PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

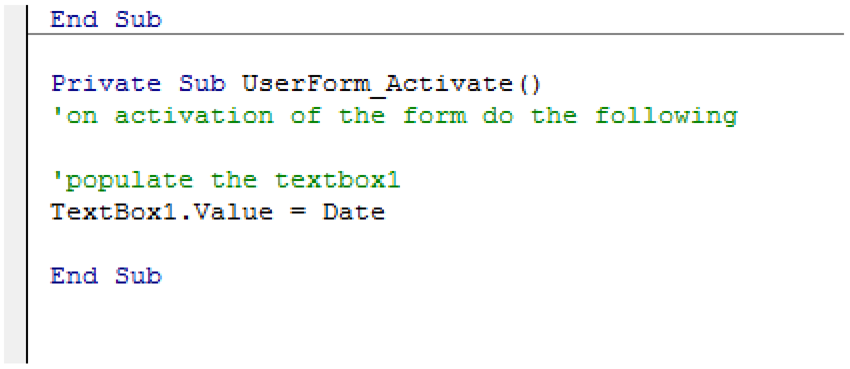
The user will select to either calculate a project duration in days based on a start date and an end date, or calculate the end date based on the start date and the project duration given in days. If the option to calculate the end date is selected then the end date text box will be highlighted to alert the user to enter data in the other controls in order to calculate the value for the highlighted control, and likewise for the project duration calculation option.

The calculation options are controlled by option buttons, also often called radio buttons. The options will be mutually exclusive, so if the end date option is selected, the project duration option will be disabled, and vice versa.

The *Calculation* button fires the code to run the calculation based on the selected option. The *Reset* button clears the text boxes and reverts the form back to original settings. The *Exit* button closes out the form.

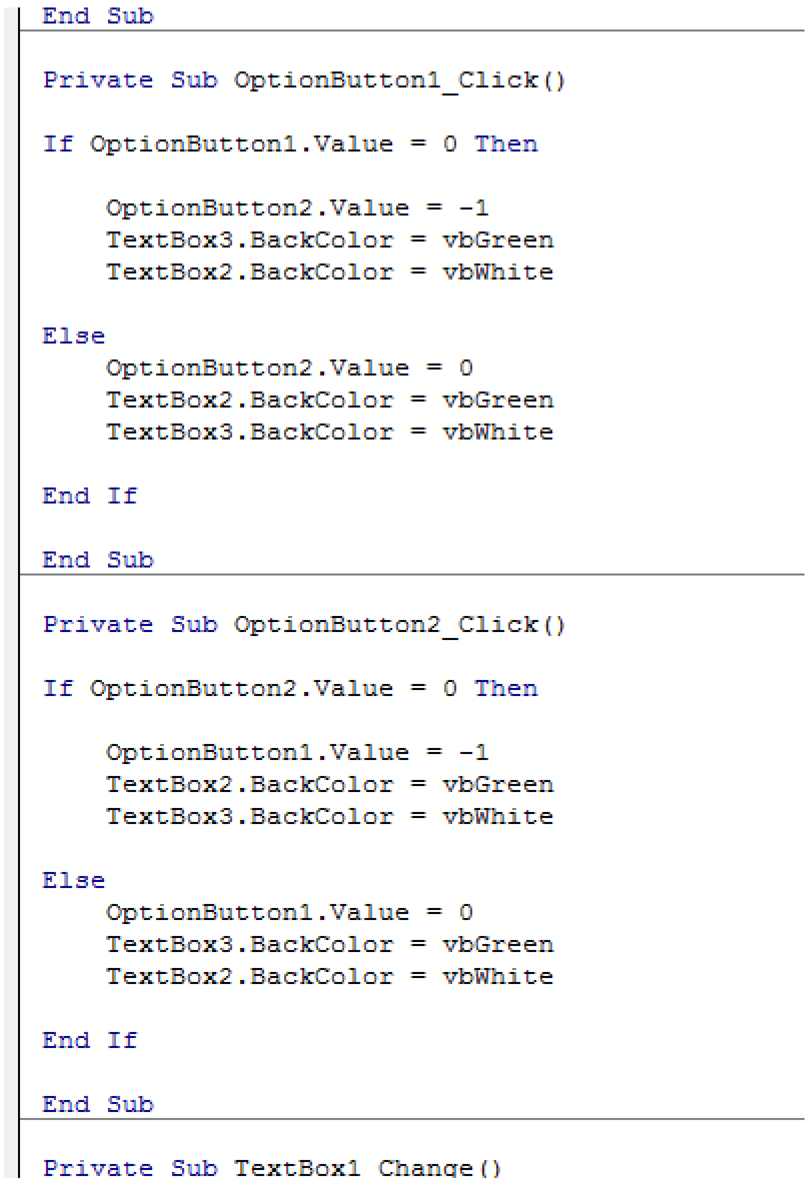
A logo of your choice will be added as an image control. This is any image type file such as a jpeg, tiff, png, etc.

On opening the form, the start date text box value will be defaulted to the current date, using the built-in *Date* function, and the radio buttons will both be unchecked. This can be implemented using the UserForm *Activate* procedure as follows



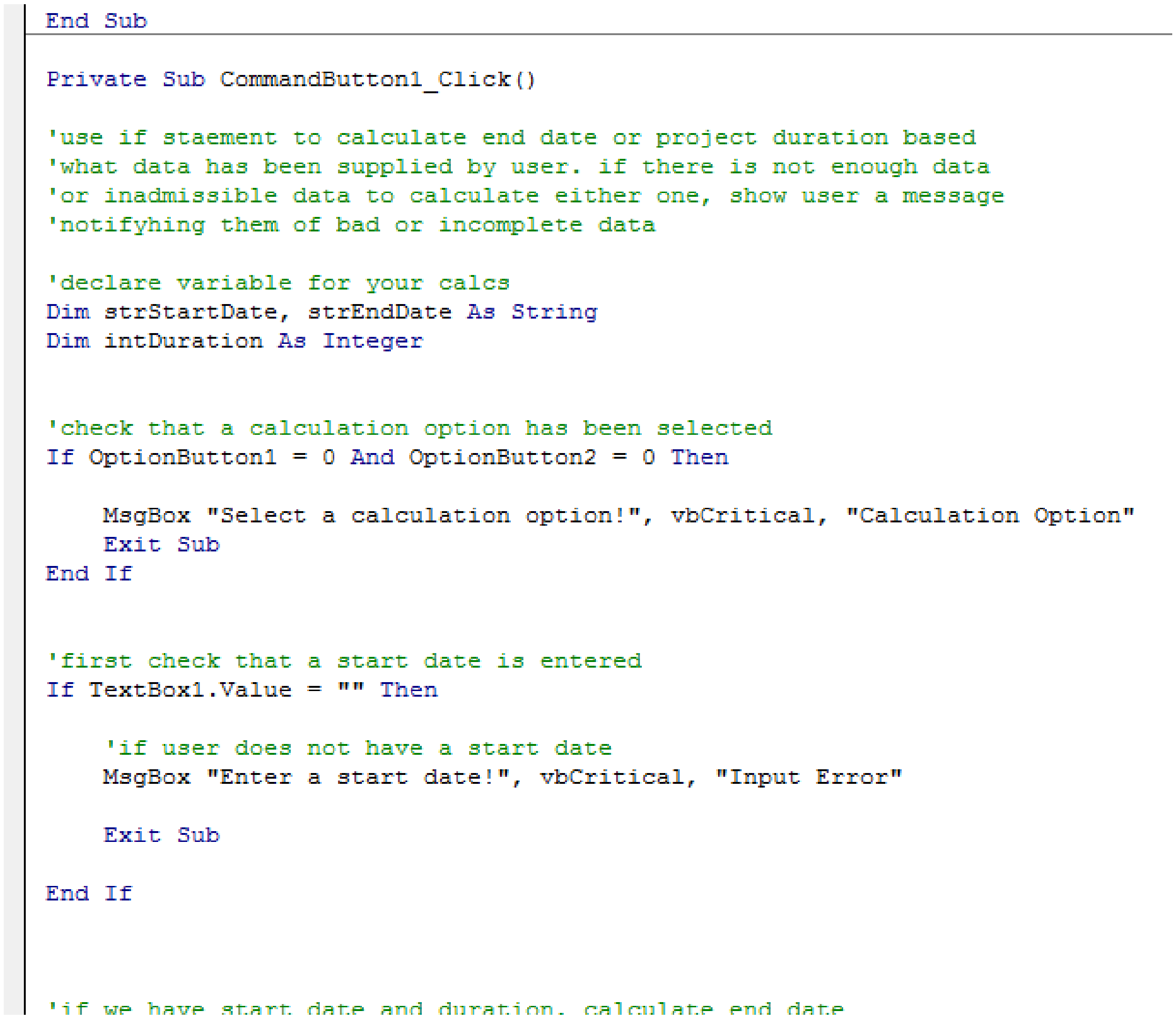
```
End Sub

Private Sub UserForm_Activate()
'on activation of the form do the following

'populate the textbox1
TextBox1.Value = Date

End Sub
```

When an option button is clicked on or selected, its value property is "-1", otherwise it is zero. The mutual exclusivity of the calculation options and the highlighting of the relevant text box value to be calculated shall be implemented using the option button values as follows



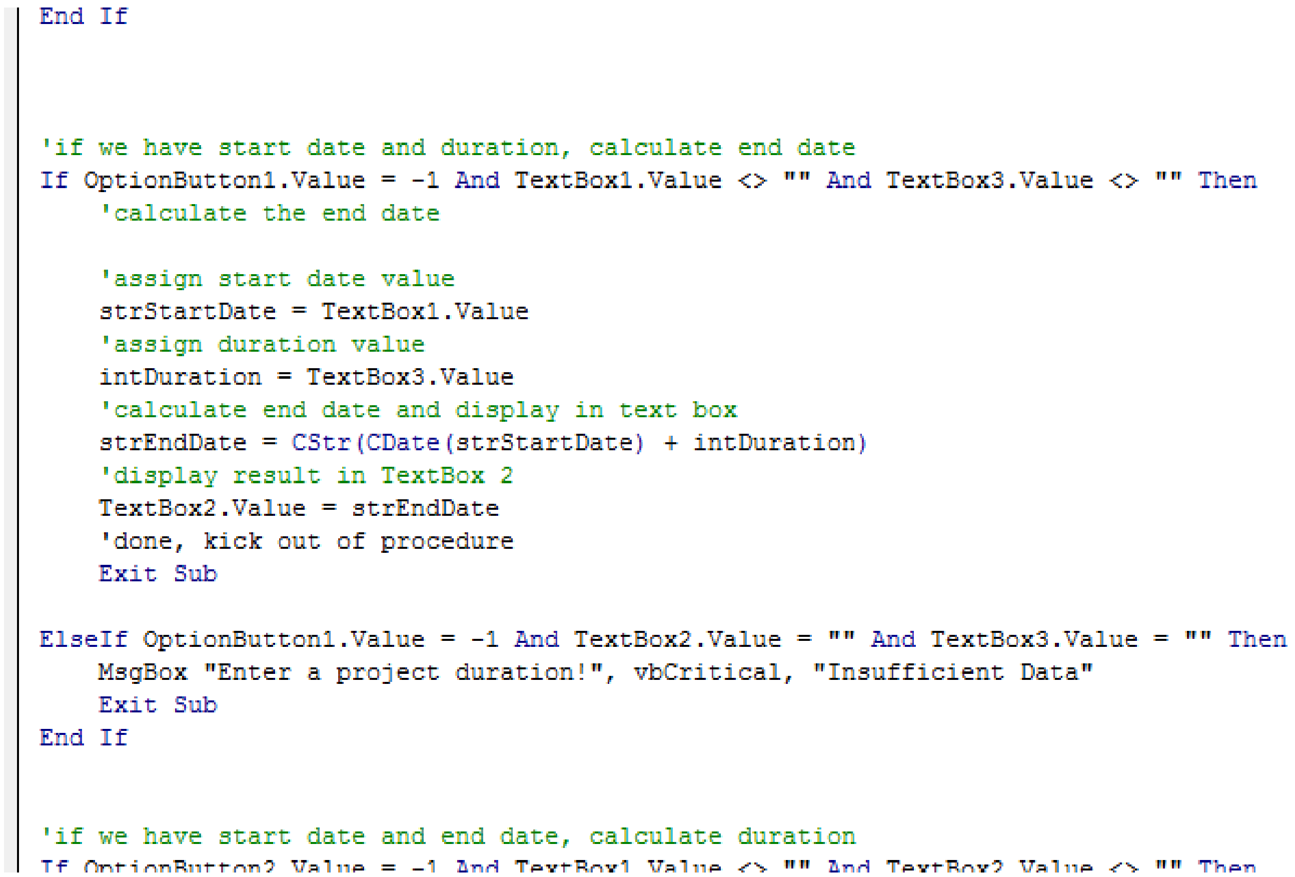
```
End Sub

Private Sub OptionButton1_Click()

If OptionButton1.Value = 0 Then

    OptionButton2.Value = -1
    TextBox3.BackColor = vbGreen
    TextBox2.BackColor = vbWhite

Else
    OptionButton2.Value = 0
    TextBox2.BackColor = vbGreen
    TextBox3.BackColor = vbWhite

End If

End Sub

Private Sub OptionButton2_Click()

If OptionButton2.Value = 0 Then

    OptionButton1.Value = -1
    TextBox2.BackColor = vbGreen
    TextBox3.BackColor = vbWhite

Else
    OptionButton1.Value = 0
    TextBox3.BackColor = vbGreen
    TextBox2.BackColor = vbWhite

End If

End Sub

Private Sub TextBox1 Change()
```

On clicking the *Calculate* button, the program will check that an option button has been selected before proceeding. If not, the user will be prompted to select a calculation option. The program then checks if a valid start date has been entered. Variables to be used later in the procedure for the calculations are declared at this stage.



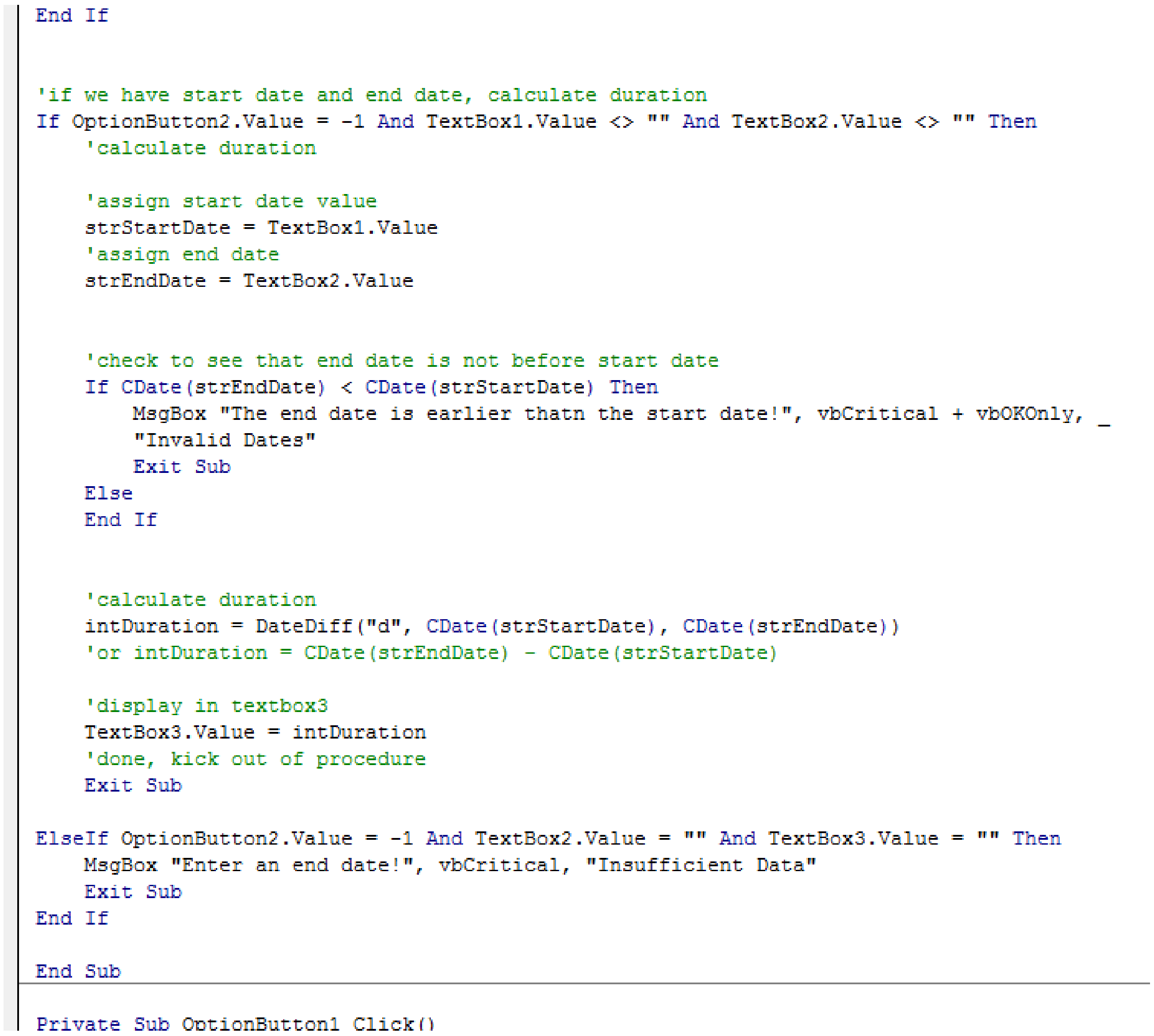
```
End Sub
Private Sub CommandButton1_Click()

'use if staement to calculate end date or project duration based
'what data has been supplied by user. if there is not enough data
'or inadmissible data to calculate either one, show user a message
'notifyhing them of bad or incomplete data

'declare variable for your calcs
Dim strStartDate, strEndDate As String
Dim intDuration As Integer


'check that a calculation option has been selected
If OptionButton1 = 0 And OptionButton2 = 0 Then

    MsgBox "Select a calculation option!", vbCritical, "Calculation Option"
    Exit Sub
End If


'first check that a start date is entered
If TextBox1.Value = "" Then

    'if user does not have a start date
    MsgBox "Enter a start date!", vbCritical, "Input Error"

    Exit Sub

End If



'if we have start date and duration, calculate end date
```

If the end date option is selected but the start date or project duration is not entered, the user is prompted to enter that information. If the inputs are valid and complete, the calculation proceeds. The text box values are saved to the variables. Type conversion functions are used to convert the text box entries into the relevant (date) type for the date function calculations. The result is displayed back onto the form in the end date text box.



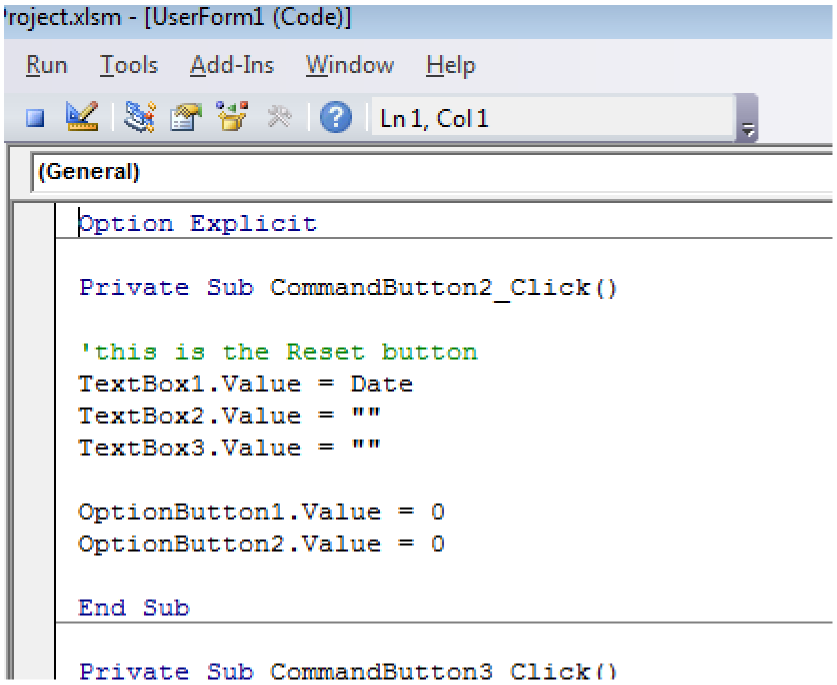
```
End If



'if we have start date and duration, calculate end date
If OptionButton1.Value = -1 And TextBox1.Value <> "" And TextBox3.Value <> "" Then
    'calculate the end date

    'assign start date value
    strStartDate = TextBox1.Value
    'assign duration value
    intDuration = TextBox3.Value
    'calculate end date and display in text box
    strEndDate = CStr(CDate(strStartDate) + intDuration)
    'display result in TextBox 2
    TextBox2.Value = strEndDate
    'done, kick out of procedure
    Exit Sub

ElseIf OptionButton1.Value = -1 And TextBox2.Value = "" And TextBox3.Value = "" Then
    MsgBox "Enter a project duration!", vbCritical, "Insufficient Data"
    Exit Sub
End If


'if we have start date and end date, calculate duration
If OptionButton2.Value = -1 And TextBox1.Value <> "" And TextBox2.Value <> "" Then
```

If the project duration option is selected but the start date or end date is not entered, the user is prompted to enter that information. If the inputs are valid and complete, the calculation proceeds. The text box values are saved to the variables. Type conversion functions are used to convert the text box entries into the relevant type for the date function calculations. The result is displayed back onto the form in the duration text box.



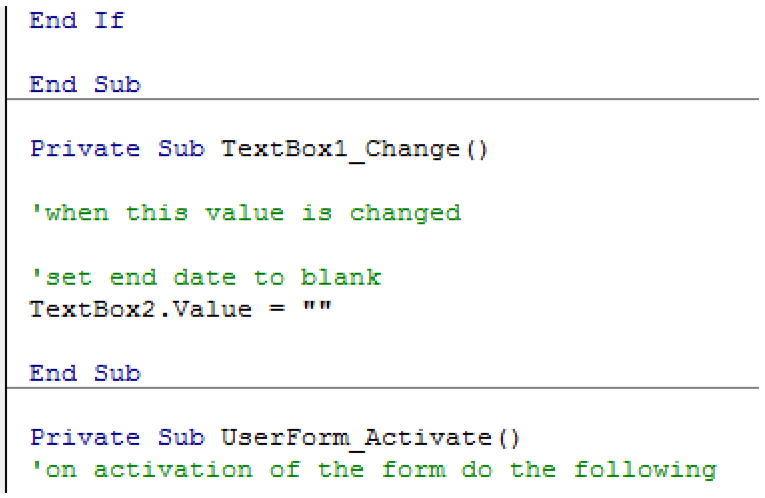
```
End If


'if we have start date and end date, calculate duration
If OptionButton2.Value = -1 And TextBox1.Value <> "" And TextBox2.Value <> "" Then
    'calculate duration

    'assign start date value
    strStartDate = TextBox1.Value
    'assign end date
    strEndDate = TextBox2.Value


    'check to see that end date is not before start date
    If CDate(strEndDate) < CDate(strStartDate) Then
        MsgBox "The end date is earlier thatn the start date!", vbCritical + vbOKOnly, _
        "Invalid Dates"
        Exit Sub
    Else
    End If


    'calculate duration
    intDuration = DateDiff("d", CDate(strStartDate), CDate(strEndDate))
    'or intDuration = CDate(strEndDate) - CDate(strStartDate)

    'display in textbox3
    TextBox3.Value = intDuration
    'done, kick out of procedure
    Exit Sub

ElseIf OptionButton2.Value = -1 And TextBox2.Value = "" And TextBox3.Value = "" Then
    MsgBox "Enter an end date!", vbCritical, "Insufficient Data"
    Exit Sub
End If

End Sub

Private Sub OptionButton1 Click()
```

The *Reset* button is to be used to restore original settings and inputs as when the program opens.


'roject.xlsm - [UserForm1 (Code)]
Run Tools Add-Ins Window Help
Ln 1, Col 1
(General)

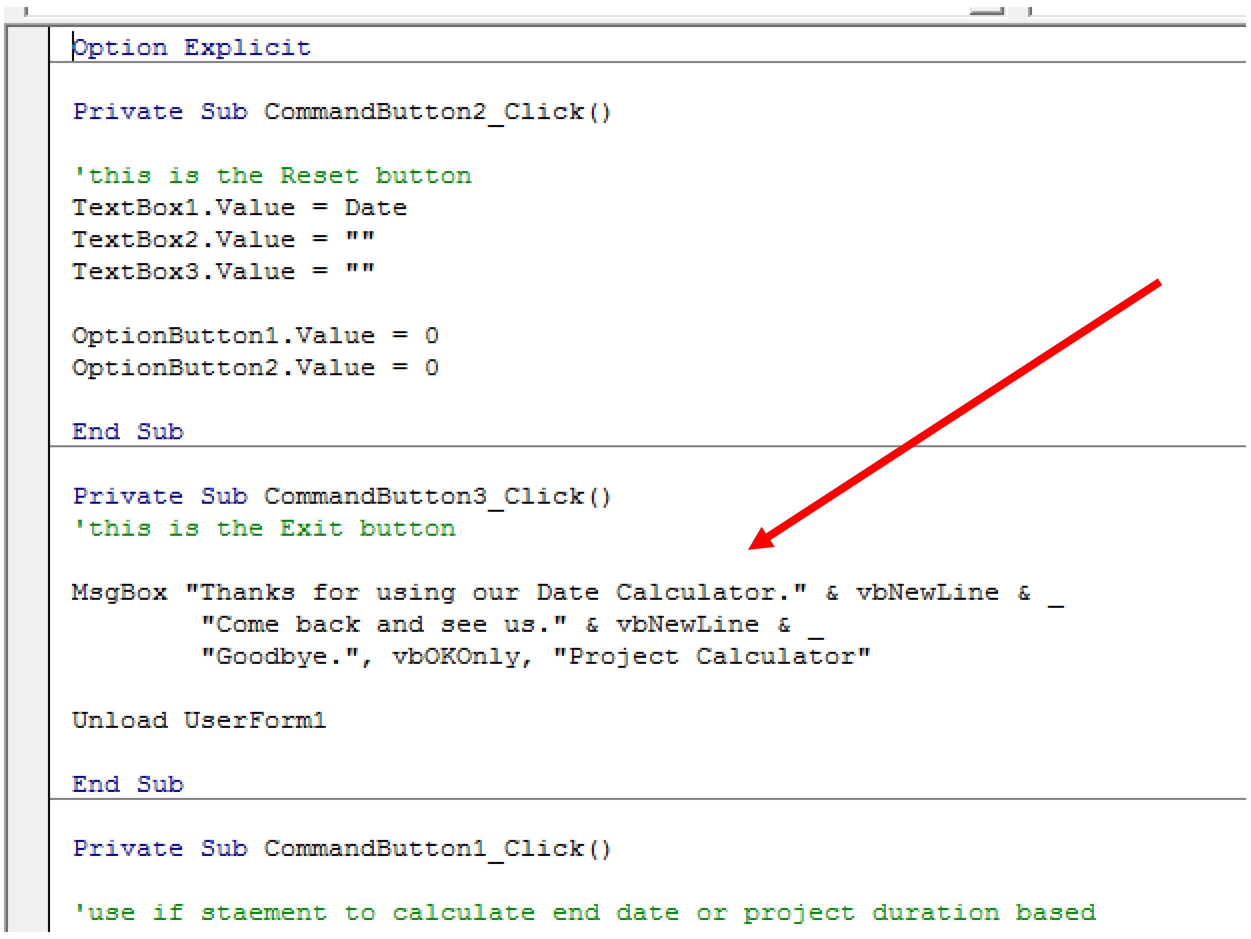
```
Option Explicit

Private Sub CommandButton2_Click()

'this is the Reset button
TextBox1.Value = Date
TextBox2.Value = ""
TextBox3.Value = ""

OptionButton1.Value = 0
OptionButton2.Value = 0

End Sub

Private Sub CommandButton3_Click()
```

To facilitate editing and reworking of calculations, a change procedure for the start date text box is also provided. This procedure clears the end date entry any time a new start date is entered.



```
End If

End Sub

Private Sub TextBox1_Change()

'when this value is changed

'set end date to blank
TextBox2.Value = ""

End Sub

Private Sub UserForm_Activate()
'on activation of the form do the following
```

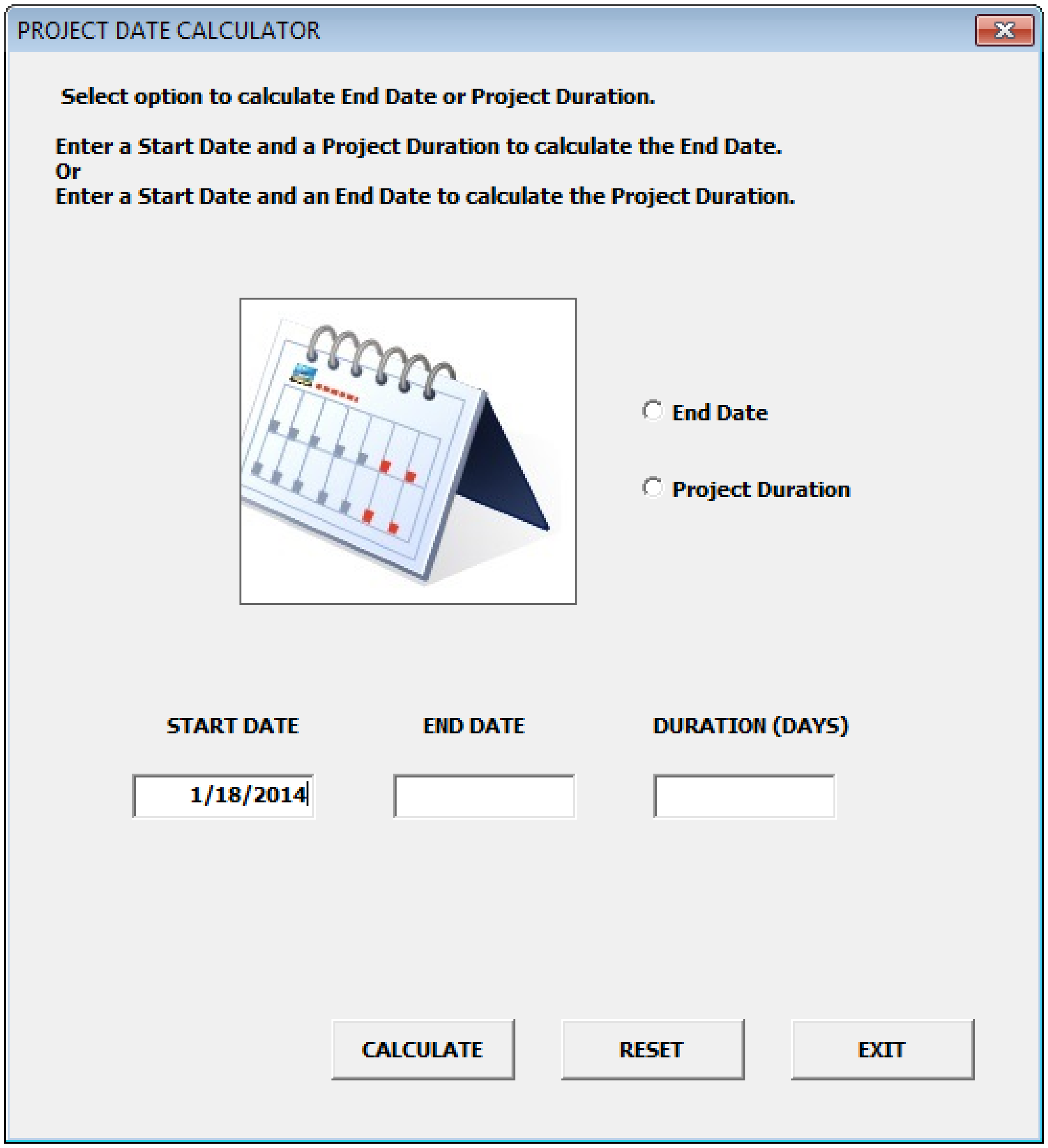
The *Exit* button closes out the form. A message prompts the user, after which the form closes.



```
Option Explicit

Private Sub CommandButton2_Click()

'this is the Reset button
TextBox1.Value = Date
TextBox2.Value = ""
TextBox3.Value = ""

OptionButton1.Value = 0
OptionButton2.Value = 0

End Sub

Private Sub CommandButton3_Click()
'this is the Exit button

MsgBox "Thanks for using our Date Calculator." & vbNewLine & _
        "Come back and see us." & vbNewLine & _
        "Goodbye.", vbOKOnly, "Project Calculator"

Unload UserForm1

End Sub

Private Sub CommandButton1_Click()

'use if staement to calculate end date or project duration based
```

Save the workbook.
Click on Debug. Select Compile VBAProject.
Correct any errors.
Resave.
If there are no errors, test the application.

Open the application.



PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

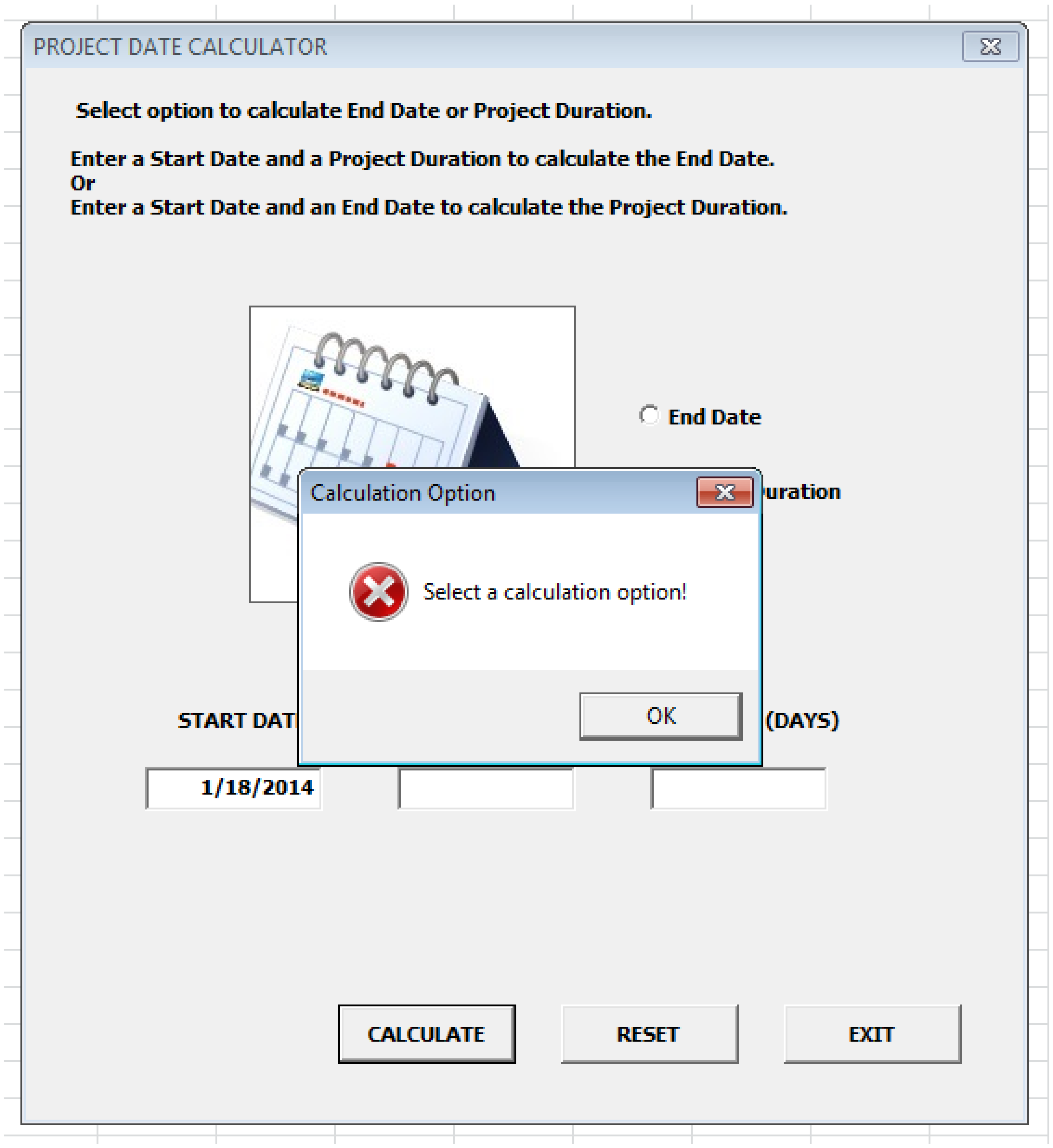
Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

○ End Date

○ Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014		

Click on *Calculate*.



PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

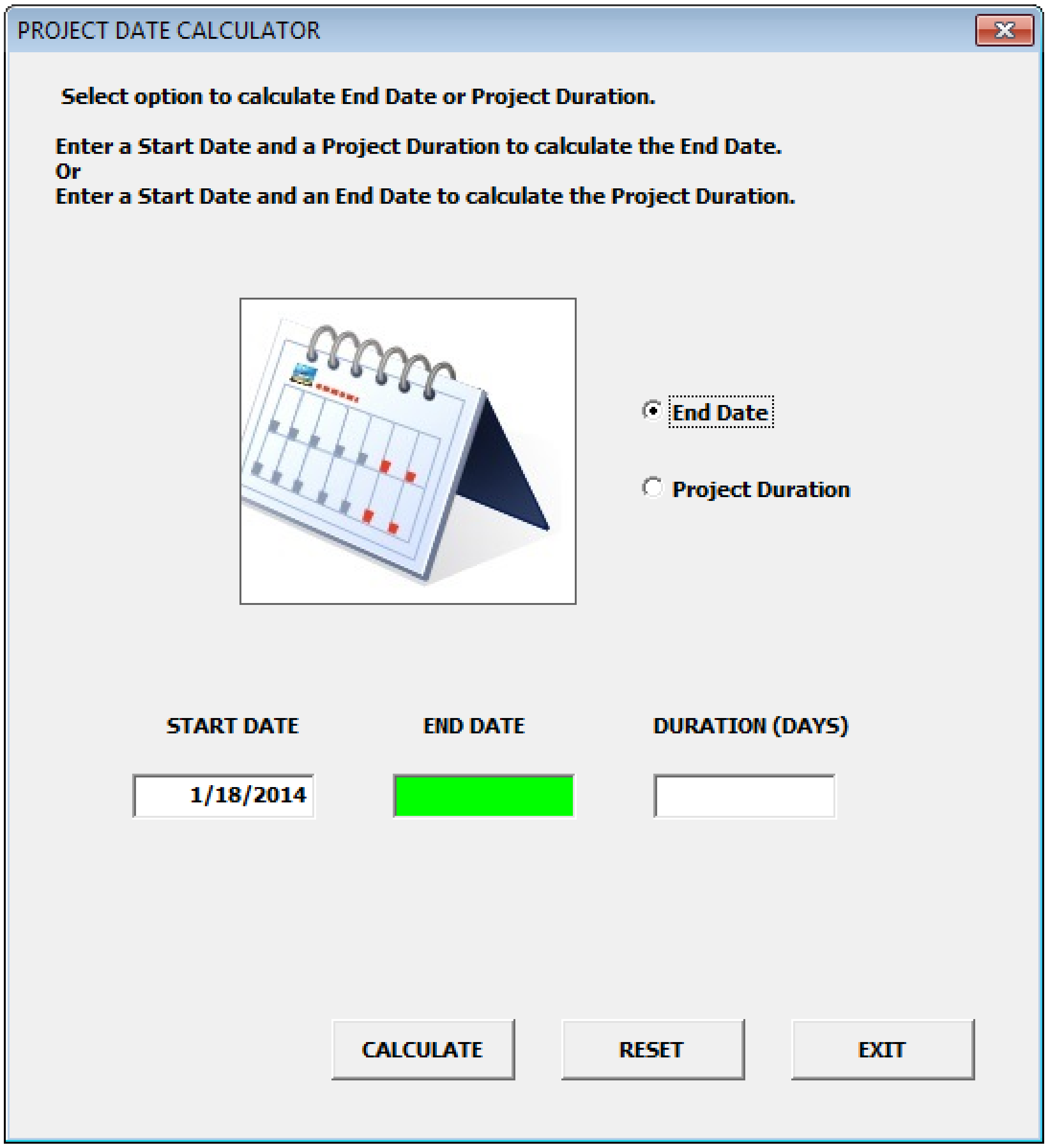
Calculation Option

Select a calculation option!

1/18/2014

Dismiss the message box.

Select to calculate *End Date*.


PROJECT DATE CALCULATOR

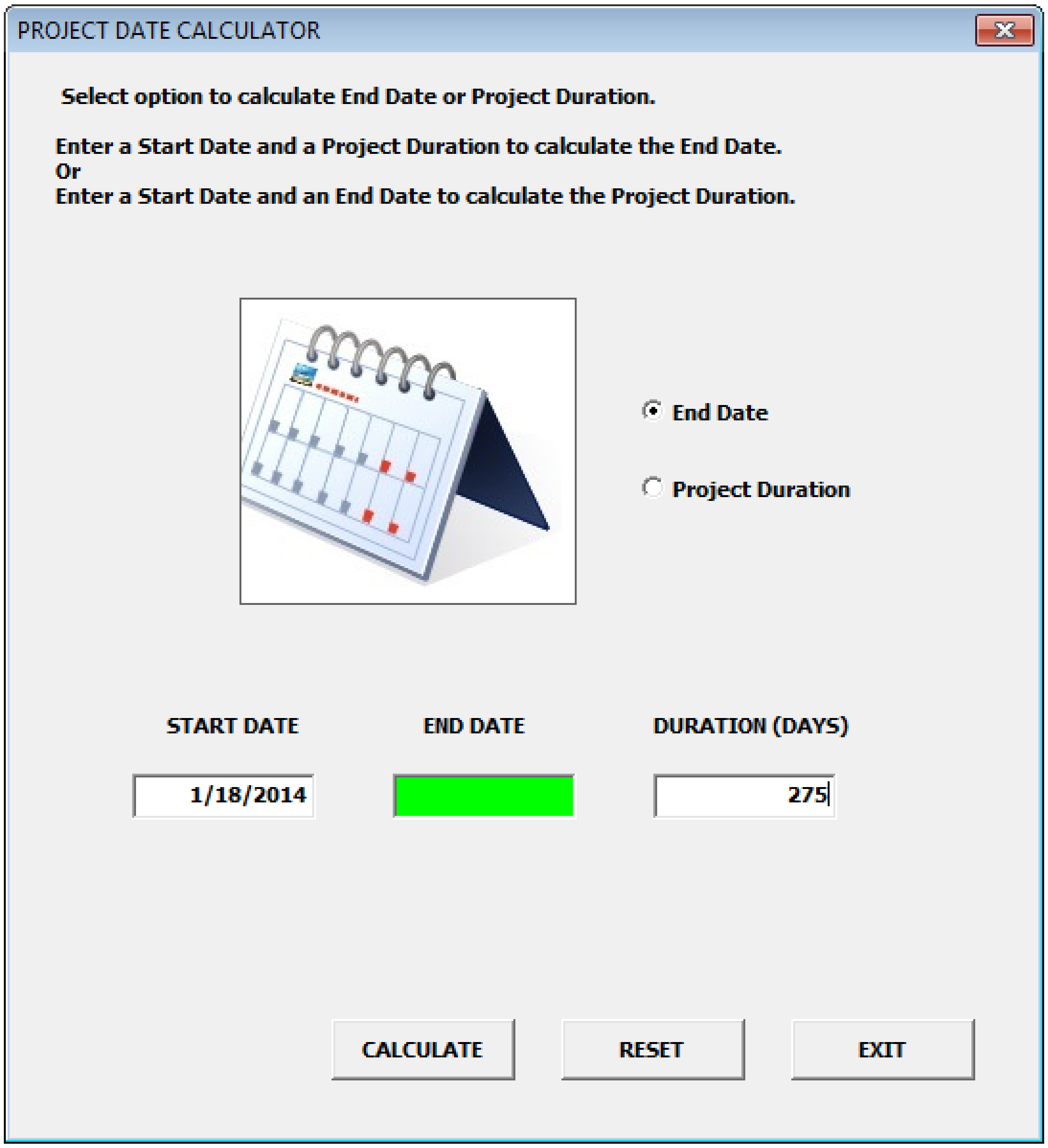
Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

[✓ End Date]
Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014	🟩	

Enter a project duration.



PROJECT DATE CALCULATOR

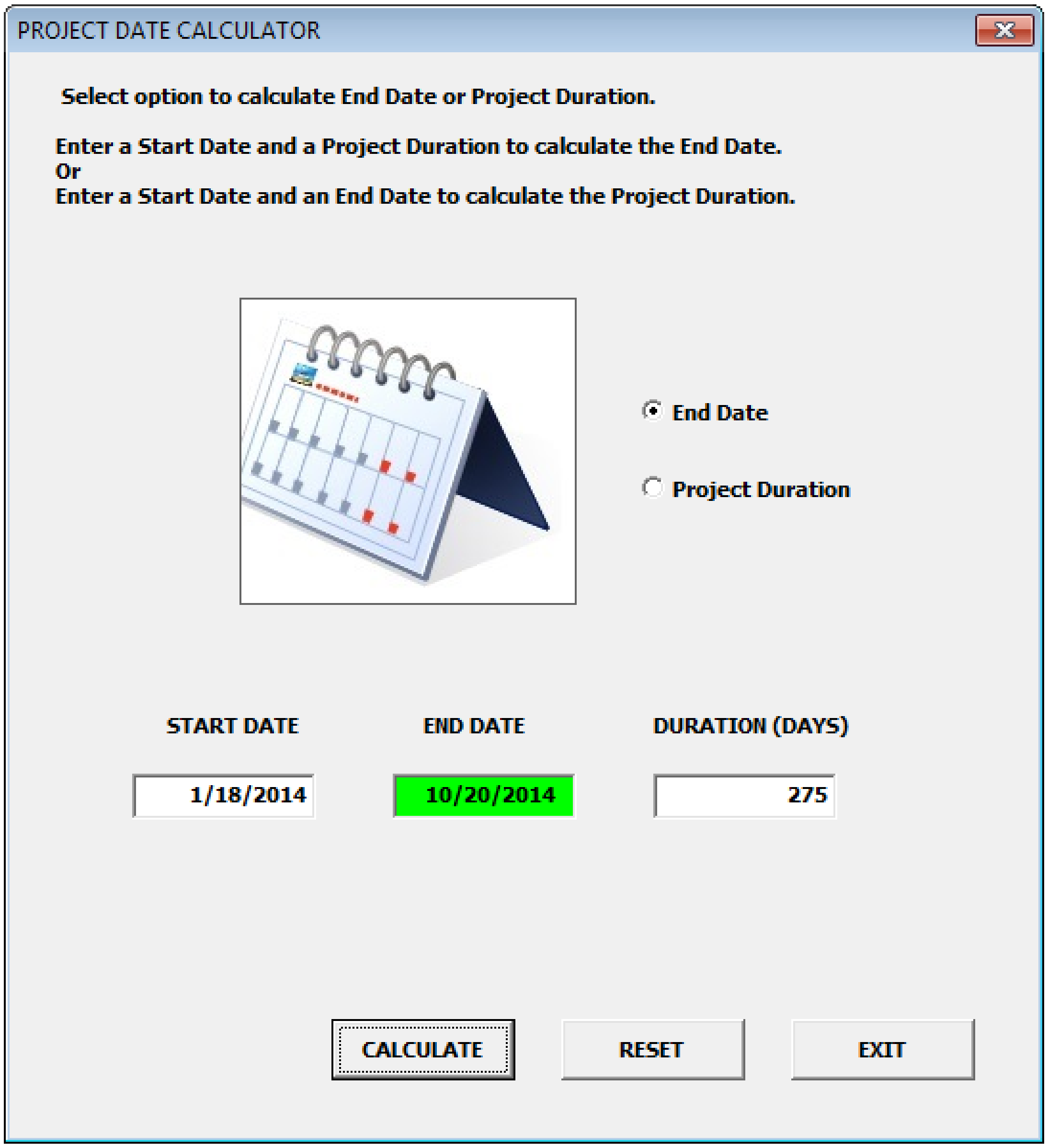
Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

[✓ End Date] Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014	🟩	275

Press *Calculate*.



PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

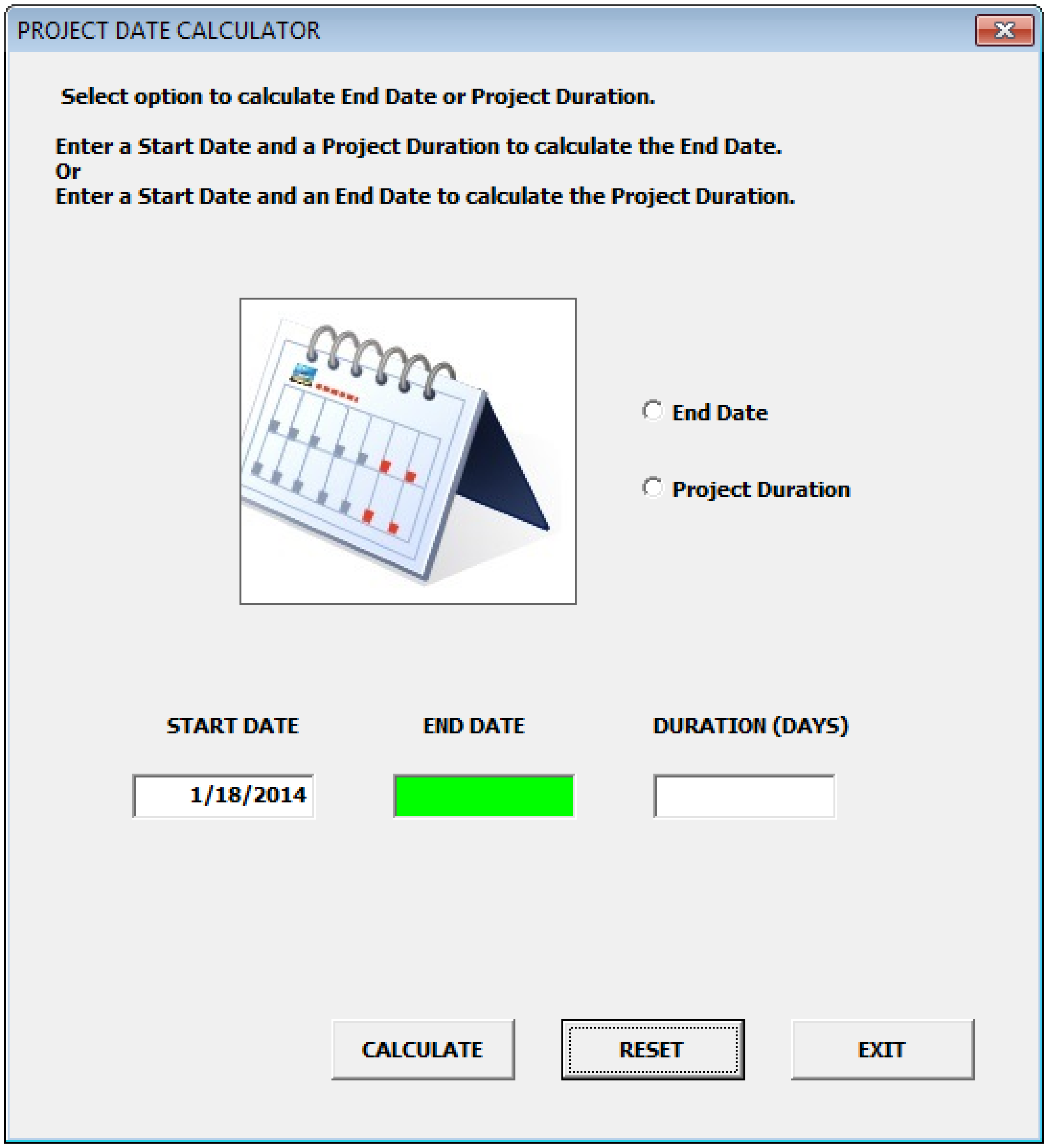
Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

[✓ End Date]

Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014	10/20/2014	275

Press *Reset*


PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

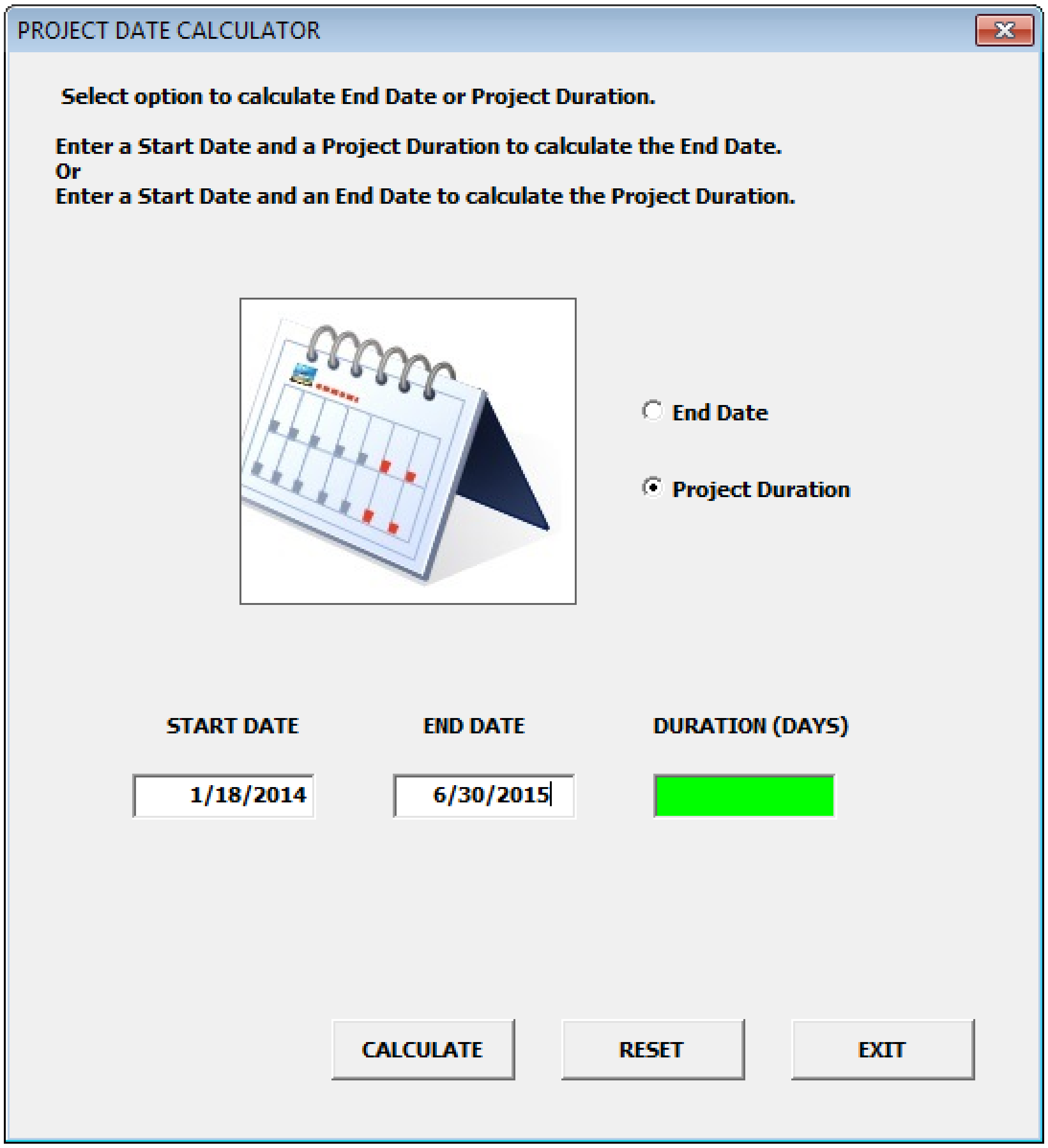
Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

○ End Date

○ Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014	🟩	

Select *Project Duration*, and enter an end date.



PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

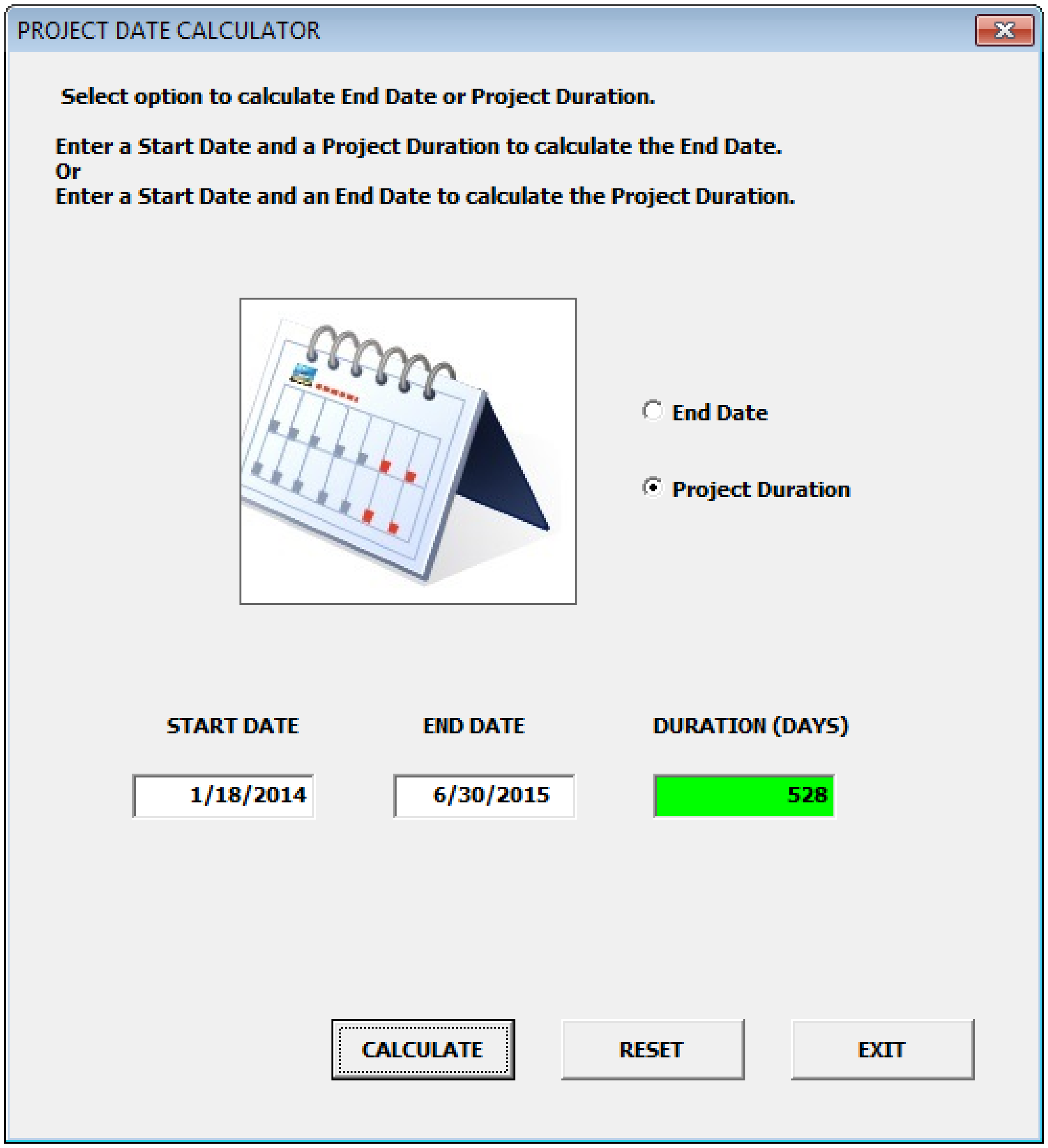
Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

( ) End Date

(●) Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014	6/30/2015	🟩

Press *Calculate*.



PROJECT DATE CALCULATOR

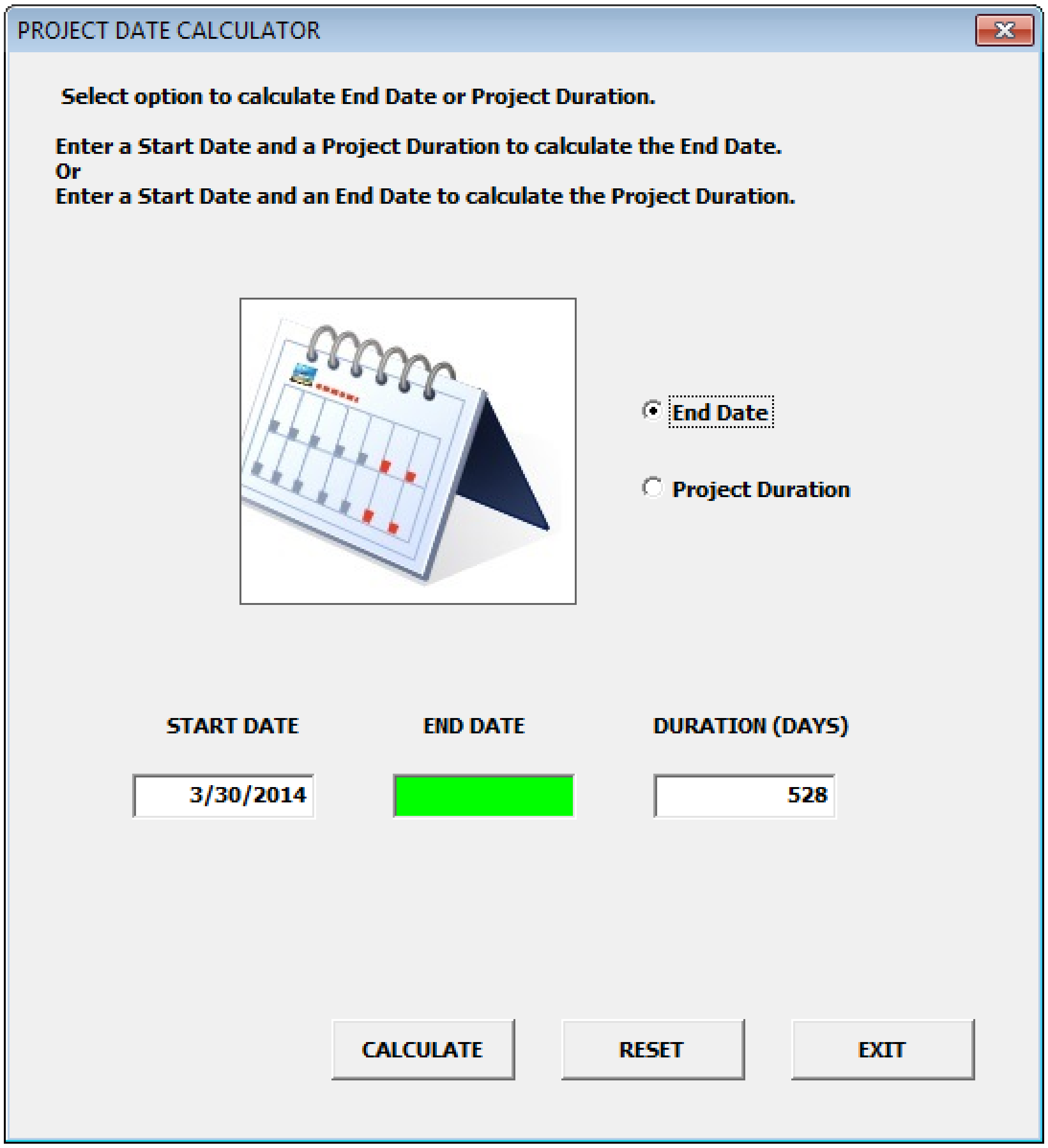
Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

○ End Date
● Project Duration

START DATE	END DATE	DURATION (DAYS)
1/18/2014	6/30/2015	528

Change the start date, and notice that the end date text box automatically clears.
Select the *End Date* option.



PROJECT DATE CALCULATOR

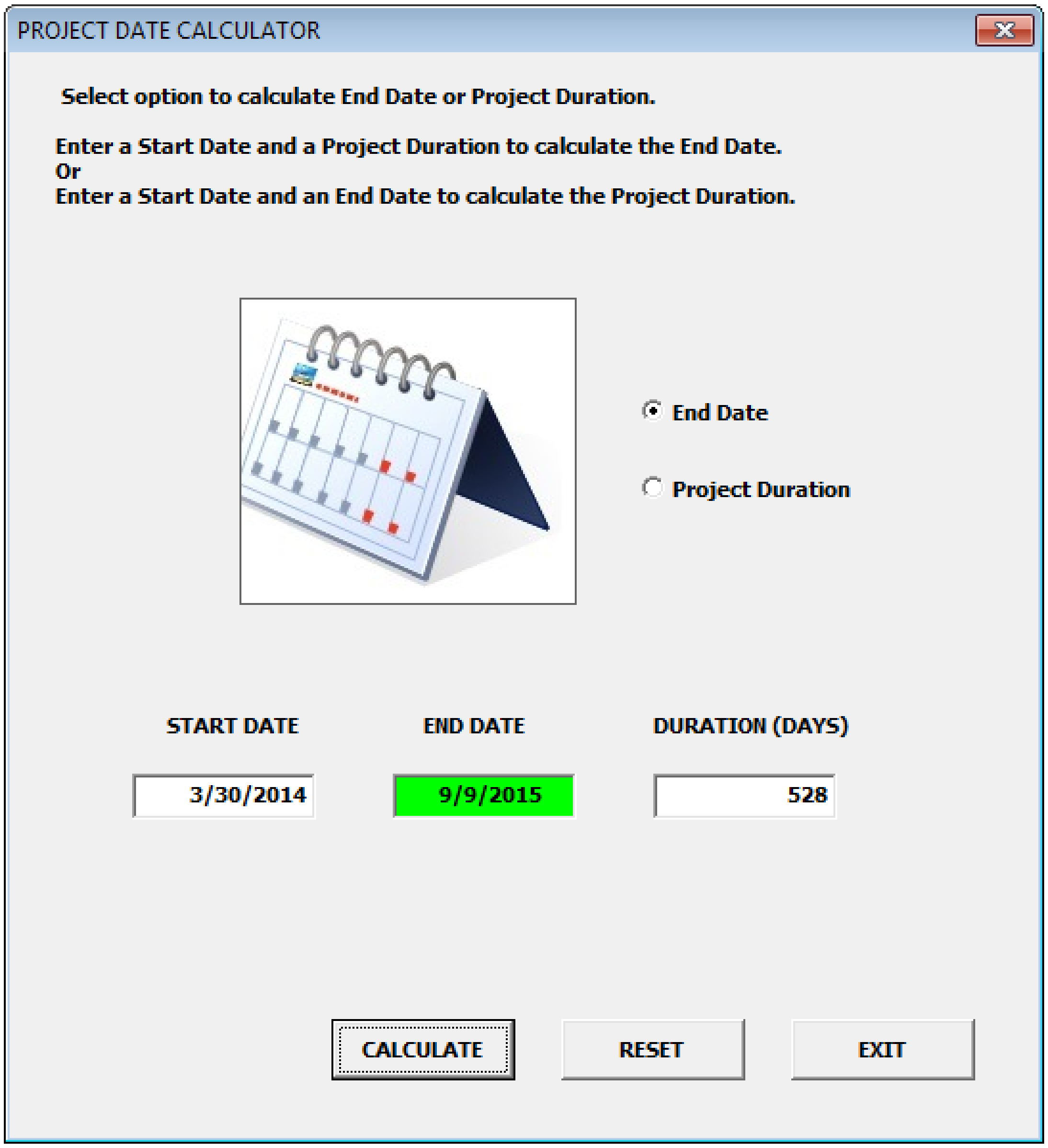
Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

[✓ End Date] Project Duration

START DATE	END DATE	DURATION (DAYS)
3/30/2014	🟩	528

Press *Calculate*.



PROJECT DATE CALCULATOR

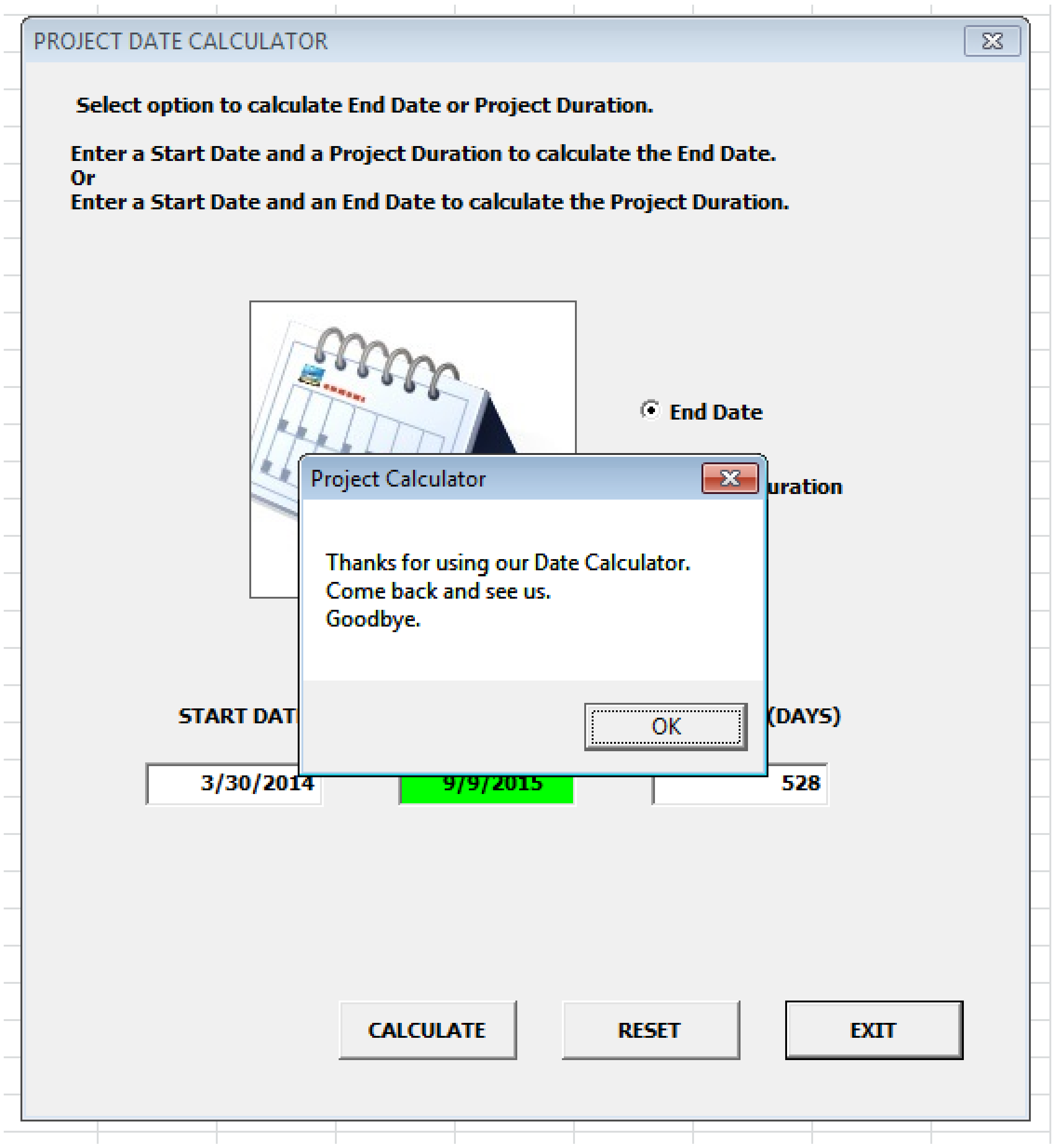
Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

[✓ End Date]
Project Duration

START DATE	END DATE	DURATION (DAYS)
3/30/2014	9/9/2015	528

Press *Exit*.



PROJECT DATE CALCULATOR

Select option to calculate End Date or Project Duration.

Enter a Start Date and a Project Duration to calculate the End Date.
Or
Enter a Start Date and an End Date to calculate the Project Duration.

(● End Date) uration

Project Calculator

Thanks for using our Date Calculator.
Come back and see us.
Goodbye.

OK

START DAT | | (DAYS)
3/30/2014 | 9/9/2015 | 528

Click OK to dismiss the message box.
The application closes.
The test has been an overwhelming success.

3. OBJECTS

3.1 Object-Oriented Programming

In VBA, developing programs consists primarily of manipulating objects by manipulating their properties and their methods. An object is a “thing” for example a form, a text box, a combo box etc. A property is something that describes some attribute of the object, for example color, size, font, etc. A method is some action associated with the object, for example to click on it, to clear its contents, to resize it, to double click on it etc.

Methods are set up by writing procedures for the relevant actions (events). The properties of an object may be manipulated in design time through the Properties Window by adjusting the values of the relevant properties. In run time (and break time), the object’s properties can be manipulated programmatically as part of some procedure. The code for such manipulations is generally of the form

Objectname.Property = newpropertyvalue

For example, changing the background color of a text box

UserForm1,TextBox5.BackColor = vbBlue

or inserting some text into a cell on a spreadsheet, hence changing the cells *Value* property.

Sheets(“Sheet1”).cells(1,3).Value = “SALARY”

Methods, similarly, have a general form

Objectname.Methodname

For example, clear the contents of text box on a form.

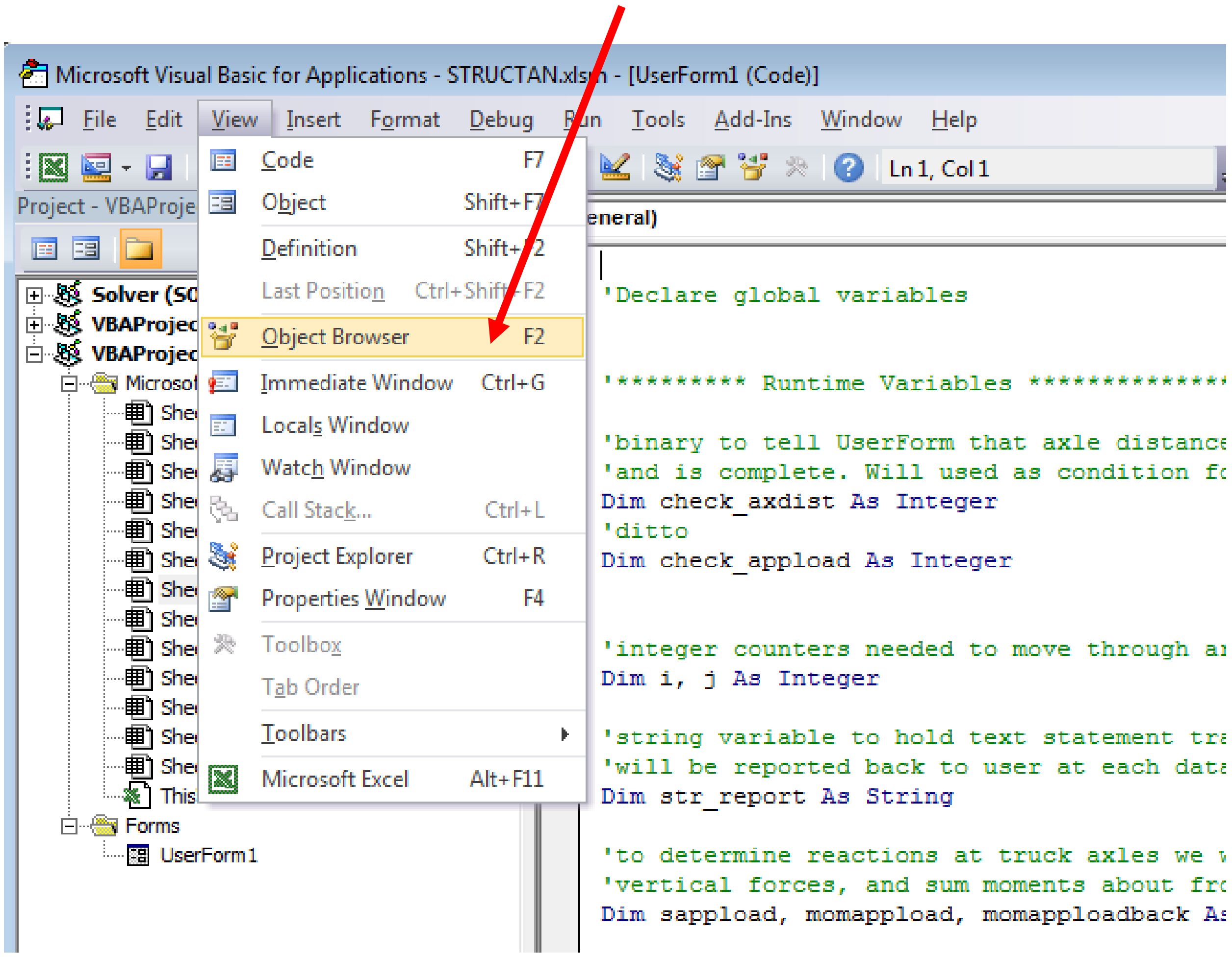
UserForm2.TextBox1.ClearContents

or close an open Excel workbook

ThisWorkBook.Close

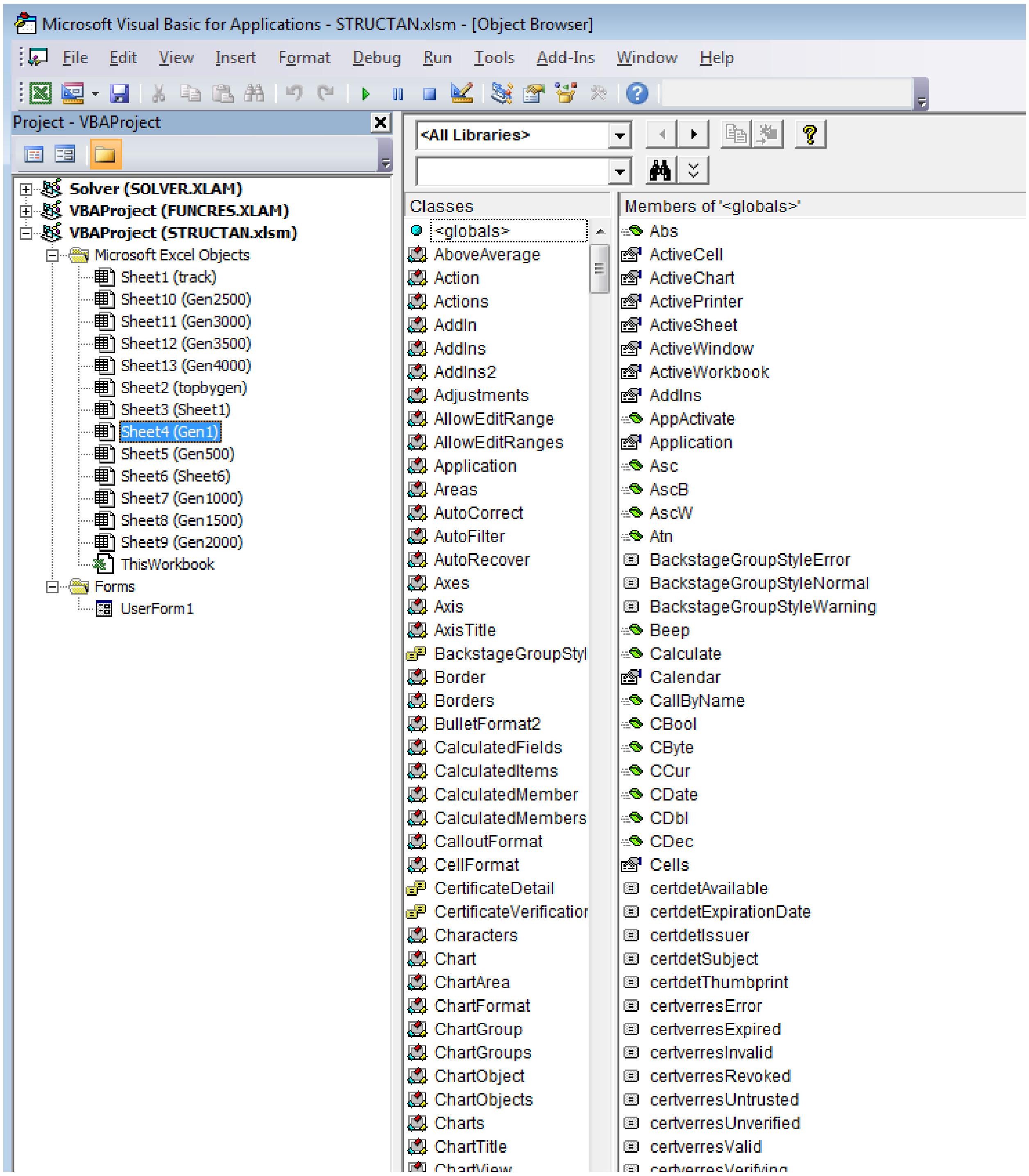
3.2 The VBA Object Browser

This element of the VBA environment provides details on all objects, methods, properties, events and the libraries available in VBA. To open the Object Browser
Click on the **View** menu.
Select **Object Browser**.

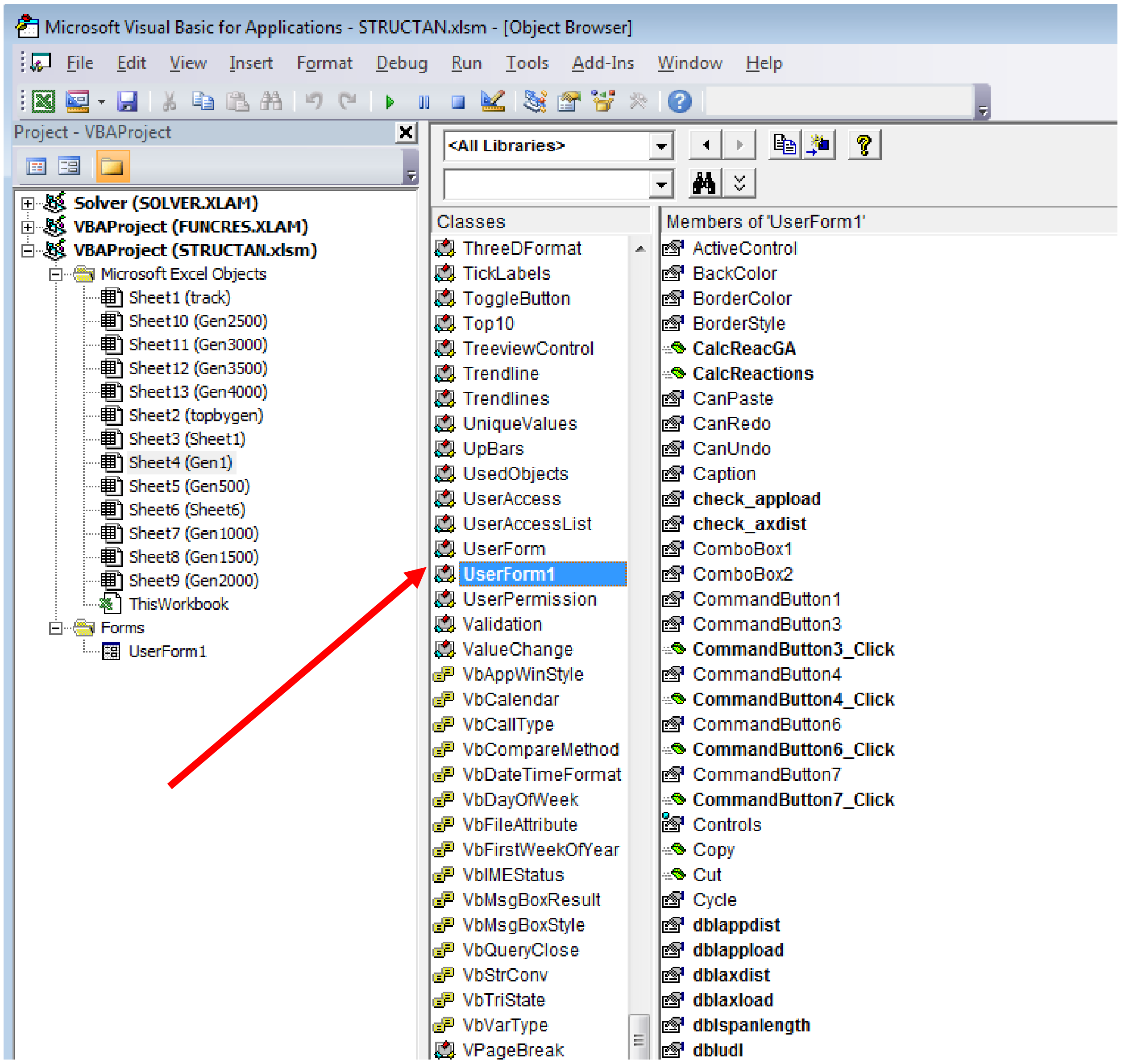


```
'Declare global variables

'********* Runtime Variables *************

'binary to tell UserForm that axle distance
'and is complete. Will used as condition fo
Dim check_axdist As Integer
'ditto
Dim check_appload As Integer


'integer counters needed to move through ar
Dim i, j As Integer

'string variable to hold text statement tra
'will be reported back to user at each data
Dim str_report As String

'to determine reactions at truck axles we w
'vertical forces, and sum moments about fro
Dim sappload, momappload, momapploadback As
```

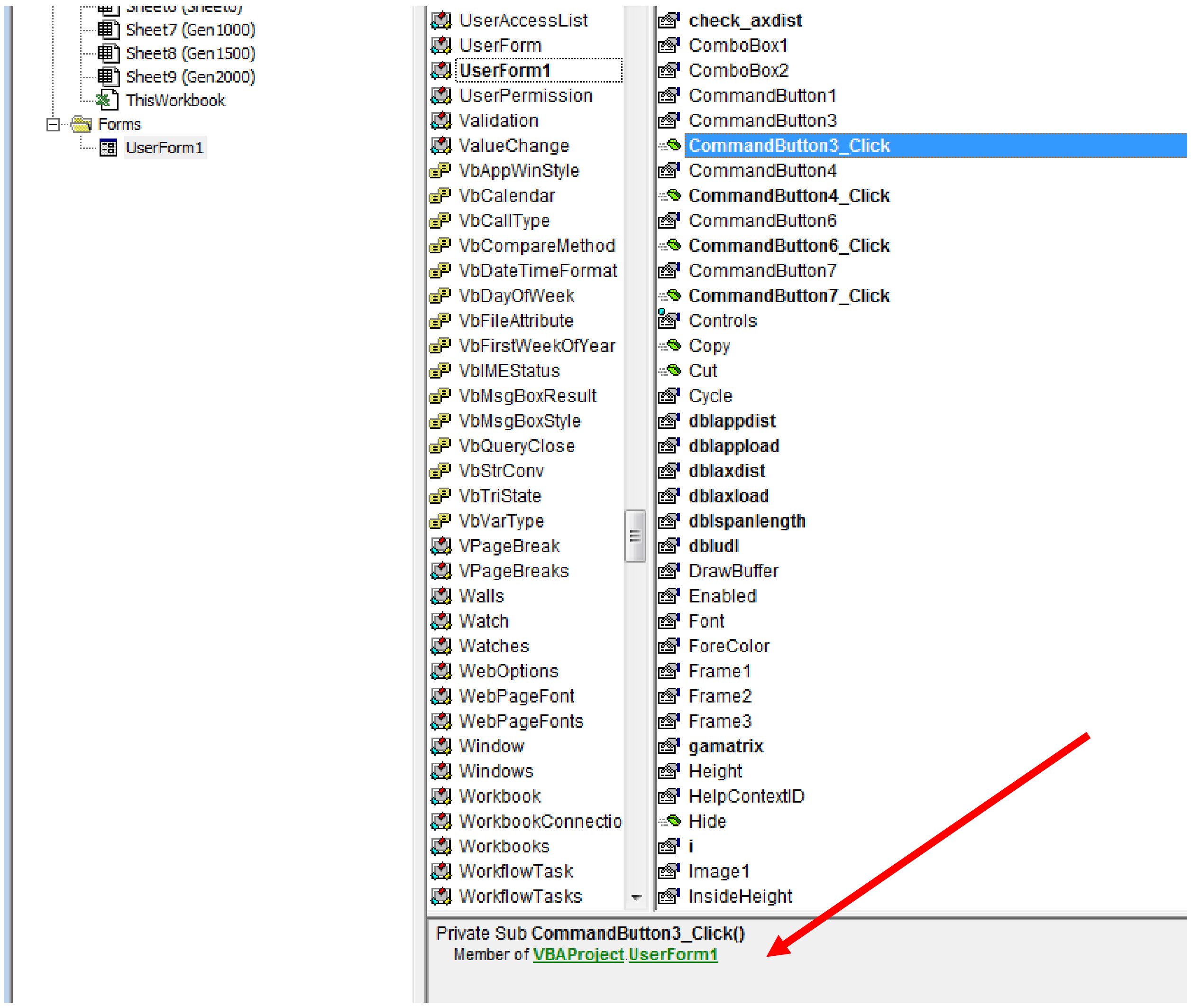
Review the Object Browser.



Look for and select an object that you created.
In the **Members** window, review the properties, methods, functions, procedures (that you created) etc., that are associated with this object.

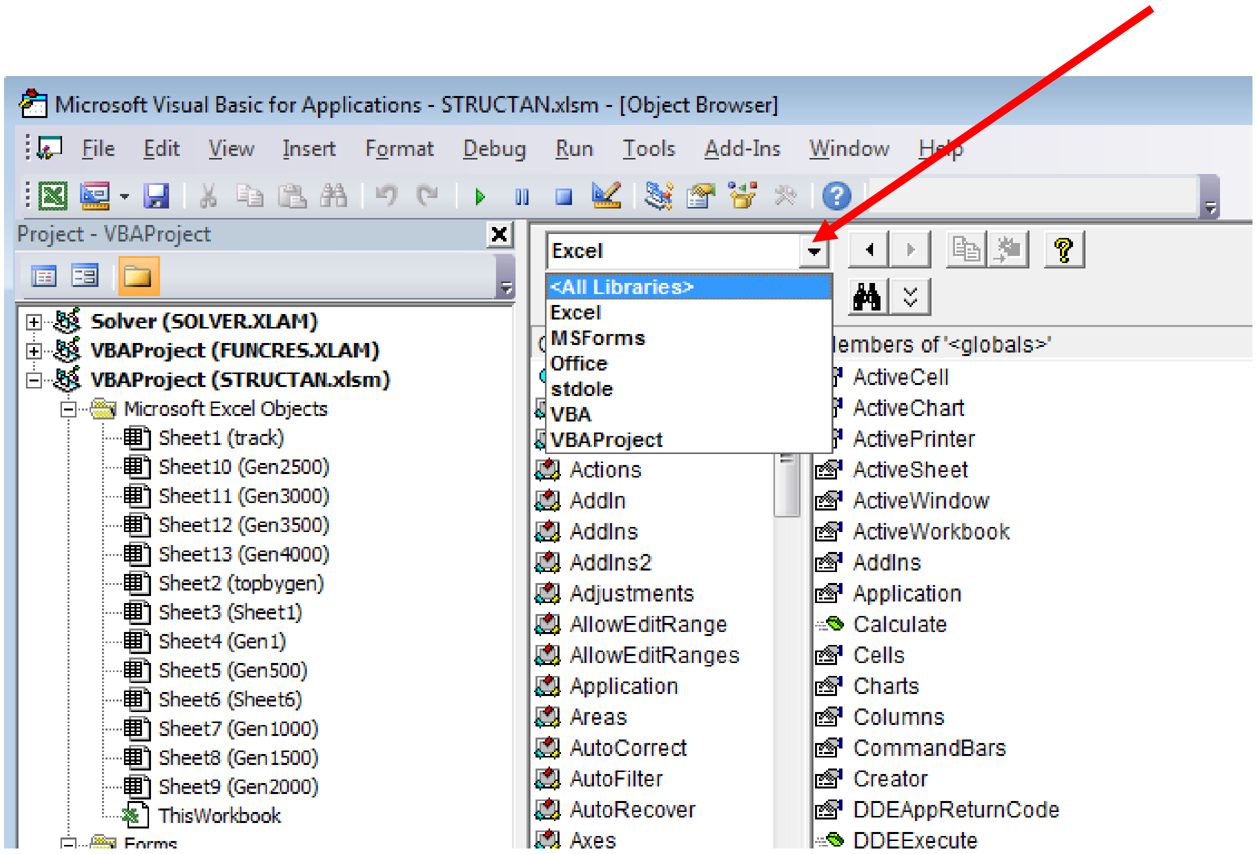


Select a sub procedure that you created.
Review a brief description at the bottom of the Object Browser.

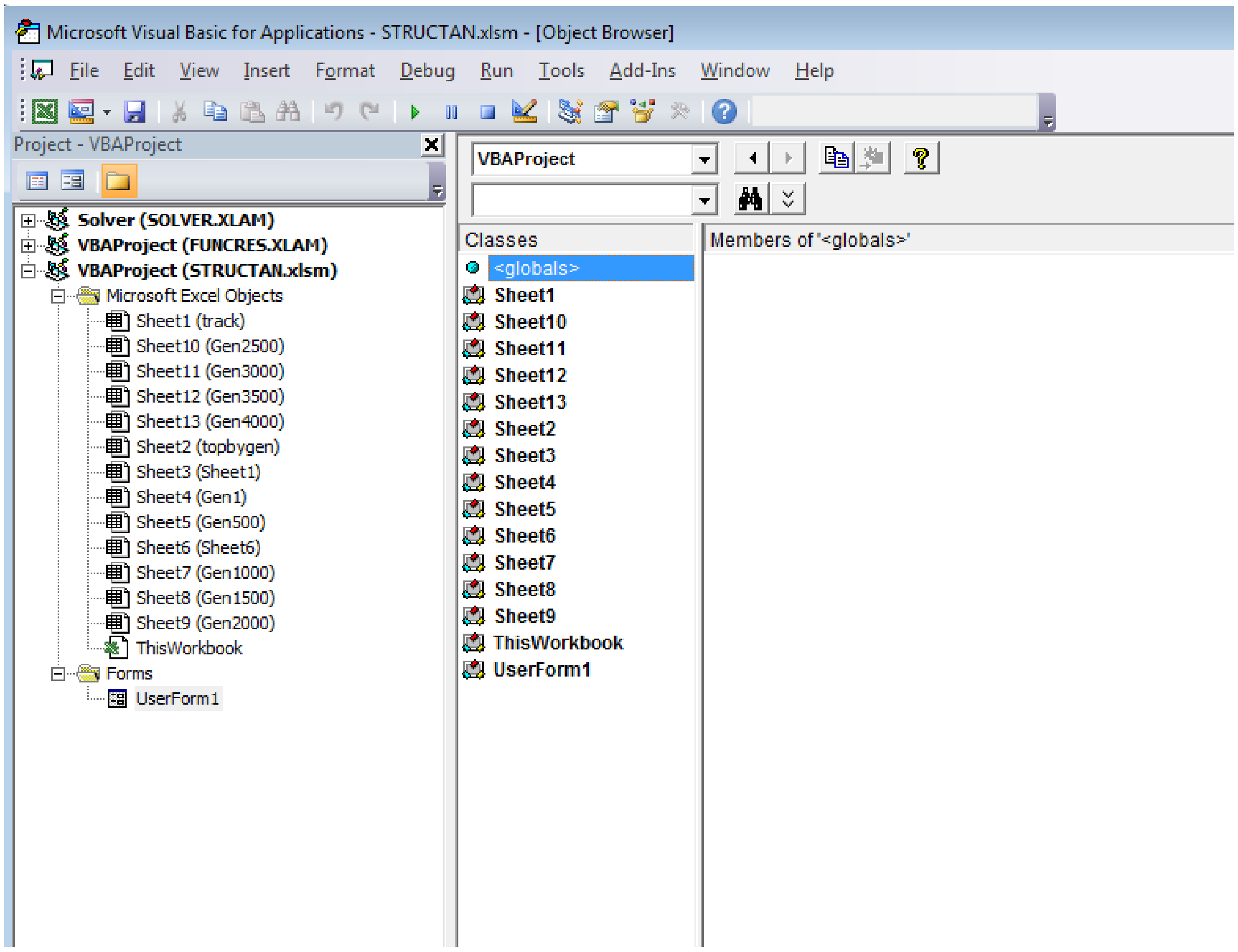


Private Sub CommandButton3_Click()
Member of VBAProject.UserForm1

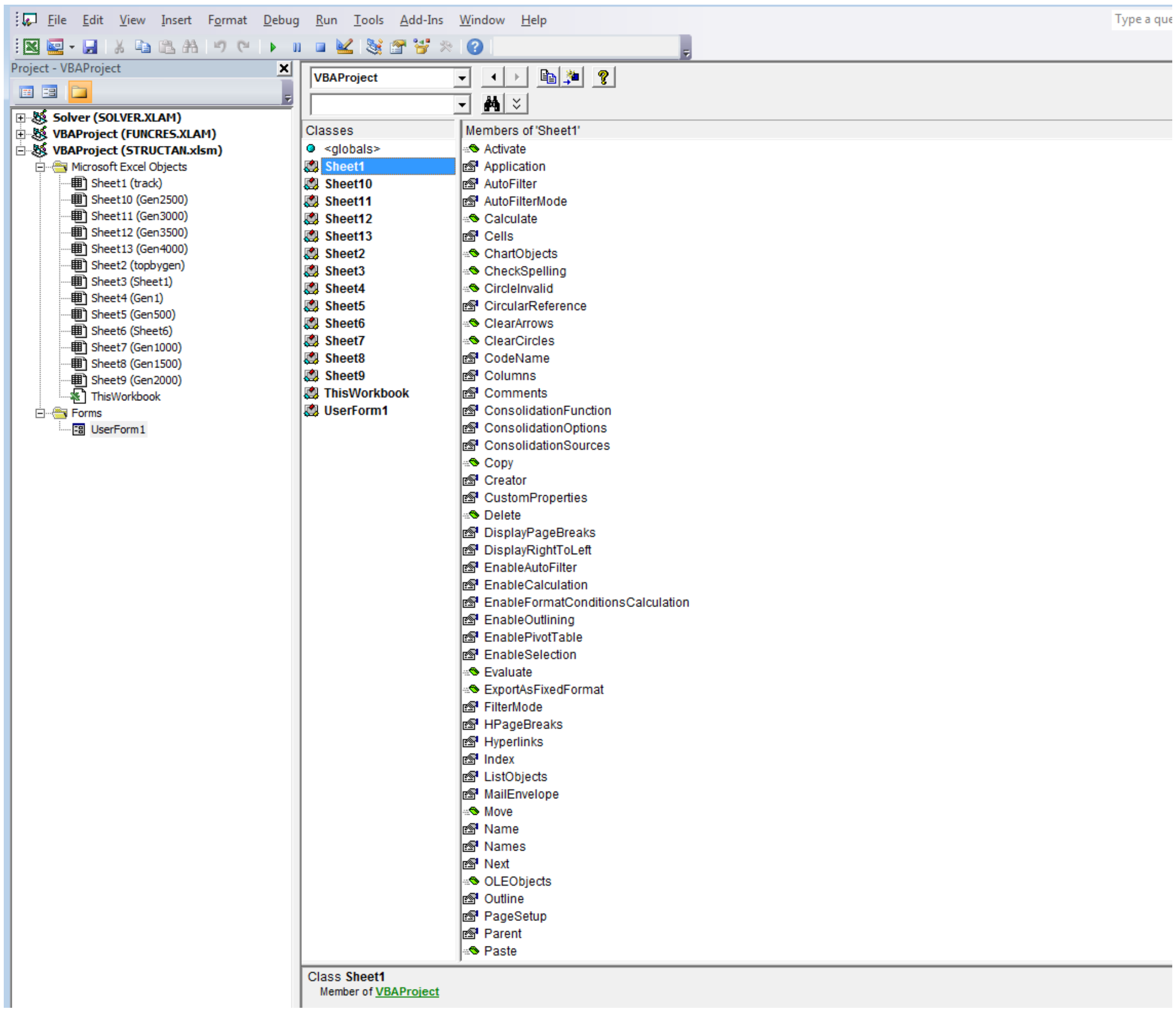
Click on the Libraries/ Project drop down.



Select *VBAProject* to review the library associated with your specific project.



Select an object to see its members.



Browsing through the libraries reveals the vast capabilities of the VBA language.

4. CONCLUSION

This course has presented a broad overview of fundamental concepts and principles of computer programming, and presented them in situations encountered by practicing engineers and scientists. All codes were developed using the *Visual Basic for Applications* (VBA) programming language.

In this course, topics on functions and procedures, and working with string variables were covered. A tour of the VBA object library was also presented. Several examples relevant to engineering were used to illustrate and demonstrate the concepts and methods learned in this class. Two mini-projects were used to demonstrate the programming concepts and methods in situations encountered by practicing engineers.

This course has enabled participants to identify situations where programming is relevant and will be of advantage to the professional. Practitioners are strongly encouraged to look out for situations in their domains of expertise where programming solutions are applicable and will be of benefit to their work and their organization.

Computer programming requires a careful and meticulous approach, and can only be mastered and retained by practice and repetition.

Good Luck and Happy Programming.

REFERENCES

Bradley, J. C., & Millspaugh, A. C. (1999). *Programming in Visual Basic 6.0.* Irwin McGraw-Hill.

FunctionX Inc. (2013). *VBA for Microsoft Office Excel 2001.* Retrieved December 21, 2013, from FunctionX Tutorials: www.functionx.com/

Microsoft. (2013). *Excel 2013 developer reference*. Retrieved October 15, 2013, from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/ee861522.aspx

Images were all drawn/ prepared by K. Ofosu