Programming MS Excel in Visual Basic (VBA)
Part 2-Branching & Looping, Message Boxes & Alerts

by

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

This course is the second of a four-part series on computer programming in *Excel Visual Basic for Applications* (VBA), tailored to practicing engineers. In this course the topics, conditional statements, message boxes and alerts, and looping structures are 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 is not required as a pre-requisite 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 of this series.
# TABLE OF CONTENTS

Abstract ........................................................................................................................................... 2  
List of Tables .................................................................................................................................. 4  
List of Figures .................................................................................................................................. 5  

1. CONDITIONAL STATEMENTS .............................................................................................. 6  
   1.1 Definition .............................................................................................................................. 6  
   1.2 If-Then-Else Statement ......................................................................................................... 6  
   1.3 Logical Operators .................................................................................................................. 7  
   1.4 Composite Conditional Expressions ..................................................................................... 7  
   1.5 Nested Conditional Statements ............................................................................................. 9  
   1.6 Select-Case Statement ......................................................................................................... 10  

2. MESSAGE BOXES AND ALERTS ........................................................................................ 15  
   2.1 Message Box ....................................................................................................................... 15  
   2.2 Simple Message Box ........................................................................................................... 16  
   2.3 Message Box with Options ................................................................................................. 22  
   2.4 Manipulating Style Value ................................................................................................... 26  
   2.5 Third VBA Project .............................................................................................................. 29  

3. LOOPING ................................................................................................................................. 62  
   3.1 For-Next Loop .................................................................................................................... 62  
   3.2 Nested Loops ...................................................................................................................... 64  
   3.3 Do-Loop .............................................................................................................................. 65  
   3.4 Exiting a Loop ..................................................................................................................... 67  
   3.5 Fourth VBA Project ............................................................................................................ 67  

4. CONCLUSION ......................................................................................................................... 94  

REFERENCES ............................................................................................................................. 95
List of Tables

Table 1: Logical operators .............................................................................................................. 8
List of Figures

Figure 1: Algorithm for Third VBA Project................................................................................. 42
Figure 2: Plan for Fourth VBA Project.......................................................................................... 72
1. CONDITIONAL STATEMENTS

1.1 Definition

A conditional statement is a feature of a programming language that executes different instructions (lines of code) based on whether some condition is met. Conditional statements enable the programmer to control the way an application interacts with the user. Conditional statements are often referred to as Branching in some texts, as they provide a means for a program to branch off in some direction or the other as some condition(s) is checked for and met, and the program then proceeds in the relevant direction(s).

1.2 If-Then-Else Statement

The most common conditional statement is the If–Then-Else statement. If the specified condition is met, a block of code will be executed, else a different block of code will be executed. In VBA the syntax is as follows

\[
\text{If condition Then} \\
\quad \text{Run this code} \\
\text{Else} \\
\quad \text{Run that code} \\
\text{End If}
\]

An alternate format allows the Else condition code to be placed on the same line as Else word, as follows:

\[
\text{If condition Then} \\
\quad \text{Run this code} \\
\text{Else:} \quad \text{Run that code} \\
\text{End If}
\]

For more than two conditions, the ElseIf condition(s) is added as follows
If condition 1 Then
    Run code 1
ElseIf condition 2 Then
    Run code 2
ElseIf condition 3 Then
    Run code 3
    
ElseIf condition (n-1) Then
    Run code (n-1)
Else
    Run code n
End If

As with the fundamental two-condition set up, the Else statement may be on the same line as the relevant code but separated from the code by a colon. In all of the above cases the Else statement is optional.

1.3 Logical Operators

The condition in the condition statement is a logical expression where a logical operator (also called a Boolean operator) is applied to compare, evaluate, or check that the inputs (called operands) meet the specified condition and give a result of “TRUE”, based upon which the relevant block of code will execute.

Examples of logical operators supported in VBA are shown in Table 1.

1.4 Composite Conditional Expressions

Conditional expressions may be combined using the “And” and/ or “Or” operators to form a composite conditional expression.
Table 1: Logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Checks if the values of two operands are equal or not. If true, then the condition is TRUE, otherwise it is FALSE.</td>
<td>If $X = Y$ Then.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Checks if the values of two operands are not equal. If the values are not equal, then the condition is TRUE, otherwise it is FALSE.</td>
<td>ElseIf $p &lt;&gt; q$ Then.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Checks if the value of the left operand is greater than the value of the right operand. If true, then the condition is TRUE, otherwise it is FALSE.</td>
<td>If $m &gt; n$ Then.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Checks if the value of the left operand is less than the value of the right operand. If true, then the condition is TRUE, otherwise it is FALSE.</td>
<td>ElseIf $x &lt; y$ Then</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Checks if the value of the left operand is greater than or equal to the value of the right operand. If true, then the condition is TRUE, otherwise it is FALSE.</td>
<td>If $a &gt;= b$ Then</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Checks if the value of the left operand is less than or equal to the value of the right operand. If true, then the condition is TRUE, otherwise it is FALSE.</td>
<td>ElseIf $q &lt;= r$ Then</td>
</tr>
<tr>
<td>IsNumeric</td>
<td>Checks if an operand is a numeric value. If yes, then the condition is TRUE, otherwise it is FALSE.</td>
<td>If IsNumeric(TextBox2.value) Then</td>
</tr>
<tr>
<td>IsNull</td>
<td>Checks if an operand is a null value (empty). If yes, then the condition is TRUE, otherwise it is FALSE.</td>
<td>If IsNull(TextBox2.value) Then</td>
</tr>
</tbody>
</table>
For example, consider a bank account that has been overdrawn. If another charge comes in and the bank pays it, the account goes further into the negative and is charged an overdraft penalty for that transaction. However, if a deposit comes in that partially clears the deficit, even though the account is still in the negative, the account is not charged an overdraft fee for that transaction. Therefore, using the negative sign for a charge transaction and positive sign for a deposit, the overdraft penalty fee is applied as follows:

\[
\text{If } \text{balance} < 0 \text{ And transaction} < 0 \text{ Then}
\]
\[
\text{newbalance = balance + transaction - fee}
\]
\[
\text{ElseIf } \text{balance} < 0 \text{ And transaction} > 0 \text{ Then}
\]
\[
\text{newbalance = balance + transaction}
\]
\[
\text{End If}
\]

1.5 Nested Conditional Statements

A \textit{nested conditional statement} is a conditional statement placed within another conditional statement. The bank account example can be rewritten using nested conditions as follows:

\[
\text{If transaction} < 0 \text{ Then}
\]
\[
\text{If balance} < 0 \text{ Then}
\]
\[
\text{newbalance = balance + transaction - fee}
\]
\[
\text{Else}
\]
\[
\text{End If}
\]
\[
\text{ElseIf transaction} > 0 \text{ Then}
\]
\[
\text{newbalance = balance + transaction}
\]
\[
\text{Else}
\]
\[
\text{End If}
\]

In each case, the \textit{If-ElseIf-Else-End} syntax for each conditional statement must be complete on its own regardless of whether it is nested or not. For instance, in the above example, if the \textit{End If} of the nested \textit{If} statement was omitted, the syntax would be incorrect and a compiler error would occur. A common format to keep track of this, as demonstrated in the above example, is by typing the code such that the \textit{If-ElseIf-Else-End If} for a specific \textit{If} statement are aligned vertically.
and that of any nested statements are offset laterally from the main statement in which they are nested. This is called **indenting** the code.

The choice, relevance, or advantage of nesting versus composite conditions must be determined by the programmer based on the specific objectives and requirements of the application.

### 1.6 Select-Case Statement

This is an alternate method to the *If–Then–Else*. It is advantageous to use when there are too many conditions and the *If-Then-Else* statement becomes cumbersome and difficult to follow and keep track of. The syntax is as follows:

```
Select Case variablebeingchecked
    Case variablevalue1
        Run code 1
    Case variablevalue2
        Run code 2
    : 
    : 
    Case Else
        Run code
End Select
```

The *Select-Case* format may also have combined logical expressions, and may involve nesting.
### Example 1:
Review the code for a grade calculator for an Engineering professor. A score of 95 and above is +A, from 90 to 91.999 is -A, from 85 to 89.999 is +B, and so on, anything less than 60 is an F. The professor enters the grade in a cell on the spreadsheet and presses the button which fires the code that checks the score and assigns the grade and a comment, and displays them back on the spreadsheet.

![Spreadsheet Image]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SCORE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A-</td>
<td>Excellent</td>
<td></td>
<td>CALCULATE GRADE</td>
</tr>
</tbody>
</table>

www.SunCam.com    Copyright © 2012 Kwabena Ofosu, Ph.D., P.E., PTOE  Page 11 of 95
The code using *If-Then-Else* is as follows:

```vbnet
Option Explicit

Private Sub CommandButton1_Click()
    Dim mark As Integer
    Dim grade, strComment As String

    mark = Sheets("Sheet5").Cells(2, 1).Value

    If mark <= 100 And mark >= 95 Then
        grade = "A"
        strComment = "Fantastic"
    ElseIf mark < 95 And mark >= 90 Then
        grade = "A-"
        strComment = "Excellent"
    ElseIf mark < 90 And mark >= 85 Then
        grade = "B+
        strComment = "Very Good"
    ElseIf mark < 85 And mark >= 80 Then
        grade = "B"
        strComment = "Good"
    ElseIf mark < 80 And mark >= 70 Then
        grade = "C"
        strComment = "Satisfactory"
    ElseIf mark < 70 And mark >= 60 Then
        grade = "D"
        strComment = "Pass"
    Else
        grade = "F"
        strComment = "Fail"
    End If

    Sheets("Sheet5").Cells(3, 2).Value = grade
    Sheets("Sheet5").Cells(3, 3).Value = strComment
End Sub
```
Example 2:
Alternately the *Select-Case* conditional statement may be used. In this example a simplified grading system is used. 90 and above is A, 80 to 89.99 is B, 70 to 79.99 is C, 60 to 69.99 is D, and anything else is F.
The VBA code for the click button is as follows:

```
Option Explicit

Private Sub CommandButton1_Click()

    Dim mark As Double
    Dim grade As String

    mark = Cells(2, 1).Value

    Select Case mark
        Case 0 To 59.999
            grade = "F"
            Cells(2, 2) = grade
        Case 60 To 69.999
            grade = "D"
            Cells(2, 2) = grade
        Case 70 To 79.999
            grade = "C"
            Cells(2, 2) = grade
        Case 80 To 89.999
            grade = "B"
            Cells(2, 2) = grade
        Case 90 To 100
            grade = "A"
            Cells(2, 2) = grade
        Case Else
            grade = "NA"
            Cells(2, 2) = grade
    End Select
```

Note that the spreadsheet object name was not prefixed to the sheet-cell identifier. This is permitted if the application involves only one spreadsheet. The same holds for form controls and the form name prefix.

Composite logic operator

“Case Else” may have been omitted as there is no code for it, and it is not relevant.
2. MESSAGE BOXES AND ALERTS

2.1 Message Box

A message box acts like a dialog box where a user can interact with the computer. Message boxes are used to alert or prompt the user. Message boxes are able to perform actions in response to what the user selects. Message boxes can be incorporated into conditional statements.

The general structure of the code for a message box is as follows:

`MsgBox (Prompt, Style Value, Title)`

The `Prompt` is the statement that will appear across the main body of the message box. The `Style Value` relates to the buttons that will appear on the message box such “OK”, “Yes”, “No”, or “Cancel”, as well as any symbols, such as a question mark, exclamation point (for a warning message box for instance), informational sign etc. The `Title` is what appears in the title bar across the top of the message box.
2.2 Simple Message Box

A simple message box will be created in this section. The message box will be activated by clicking a push button on a spreadsheet.

From the Developer tab, select Design Mode. Applications can be built directly on a spreadsheet once in Design Mode.

Click on Insert.
Select Command Button under ActiveX Controls.
Holding down the mouse, trace the shape and size of the command button on the spreadsheet.

Release the mouse.
Right click on the new button and select **Properties**.

Change the **Caption** property to "Simple Message Box"
Right click on the command button again, this time select **View Code**.

The code window opens
In the click procedure for CommandButton1 type the following code:

```vba
Option Explicit

Private Sub CommandButton1_Click()
    MsgBox("Welcome to VBA Programming")
End Sub
```
Return the spreadsheet.
Click on **Design Mode** to deactivate it.

The command button is now “live’.
Click on the **Simple Message Box** button
2.3 Message Box with Options

Click on **Design Mode** to activate it.
Select the Message Box.
Right click on it, select Copy

Click on a cell near the Message Box,
Right click, select Paste
Drag the new command button to reposition and align it with the previous one.
Right click on the new command button.
Select Properties.

Change the Caption property to “Message Box With Options”
Double click on the Message Box with Options and type in the following code in its click procedure:

```vba
Option Explicit

Private Sub CommandButton1_Click()
    MsgBox("Welcome to VBA Programming")
End Sub

Private Sub CommandButton2_Click()
    Dim message As String
    message = MsgBox("Answer this question before you login: Are you having fun?", vbYesNoCancel + vbQuestion, "Login")
    If message = vbYes Then
        Cells(18, 1).Value = "YES THIS IS GREAT!"
    ElseIf message = vbNo Then
        Cells(18, 1).Value = "NO THIS VBA STUFF IS NOT WORKING FOR ME!"
    Else
        MsgBox("You did not answer the question. Excel will now close")
        ActiveWorkbook.Save
        ActiveWorkbook.Close
    End If
End Sub
```

*If* Statement in conjunction with what user clicks on the message box. If user clicks on Yes on the message box, or No, the corresponding line of code will execute.

Prompt: Answer this question before you login: Are you having fun?

Title: Login

Style Value: vbYesNoCancel + vbQuestion

Code to save and close the Excel file if user selects to Cancel
Return to the spreadsheet.
Click on **Design Mode** to deactivate it.
Click on the **Message Box with Options**

Answer “Yes”

Review code to see where this came from
2.4 Manipulating Style Value

Copy and paste a third command button. 
Change the Caption property to “Message Box with Yes No Only” 
Reposition and align as needed. 
In the code window, in the click procedure for this button, type the following code:

```vbnet
Private Sub CommandButton3_Click()

Dim message As Integer

message = MsgBox("Click Yes to Continue, No to Stop", vbYesNo, "Login")
If message = 6 Then Range("D25").Value = "You may proceed"
ActiveWorkbook.Activate
ElseIf message = 7 Then ActiveWorkbook.Save
ActiveWorkbook.Close

End If

End Sub
```

Style Value
Alternate conditional statement setup for message box
Test the button.
The button is working successfully.
2.5 Third VBA Project

Problem Statement:
Develop an application a city traffic engineer can use to determine whether it is justified to install speed humps on a residential street. There have been complaints of speeding and cut-through traffic. The applicable laws, rules, and industry guidelines are as follows:

a) The street must be owned and maintained by the City
b) The road classification must be local access street.
c) The number of lanes cannot be more than 2.
d) The posted speed limit must be 40 mph or less.
e) The average daily traffic must be more than 500 vehicles per day
f) The 85th percentile speed must at least 12 mph over the posted speed limit
g) At least 65% of all traffic is cut-through traffic.

For the speed humps to be approved, a) through e) must apply in addition to a combination of e) and f) or e) and g).

The engineer will be using this tool on a laptop in the field. To facilitate the data entry in the field, the engineer prefers to be able to select the applicable criteria from drop down menus rather than typing in all the information. The engineer wants to have a click button which will cause a text box to indicate that the speed humps are or approved or not approved. A pop up will then inform the engineer that all commands have been executed to completion.

Solution:
Based on the criteria and the requirements for this tool, the form will have 7 combo boxes (drop downs), and a text box to display the result. Add two more text boxes so the engineer can enter his/ her name and date. Give the form a caption such as “Neighborhood Traffic Analyst” and add a logo onto the form.

The framework proposed is as follows:
The inputs in the evaluation for speed humps will come predominantly from drop down menus (combo boxes) as requested by the engineer. The form has been split into four sections using the **Frame** control. All buttons pulled onto a Frame can be moved around as one block by clicking and dragging on the Frame. The frames provide clarity by enhancing organization of the controls and being visually appealing. The Caption property of a frame is used to change the wording on the top of the frame.

On clicking the Run button the calculations will execute and a result indicating that speed humps are approved or otherwise will appear in the Result text box in the Analysis frame. One frame holds the company logo. This is an image control dragged onto and sized to fill the frame. By selecting the blank image control and going to its property **Picture**, the programmer can navigate to a picture on the computer and select it to be assigned to the image control.
Combo Boxes
The combo boxes are currently empty. To populate them, code has to be written to load a drop down list of items. This can be done in a number of ways. One common approach is to add the code to the event of opening or activating the main form. The syntax is

UserFormname.ComboBoxname.AddItem “the item”

This code will be repeated for each item in the drop down list. The process will then be repeated for each combo box on the form as follows:
Maintenance Responsibility:
Double click on the form to open the code window. The default event for the form is **Click**, change it to **Activate** in the event menu, or in the code window, simply type over the “Click” to replace it with “Activate”.

```
Option Explicit

Private Sub CommandButton2_Click()

  'Application.Visible = True
  ThisWorkbook.Save
  'Unload Userform

  Application.Quit

End Sub

Private Sub UserForm_Click()

End Sub
```
Now add the list items for Maintenance Responsibility as follows:

```vba
Private Sub UserForm_Activate()
    'when the form activates (opens) fill the list items in the combo boxes
    'maintenance responsiblity
    UserForm1.ComboBox1.AddItem "City"
    UserForm1.ComboBox1.AddItem "County"
    UserForm1.ComboBox1.AddItem "State"
    UserForm1.ComboBox1.AddItem "Federal Agency"
End Sub
```
Save your work, as a Macros-Enabled Excel Workbook.
Test the drop down.
Run the form and select the Maintenance Responsibility drop down.
Select a drop down item to ensure it populates the box without any error messages. This has been a success.
The process will be repeated for all the other combo boxes. Append the code to the Maintenance Responsibility code in the UserForm Activate event.

Road Classification:

```vbnet
Private Sub UserForm_Activate()
    'when the form activates (opens) fill the list items in the combo boxes
    'maintenance responsibility
    UserForm1.ComboBox1.AddItem "City"
    UserForm1.ComboBox1.AddItem "County"
    UserForm1.ComboBox1.AddItem "State"
    UserForm1.ComboBox1.AddItem "Federal Agency"

    'Road Classification
    UserForm1.ComboBox2.AddItem "Principal Arterial"
    UserForm1.ComboBox2.AddItem "Minor Arterial"
    UserForm1.ComboBox2.AddItem "Other Arterial"
    UserForm1.ComboBox2.AddItem "Major Collector"
    UserForm1.ComboBox2.AddItem "Minor Collector"
    UserForm1.ComboBox2.AddItem "Local"
```
Number of Lanes:

Private Sub UserForm_Activate()
    'when the form activates (opens) fill the list items in the combo boxes

    'maintenance responsibility
    UserForm1.ComboBox1.AddItem "City"
    UserForm1.ComboBox1.AddItem "County"
    UserForm1.ComboBox1.AddItem "State"
    UserForm1.ComboBox1.AddItem "Federal Agency"

    'Road Classification
    UserForm1.ComboBox2.AddItem "Principal Arterial"
    UserForm1.ComboBox2.AddItem "Minor Arterial"
    UserForm1.ComboBox2.AddItem "Other Arterial"
    UserForm1.ComboBox2.AddItem "Major Collector"
    UserForm1 ComboBox2.AddItem "Minor Collector"
    UserForm1ComboBox2.AddItem "Local"

    'Number of lanes
    UserForm1.ComboBox3.AddItem "2"
    UserForm1.ComboBox3.AddItem "4"
    UserForm1.ComboBox3.AddItem "6"
    UserForm1.ComboBox3.AddItem "8"
Posted Speed Limit:

UserForm1.ComboBox2.AddItem "Major Collector"
UserForm1.ComboBox2.AddItem "Minor Collector"
UserForm1.ComboBox2.AddItem "Local"

'Number of lanes
UserForm1.ComboBox3.AddItem "2"
UserForm1.ComboBox3.AddItem "3"
UserForm1.ComboBox3.AddItem "4"
UserForm1.ComboBox3.AddItem "6"
UserForm1.ComboBox3.AddItem "8"

'posted speed limit
UserForm1.ComboBox4.AddItem "15"
UserForm1.ComboBox4.AddItem "20"
UserForm1.ComboBox4.AddItem "25"
UserForm1.ComboBox4.AddItem "30"
UserForm1.ComboBox4.AddItem "35"
UserForm1.ComboBox4.AddItem "40"
UserForm1.ComboBox4.AddItem "45"
UserForm1.ComboBox4.AddItem "50"
UserForm1.ComboBox4.AddItem "55"
UserForm1.ComboBox4.AddItem "60"
UserForm1.ComboBox4.AddItem "65"
UserForm1.ComboBox4.AddItem "70"
85th Percentile Speed:
This speed is calculated from the traffic data collected by the traffic monitoring device that provides the average daily traffic. If the speed of all the vehicles recorded were arranged in ascending order, one value will have 85% of all the data below it. This is the 85th percentile speed. In the United States, generally, the 85th percentile speed is used as a baseline to set the speed limit on a particular roadway.

```
UserForm.ComboBox3.AddItem "6"
UserForm.ComboBox3.AddItem "8"

' posted speed limit
UserForm.ComboBox4.AddItem "15"
UserForm.ComboBox4.AddItem "20"
UserForm.ComboBox4.AddItem "25"
UserForm.ComboBox4.AddItem "30"
UserForm.ComboBox4.AddItem "35"
UserForm.ComboBox4.AddItem "40"
UserForm.ComboBox4.AddItem "45"
UserForm.ComboBox4.AddItem "50"
UserForm.ComboBox4.AddItem "55"
UserForm.ComboBox4.AddItem "60"
UserForm.ComboBox4.AddItem "65"
UserForm.ComboBox4.AddItem "70"

' 85th percentile speed
UserForm.ComboBox6.AddItem "15"
UserForm.ComboBox6.AddItem "20"
UserForm.ComboBox6.AddItem "25"
UserForm.ComboBox6.AddItem "30"
UserForm.ComboBox6.AddItem "35"
UserForm.ComboBox6.AddItem "40"
UserForm.ComboBox6.AddItem "45"
UserForm.ComboBox6.AddItem "50"
UserForm.ComboBox6.AddItem "55"
UserForm.ComboBox6.AddItem "60"
UserForm.ComboBox6.AddItem "65"
UserForm.ComboBox6.AddItem "70"
```
Cut-Through Traffic:
This is the volume of traffic trips that do not originate on the street and do not have a location on the street as the destination for the traffic trip. These are vehicles that are using this street as a by-pass route to get to wherever they are going. In this model, cut-throughs will be expressed as a percentage of the average daily traffic volume.

```
'cut through traffic
UserForm1.ComboBox7.AddItem "0"
UserForm1.ComboBox7.AddItem "5"
UserForm1.ComboBox7.AddItem "10"
UserForm1.ComboBox7.AddItem "15"
UserForm1.ComboBox7.AddItem "20"
UserForm1.ComboBox7.AddItem "25"
UserForm1.ComboBox7.AddItem "30"
UserForm1.ComboBox7.AddItem "35"
UserForm1.ComboBox7.AddItem "40"
UserForm1.ComboBox7.AddItem "45"
UserForm1.ComboBox7.AddItem "50"
UserForm1.ComboBox7.AddItem "55"
UserForm1.ComboBox7.AddItem "60"
UserForm1.ComboBox7.AddItem "65"
UserForm1.ComboBox7.AddItem "70"
UserForm1.ComboBox7.AddItem "75"
UserForm1.ComboBox7.AddItem "80"
UserForm1.ComboBox7.AddItem "85"
UserForm1.ComboBox7.AddItem "90"
UserForm1.ComboBox7.AddItem "95"
UserForm1.ComboBox7.AddItem "100"
```
All combo boxes have the property **Locked** set to **False**. This means that the combo boxes are not locked and that users can type in entries in addition to selecting from the drop down list if they so choose.
At this stage all input data has been provided on the form for the calculation to proceed. Based on the narrative of the problem, the following algorithm models the evaluation process.

![Figure 1: Algorithm for Third VBA Project](image)

Figure 1: Algorithm for Third VBA Project
Figure 1 (continued): Algorithm for Third VBA Project
A

Yes

Enter: 85th Percentile Speed

No

>PSL+12 ?

Yes

Enter % Cut-through

No

>65% ?

Yes

Speed humps APPROVED

No

Speed humps NOT APPROVED

STOP

Figure 1 (continued): Algorithm for Third VBA Project
The evaluation process will execute upon clicking the Run button. Therefore the Run button’s click event shall contain all variables needed for the evaluation to proceed. Declare variables for each input. Note that as code is typed the VBA Library provides tips and pointers to guide the programmer.

```vba
Private Sub CommandButton1_Click()
' this is the Run button

' variables for maintenance responsibility, and roadway classification
Dim strMaintResp, strRoadClass As String

' variables for number of lanes, posted speed limit
Dim intNumLanes, intPostSpeed As Integer
```

End Sub
Note that the programmer may choose any name of their choice for variables provided they meet the variable naming rules and conventions discussed in Chapter 3. The final list of variable declarations used in this application is as follows:

```vba
End Sub

Private Sub CommandButton1_Click()
' this is the Run button

' variables for maintenance responsibility, and roadway classification
Dim strMaintResp, strRoadClass As String

' variables for number of lanes, posted speed limit
Dim intNumLanes, intPostSpeed As Integer

' variable to hold traffic volume. As this may be very large, use a Long type rather than Integer type
Dim lngAVDailyTraffic As Long

' Variables for 05th percentile and cut through percentage
' use double type as these may involve decimal values
Dim dblISPercent, dblCutInHu As Double
```
The next step is the assignment of values to the variables from the form. For example, `ComboBox1` contains the information that will be stored in the maintenance responsibility variable; `TextBox1` contains the average daily traffic, and so on. The variable assignments are follows:

```vbnet
Private Sub CommandButton1_Click()

' Variables for maintenance responsibility, and roadway classification
Dim strMaintResp, strRoadClass As String

' Variables for number of lanes, posted speed limit
Dim intNumLanes, intPostSpeed As Integer

' Variable to hold traffic volume. As this may be very large, use
' a Long type rather than Integer type
Dim lngAvgDailyTraffic As Long

' Variables for 85th percentile and cut through percentage
' use double type as these may involve decimal values
Dim dbl85Percent, dblCutThu As Double

' Variable assignments
strMaintResp = UserForm1.ComboBox1.Value
strRoadClass = UserForm1.ComboBox2.Value
intNumLanes = UserForm1.ComboBox3.Value
intPostSpeed = UserForm1.ComboBox4.Value
lngAvgDailyTraffic = UserForm1.TextBox1.Value

dbl85Percent = UserForm1.ComboBox6.Value
dblCutThu = UserForm1.ComboBox7.Value

End Sub
```
The speed humps will not be approved if the street is not under City maintenance responsibility. It will not be approved if the posted speed limit exceeds 40 mph etc. These requirements or conditions must be implemented using conditional statements with the appropriate logical expressions. Note that the string variables must have their values wrapped in double quotation marks whereas the numerical variables do not.

```
If strMaintResp <> "City" Or strRoadClass <> "Local" Or _
    intNumLanes > 2 Or intPostSpeed > 40 Then
    UserForm1.TextBox2.Value = "SPEED HUMPS NOT APPROVED"
Else
    If lngAvDailyTraffic > 500 And (dbl185Percent - intPostSpeed) > 12 Then
        UserForm1.TextBox2.Value = "SPEED HUMPS APPROVED"
    ElseIf lngAvDailyTraffic > 500 And dblCutThu > 65 Then
        UserForm1.TextBox2.Value = "SPEED HUMPS APPROVED"
    End If
End If
```

Note the use of a nested If statement in addition to the composite logical statements. Think of other ways this can be accomplished.

Note the use of a line continuation at the end of the first line. This is used if the line of code becomes excessively long. A line continuation is created in the following way:

Hit the spacebar.
Type an underscore.
Hit Enter.

It may happen that a user may forget to enter information in a box where it is required. This may cause the program to produce a run-time error. To prevent this, code must be added that will alert
the user that there is missing data in the process and will terminate the procedure, enabling the user to go back, review, and make the necessary corrections.

Add an *If* statement that checks if there are any controls (or variables) with null values, and if so alert the user with a message box and prematurely exit the procedure. This block of code shall be placed anywhere before the variable assignments.

```vbnet
Dim InduVudlyIIddc AS Long

'Variables for 85th percentile and cut through percentage
'use double type as these may involve decimal values
Dim db105Percent, dblCutThu As Double

'check there are no missing data
   Or UserForm1.TextBox1.Value = "" Then
   MsgBox "Please fill out all input requirements.", vbOKOnly + vbExclamation, "Missing Data"
   'exit the procedure and start all over
   Exit Sub
End If

'variable assignments
strMaintResp = UserForm1.ComboBox1.Value
strEndClass = UserForm1.ComboBox2.Value
```
Add a message box at the very end of the procedure alerting the user that the program has run to completion.

```vba
' the evaluation process
If strMaintResp <> "City" Or strRoadClass <> "Local" Or _
    intNumLanes > 2 Or intPostSpeed > 40 Then
    UserForm1.TextBox2.Value = "SPEED HUMPS NOT APPROVED"
Else
    If lngAvDailyTraffic > 500 And (dbl185Percent - intPostSpeed) > 12 Then
        UserForm1.TextBox2.Value = "SPEED HUMPS APPROVED"
    ElseIf lngAvDailyTraffic > 500 And dblCutThu > 65 Then
        UserForm1.TextBox2.Value = "SPEED HUMPS APPROVED"
    End If
End If

' use message box to alert user that the evaluation process is over
MsgBox "ALL COMMANDS EXECUTED", vbOKOnly, "COMPLETE"
```

End Sub
The program may now be debugged and compiled before testing.
A compile error is detected and highlighted. The traffic volume variable is being assigned the value in `Textbox4` which does not exist. This is a typo and needs to be changed to `TextBox1`.

The error is corrected and the application re-saved.
Testing may now proceed. Click the Run button.
Fill out the form in its entirety. Click the Run button to conduct the evaluation.
The test is a success. All requirements and features requested by the engineer have been accomplished.
The Exit Button:
The Exit button shall close out the form and close out of Excel entirely when clicked on.
In design time select the Exit button.
Click on **View code**.
The syntax to close an open Excel workbook is as follows;

```
End Sub

Private Sub CommandButton2_Click()
   ThisWorkbook.Save
   'Unload UserForm
   Application.Quit
End Sub
```

Saves current workbook
Closes current form and returns to spreadsheet. In this case not being used
Closes out of Excel completely
As seen, this tool opens upon activating it through the Excel program. It also opens with the Excel program in the back ground. The client wants this tool to open automatically and as a stand-alone, with no other program open in the background. To write the code for this, open code window for this Excel workbook as follows:

Double click on **ThisWorkbook** in the **Project Window**.
In the default **Open** procedure enter the following code

```
Option Explicit

Private Sub Workbook_Open()
    Application.Visible = False
    UserForm1.Show

End Sub
```

Therefore on clicking to open this Excel file, once it opens, the Excel background will become invisible, and then the form will open.

Save and close out of Excel
Navigate to the folder in which this file resides.
Double click on the file to open it.
The form opens with the Excel spreadsheet background deactivated.
If it ever becomes necessary to reactivate the Excel background for example to update some codes, perform the following:
Save and close out of Excel.
Open the folder containing the file.
Hold down the **Shift** key
Double click on the file to open it
The full Excel view is opened.

The test is a success. This project has been completed to the satisfaction of the client.
3. LOOPING

Looping is a procedure in a programming language that performs repetitive (iterative) tasks. The loop is a sequence of instructions that is executed repeatedly while or until some condition is met or satisfied. Looping is fundamental to all programming languages. Most programming languages have numerous looping constructs. VBA has two types loops based on whether the number of iterations is known beforehand or otherwise.

3.1 For-Next Loop

In VBA this type of loop is used when it is pre-known how many times the loop will be repeated. The syntax is

\[
\text{For loopvariable 1 To } n \\
\text{Code that is to be repeated } n \text{ times} \\
\text{Next loopvariable}
\]

The \textbf{loop variable} (or \textbf{loop counter variable}) is an integer or long type. It will start at 1, and the code will run. The \textit{Next} will increment the loop variable to 2 and send it back to the \textit{For} line where it will be checked against the \textit{n} value. If it is less than or equal to \textit{n} the next iteration of the code proceeds, if it is greater than \textit{n} the loop has run to completion and the code will not be repeated further. The cursor then moves to the line after the \textit{Next}.

The loop variable call on the \textit{Next} line is optional in VBA, and is by and large omitted or made a comment. If the increment needed is to be in steps of 2 or more the code is as follows

\[
\text{For loopvariable 1 To } n \text{ Step } m \\
\text{Code that is to be repeated up to } n/m \text{ times} \\
\text{Next}
\]

In this case the loop variable will jump from 1 to \(1 + m\), and so on, up to the value of \(n\).
Loops can also be set up to run “backwards”, for example

\textit{For loopvariable n To 1 Step -1}

\textit{Code that is to be repeated n times but in reverse order}

\textit{Next}

\textbf{Example:}
In this example the main body of the loop is such that the value of the loop variable is displayed in a cell on the spreadsheet in the column 1 (or A). The cell row designator is the loop variable \(i\), therefore each time \(i\) increments, the display location on the spreadsheet will jump to the next row below, hence the results filling down the spreadsheet. Change the number of iterations to 10,000 and see what happens.

![Spreadsheet with VBA code and loop example](image-url)
3.2 Nested Loops

A nested loop is a loop inside of another loop.

Example:
Fill cells A18 through E35 with a value which is calculated by addition of the cell’s row number and its column number.
3.3 Do-Loop

If the number of iterations needed is not known *a priori*, then the *For-Next* loop cannot be used. In that case a *Do-Loop* structure will have to be used. In the *Do-Loop*, the programmer sets conditions upon which the looping will terminate but will not know how much iteration will actually run until termination occurs.

The syntax is as follows:

*Do while* termination condition

    Code to be repeated

*Loop*

The termination condition is generally some logical expression. After each iteration, the termination condition will be checked. If it has been met the loop will stop and the cursor moves to the line after the “Loop” line, otherwise it will move to the next iteration of the code in the main body of the loop.

The termination condition must be chosen carefully and studied closely otherwise the program may fall into an infinite loop. An infinite loop is a loop that lacks a functioning exit routine. As a result the loop repeats continuously until the operating system senses the issue and terminates the program, or until some event, for instance having the program terminate automatically after a certain duration or number of iterations, occurs. Infinite loops are capable of easily crashing personal computers.

There are other styles of the *Do-Loop*, namely

* Do.........Loop While
* Do until..........Loop
* Do.........Loop until

Each style requires the termination statement be set up in a certain way that is consistent with the logic of that style.
Example:

```
Option Explicit

Private Sub CommandButton1_Click()
    Dim counter, sum As Integer

    Cells(1, 1) = "X"
    Cells(1, 2) = "Y"
    Cells(1, 3) = "X+Y"

    Do While counter < 10
        counter = counter + 1
        Cells(counter + 1, 1) = counter
        Cells(counter + 1, 2) = counter * 2

        sum = Cells(counter + 1, 1) + Cells(counter + 1, 2)
        Cells(counter + 1, 3) = sum
    Loop

End Sub
```

```
1   X   Y   X+Y
2   1   2     3
3   2   4     6
4   3   6     9
5   4   8     12
6   5  10     15
7   6  12     18
8   7  14     21
9   8  16     24
10  9  18     27
11  10 20   30
```

In this simplified example the loop variable is `counter`. Note that unlike the `For-Next` loop where the program automatically increments the loop variable, in the `Do-Loop`, the loop variable must be incremented “manually” by the programmer. However this makes the incrementing more flexible than in the `For-Next` loop. In this example the termination condition is based on the value of the counter variable, however that is not always the case. The termination condition may be based on any of the variables in the main body of the loop and based on specific requirements of the application. Do-Loops may be nested with each other and they may be nested within `For-Next` loops and vice versa. All loops may be nested in conditional statements and vice versa.
3.4 Exiting a Loop

In some cases it may be necessary to abruptly or prematurely exit a loop based on the progress of the program. The syntax is

*Exit Do*

Or

*Exit For*

The program will exit the loop at the location the *Exit* code is written. The program will exit out of the current (or most recent) loop. Typically the *Exit* command will be associated with some conditional statement nested in the loop such that if that condition is met abort the loop and move to some other area of the code or to some other relevant command. The *Exit* code can also be called to exit out of a conditional statement (*Exit If*). The advantage of that being if for example the conditional statement involves hundreds, if not thousands of conditions (including composite conditions) and related code that need to be checked or evaluated, the program will exit out once the first “True” condition is encountered, and not have to continue evaluating through the other hundreds or thousands of other conditions and instructions.

3.5 Fourth VBA Project

Problem Statement:

A Florida bridge engineer maintains a large bridge inventory with the following attributes:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bridge Number</td>
<td>County Code</td>
<td>Facility Carried by Structure</td>
<td>Features Under</td>
<td>Year Built</td>
<td>Num of Lanes</td>
<td>Ave Daily Traffic</td>
<td>Percent Truck Traffic</td>
</tr>
<tr>
<td>2</td>
<td>490805</td>
<td>57 &amp; 13 ROAD</td>
<td>415 RE / DRAINAGE DITCH</td>
<td>1967</td>
<td>1</td>
<td>25</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>124083</td>
<td>71 CRYSTAL DRIVE</td>
<td>10 MILE CANAL</td>
<td>1969</td>
<td>2</td>
<td>10800</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>115745</td>
<td>71 WINKLER AVENUE</td>
<td>10 MILE DRAIN</td>
<td>2000</td>
<td>5</td>
<td>27800</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>720253</td>
<td>31 1-295 SB (SR-9A)</td>
<td>103rd ST.</td>
<td>1970</td>
<td>3</td>
<td>51500</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>720253</td>
<td>31 1-295 SB (SR-9A)</td>
<td>103rd ST.</td>
<td>1970</td>
<td>3</td>
<td>40000</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>720346</td>
<td>31 1-295 NB (SR-9A)</td>
<td>103rd ST.</td>
<td>1970</td>
<td>3</td>
<td>51500</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>720346</td>
<td>31 1-295 NB (SR-9A)</td>
<td>103rd ST.</td>
<td>1970</td>
<td>3</td>
<td>40000</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>870437</td>
<td>86 I-95</td>
<td>119 ST</td>
<td>1963</td>
<td>10</td>
<td>190500</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>870437</td>
<td>86 I-95</td>
<td>119 ST</td>
<td>1962</td>
<td>10</td>
<td>20100</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>720132</td>
<td>51 US ALT-1 (SR-115)</td>
<td>11th ST.</td>
<td>1970</td>
<td>4</td>
<td>48500</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>720132</td>
<td>51 US ALT-1 (SR-115)</td>
<td>11th ST.</td>
<td>1970</td>
<td>4</td>
<td>40500</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>790013</td>
<td>177 US-1</td>
<td>11th St. Canal</td>
<td>1951</td>
<td>5</td>
<td>28500</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>790013</td>
<td>177 US-1</td>
<td>11th St. Canal</td>
<td>1951</td>
<td>5</td>
<td>31000</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
The inventory currently holds 15,332 records that are updated periodically by inspectors. The engineer periodically performs spreadsheet calculations for update and reporting purposes. The inspectors’ data entry requirements are different from what the engineer would prefer. For example, the County Codes are given by the US Census Code number but the engineer would prefer the name of the County. The engineer wants a button which when clicked on will run through the entire data and for each record will conduct the following manipulations and calculations:

1. Converts the County Code to the County name
2. In a separate column calculates the Age of the bridge
3. In a separate column calculates Average Daily Truck Traffic by multiplying the Average Daily Traffic by the Percent Truck Traffic

At the beginning of the program the engineer wants pop up to appear stating the number of records that are about to be manipulated and giving the engineer the options to continue or stop the process. At the end of the process a pop up shall appear notifying the engineer that all commands have been successfully executed.

As the program runs it will not be visually possible for the engineer to follow the calculations on the screen. Set up a digital dashboard which displays the following information:

1. Number of records in the inventory.
2. Current record at which calculations are taking place.
3. Number of records remaining to be processed.
Solution:
Insert a command button onto the spreadsheet (alternatively, create a form and add a command button onto the form)
Resize the command button as needed.
Double click on the command button to open the code window
Convert Census Code to Florida County name. The list of Florida counties by census code may be downloaded from the internet. An abridged list is provided.

<table>
<thead>
<tr>
<th>County name</th>
<th>FIPS code</th>
<th>County name</th>
<th>FIPS code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alachua</td>
<td>1</td>
<td>Okeechobee</td>
<td>93</td>
</tr>
<tr>
<td>Baker</td>
<td>3</td>
<td>Orange</td>
<td>95</td>
</tr>
<tr>
<td>Bay</td>
<td>5</td>
<td>Osceola</td>
<td>97</td>
</tr>
<tr>
<td>Bradford</td>
<td>7</td>
<td>Palm Beach</td>
<td>99</td>
</tr>
<tr>
<td>Brevard</td>
<td>9</td>
<td>Pasco</td>
<td>101</td>
</tr>
<tr>
<td>Broward</td>
<td>11</td>
<td>Pinellas</td>
<td>103</td>
</tr>
<tr>
<td>Calhoun</td>
<td>13</td>
<td>Polk</td>
<td>105</td>
</tr>
<tr>
<td>Charlotte</td>
<td>15</td>
<td>Putnam</td>
<td>107</td>
</tr>
<tr>
<td>Citrus</td>
<td>17</td>
<td>St. Johns</td>
<td>109</td>
</tr>
<tr>
<td>Clay</td>
<td>19</td>
<td>St. Lucie</td>
<td>111</td>
</tr>
<tr>
<td>Collier</td>
<td>21</td>
<td>Santa Rosa</td>
<td>113</td>
</tr>
<tr>
<td>Columbia</td>
<td>23</td>
<td>Sarasota</td>
<td>115</td>
</tr>
<tr>
<td>Dade</td>
<td>25</td>
<td>Seminole</td>
<td>117</td>
</tr>
<tr>
<td>DeSoto</td>
<td>27</td>
<td>Sumter</td>
<td>119</td>
</tr>
<tr>
<td>Dixie</td>
<td>29</td>
<td>Suwannee</td>
<td>121</td>
</tr>
<tr>
<td>Duval</td>
<td>31</td>
<td>Taylor</td>
<td>123</td>
</tr>
<tr>
<td>Escambia</td>
<td>33</td>
<td>Union</td>
<td>125</td>
</tr>
<tr>
<td>Flagler</td>
<td>35</td>
<td>Volusia</td>
<td>127</td>
</tr>
<tr>
<td>Franklin</td>
<td>37</td>
<td>Wakulla</td>
<td>129</td>
</tr>
<tr>
<td>Gadsden</td>
<td>39</td>
<td>Walton</td>
<td>131</td>
</tr>
<tr>
<td>Gilchrist</td>
<td>41</td>
<td>Washington</td>
<td>133</td>
</tr>
<tr>
<td>Glades</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gulf</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamilton</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardee</td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hendry</td>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hernando</td>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highlands</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hillsborough</td>
<td>57</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2: Plan for Fourth VBA Project**
Consider the bridge data is stored on a spreadsheet named “bridges”. Also, the county code number is in column “B” (column 2). So for the first record the reference to the county code number will be

\[ \text{Sheets(“bridges”).cells(2, 2).value} \]

For the county code of the next record on the spreadsheet is

\[ \text{Sheets(“bridges”).cells(3, 2).value} \]

And so on.

The county code for a record can therefore be generalized as follows

\[ \text{Sheets(“bridges”).cells(i, 2).value} \]

where \( i \) is the row number for that record.

An \textit{If} statement (or Select-Case statement) may be used to reference county code number to county name as follows:

\begin{verbatim}
If Sheets(“bridges”).cells(i, 2).value = 1 Then
    Sheets(“bridges”).cells(i, 2).value = “Alachua”
ElseIf …………………
   .
End If
\end{verbatim}

As there are 67 counties in Florida, the \textit{If} statement will have 67 conditions to check. Note that in this code, the name of the county will replace the county code number in column 2. If the intent is to post the county name in a separate column, then the column number in the \textit{cells} code must be changed to that column number and the county name will now be posted in that column at row \( i \).
The full *If* statement will therefore be of the set up:

```vba
Private Sub CommandButton1_Click()
    'this is the click button on the spreadsheet
    'assign correct county name from census code
    If Sheets("bridges").Cells(i, 2).Value = 1 Then
        Sheets("bridges").Cells(i, 2).Value = "ALACHUA"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 2 Then
        Sheets("bridges").Cells(i, 2).Value = "BAKER"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 5 Then
        Sheets("bridges").Cells(i, 2).Value = "BAY"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 7 Then
        Sheets("bridges").Cells(i, 2).Value = "BRADFORD"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 9 Then
        Sheets("bridges").Cells(i, 2).Value = "BREVARD"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 11 Then
        Sheets("bridges").Cells(i, 2).Value = "BROWARD"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 13 Then
        Sheets("bridges").Cells(i, 2).Value = "CALHOUN"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 15 Then
        Sheets("bridges").Cells(i, 2).Value = "CHARLOTTE"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 17 Then
        Sheets("bridges").Cells(i, 2).Value = "CITRUS"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 19 Then
        Sheets("bridges").Cells(i, 2).Value = "CLAY"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 21 Then
        Sheets("bridges").Cells(i, 2).Value = "COLLIER"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 23 Then
        Sheets("bridges").Cells(i, 2).Value = "COLUMBIA"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 25 Then
        Sheets("bridges").Cells(i, 2).Value = "DADE"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 27 Then
        Sheets("bridges").Cells(i, 2).Value = "DE SOTO"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 29 Then
        Sheets("bridges").Cells(i, 2).Value = "DIAMOND"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 31 Then
        Sheets("bridges").Cells(i, 2).Value = "DUVAL"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 33 Then
        Sheets("bridges").Cells(i, 2).Value = "ESCAMBIA"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 35 Then
        Sheets("bridges").Cells(i, 2).Value = "FLORIDA"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 37 Then
        Sheets("bridges").Cells(i, 2).Value = "FRANKLIN"
End If
```
Save and Compile to check for errors.
Resave.
Next declare the variables that will be used for the calculation of age and truck traffic.
(preferably at the top of the procedure)

```vba
Private Sub CommandButton1_Click()
  'this is the click button on the spreadsheet

  'variable for age of bridge
  Dim intAge As Integer

  'year bridge was built
  Dim intYearBuilt

  'variable for trick traffic
  Dim intAveDailyTruckTraffic As Integer

  'Truck volume will calculated from
  Dim intAveDailyTraffic As Long

  'and % trucks
  Dim intPercentTrucks As Integer

  'assign correct county name from census code
  If Sheets("bridges").Cells(i, 2).Value = 1 Then
      Sheets("bridges").Cells(i, 2).Value = "ALACHUA"
  ElseIf Sheets("bridges").Cells(i, 2).Value = 3 Then
      Sheets("bridges").Cells(i, 2).Value = "BARNES"
  Else
      Sheets("bridges").Cells(i, 2).Value = "BROWNS"
  End If
```
Next, assign the values of the input variables for the calculations. For example, age of bridge is the current year (2014) minus the year built. The year built is in column “E” (column 5).

```vba
Private Sub CommandButton1_Click()
    'this is the click button on the spreadsheet

    'variable for age of bridge
    Dim intAge As Integer

    'year bridge was built
    Dim intYearBuilt

    'variable for trick traffic
    Dim intAveDailyTruckTraffic As Integer

    'Truck volume will calculated from
    Dim intAveDailyTraffic As Long

    'and % trucks
    Dim intPercentTrucks As Integer

    'assign values from the data
    intYearBuilt = Sheets("bridges").Cells(i, 5).Value
    intAveDailyTruckTraffic = Sheets("bridges").Cells(i, 7).Value
    intPercentTrucks = Sheets("bridges").Cells(i, 8).Value

    'assign correct county name from census code
    If Sheets("bridges").Cells(i, 2).Value = 1 Then
        Sheets("bridges").Cells(i, 2).Value = "ALACHUA"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 2 Then
        Sheets("bridges").Cells(i, 2).Value = "BROWARD"
    End If
```
Save and Compile to check for errors.
Resave.
Next perform the calculations. For example Truck Traffic is the product of Average Daily Traffic and Percent Trucks in the traffic (divide spreadsheet value by 100).

```vba
'assign values from the data
intYearBuilt = Sheets("bridges").Cells(i, 5).Value
intAveDailyTraffic = Sheets("bridges").Cells(i, 7).Value
intPercentTrucks = Sheets("bridges").Cells(i, 8).Value

'calculate bridge age
intAge = 2014 - intYearBuilt

'calculate truck traffic
intAveDailyTruckTraffic = intAveDailyTraffic * intPercentTrucks / 100

'assign correct county name from census code
If Sheets("bridges").Cells(i, 2).Value = 1 Then
    Sheets("bridges").Cells(i, 2).Value = "ALACHUA"
ElseIf Sheets("bridges").Cells(i, 2).Value = 3 Then
    Sheets("bridges").Cells(i, 2).Value = "BAKER"
ElseIf Sheets("bridges").Cells(i, 2).Value = 5 Then
    Sheets("bridges").Cells(i, 2).Value = "BAY"
ElseIf Sheets("bridges").Cells(i, 2).Value = 7 Then
```
Save the result to the spreadsheet.
Save truck volume to column “I”, and Age to column “J”.

```vba
'assign values from the data
intYearBuilt = Sheets("bridges").Cells(i, 5).Value
intAveDailyTraffic = Sheets("bridges").Cells(i, 7).Value
intPercentTrucks = Sheets("bridges").Cells(i, 8).Value

'calculate bridge age
intAge = 2014 - intYearBuilt
Sheets("bridges").Cells(i, 10).Value = intAge

'calculate truck traffic
intAveDailyTruckTraffic = intAveDailyTraffic * intPercentTrucks / 100
Sheets("bridges").Cells(i, 9).Value = intAveDailyTruckTraffic

'assign correct county name from census code
If Sheets("bridges").Cells(i, 2).Value = 1 Then
    Sheets("bridges").Cells(i, 2).Value = "ALACHUA"
ElseIf Sheets("bridges").Cells(i, 2).Value = 3 Then
    Sheets("bridges").Cells(i, 2).Value = "BAKER"
ElseIf Sheets("bridges").Cells(i, 2).Value = 5 Then
    Sheets("bridges").Cells(i, 2).Value = "BAY"
ElseIf Sheets("bridges").Cells(i, 2).Value = 7 Then
```

"https://www.SunCam.com" Copyright © 2012 Kwabena Ofosu, Ph.D., P.E., PTOE
Save and Compile to check for errors.
Resave.
The calculations will be performed for each record (each row in the table) therefore the calculation code must be placed within a For-Next loop so as to iterate down the table for each record. The row number \( i \) must therefore be in the \textit{For} statement.
Note that it is not necessary to put the variable declaration within the \textit{For} loop, in fact it will cause a run-time error. Once the variables are declared once, they can be recycled through each iteration of the For-Next loop.

```vba
Dim intAveDailyTraffic As Long

'and & trucks
Dim intPercentTrucks As Integer

For i = 2 To 15332

'assign values from the data
intYearBuilt = Sheets("bridges").Cells(i, 5).Value
intAveDailyTruckTraffic = Sheets("bridges").Cells(i, 7).Value
intPercentTrucks = Sheets("bridges").Cells(i, 8).Value

'calculate bridge age
intAge = 2014 - intYearBuilt
Sheets("bridges").Cells(i, 10).Value = intAge

'calculate truck traffic
intAveDailyTruckTraffic = intAveDailyTruckTraffic * intPercentTrucks
Sheets("bridges").Cells(i, 9).Value = intAveDailyTruckTraffic

'assign correct county name from census code
If Sheets("bridges").Cells(i, 2).Value = 1 Then
    Sheets("bridges").Cells(i, 2).Value = "ALACHUA"
ElseIf Sheets("bridges").Cells(i, 2).Value = 3 Then
    Sheets("bridges").Cells(i, 2).Value = "BAKER"
ElseIf Sheets("bridges").Cells(i, 2).Value = 5 Then
    Sheets("bridges").Cells(i, 2).Value = "BAY"
```

Note that a *For* statement must always have a corresponding *Next* statement at the bottom otherwise the syntax is incomplete and a run-time error will occur.

```
Sub FunctionName
    For i = 1 To 10
        Select Case Sheets("bridges").Cells(i, 2).Value
            Case 123
                Sheets("bridges").Cells(i, 2).Value = "TAYLOR"
            Case 125
                Sheets("bridges").Cells(i, 2).Value = "Union"
            Case 127
                Sheets("bridges").Cells(i, 2).Value = "VOLUSIA"
            Case 129
                Sheets("bridges").Cells(i, 2).Value = "WAKULLA"
            Case 131
                Sheets("bridges").Cells(i, 2).Value = "WALTON"
            Case 133
                Sheets("bridges").Cells(i, 2).Value = "WASHINGTON"
        End Select
    Next i
End Sub
```
Save and Compile to check for errors.
Resave.
On the spreadsheet highlight the columns receiving the calculated results to facilitate review.

Now set up the digital dashboard. The purpose of the digital dashboard is to enable the user “watch” the progress of the program as it moves down the table performing the calculations, as it will be physically impossible to see the calculations in real-time as the computer monitor is not large enough to display the entire spreadsheet of 15,332 records.
The progress of the program will therefore be monitored by viewing the dashboard information.
In the project, the client requested to be able to track the total number of records, the current record being processed, and the number of records remaining to be processed, on the dashboard.
Save and Compile to check for errors.
Resave.
Code must be added to send the relevant values to the dashboard.
The number of records is known from the spreadsheet as 15,332. Report this number to the dashboard. The current record is the current value of the loop variable \(i\). Report it to the dashboard at the start (or end) of each iteration. Subtracting the current record from the total number of records will give the number of records left to be processed.

```vba
' and % trucks
Dim intPercentTrucks As Integer

'dashboard reporting
'total records
Sheets("bridges").Cells(18, 12).Value = 15332

For i = 2 To 15332

'dashboard reporting
'current record
Sheets("bridges").Cells(21, 12).Value = i
'dashboard reporting
'number of records left
Sheets("bridges").Cells(24, 12).Value = 15332 - i

'assign values from the data
intYearBuilt = Sheets("bridges").Cells(i, 5).Value
```
Next, add a message box at the end that alerts the user that the program has run to completion

    Sheets("bridges").Cells(i, 2).Value = "WAKULLA"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 131 Then
        Sheets("bridges").Cells(i, 2).Value = "WALTON"
    ElseIf Sheets("bridges").Cells(i, 2).Value = 133 Then
        Sheets("bridges").Cells(i, 2).Value = "WASHINGTON"
    End If

    Next 'i |

    MsgBox "ALL COMMANDS EXECUTED", vbOKonly, "SUCCESSFUL COMPLETION"

End Sub
It has been requested by the engineer that before the process runs a message box pops up giving a caution that a large data set is about to be processed, and give the user the option to continue or stop the process. Therefore before the *For* loop starts a message box with options must appear

```vba
' total records
Sheets("bridges").Cells(18, 12).Value = 15532

' caution user that data is very large and processing will consume significant computer resources
Dim xmessage, zmessage As String
xmessage = MsgBox("You are about to process a very large data set which may tie up computer resources" _ & vbCrLf & "Do you want to continue?", vbYesNo + vbExclamation, "CAUTION")

' so if user selects to continue, start the For loop
If xmessage = vbYes Then

For 1 = 2 To 15532

' dashboard reporting
' current record
Sheets("bridges").Cells(21, 12).Value = 1
' dashboard reporting
' number of records left
Sheets("bridges").Cells(24, 12).Value = 15532 - 1

' assign values from the data
intValueBuilt = Sheets("bridges").Cells(1, 5).Value
```
Always remember to complete an *If* statement with the *ElseIf*-Else-End *If*.

```vbnet
ElseIf Sheets("bridges").Cells(i, 2).Value = 131 Then
    Sheets("bridges").Cells(i, 2).Value = "WALTON"
ElseIf Sheets("bridges").Cells(i, 2).Value = 133 Then
    Sheets("bridges").Cells(i, 2).Value = "WASHINGTON"
End If

Next 'i

MsgBox "ALL COMMANDS EXECUTED", vbOKOnly, "SUCCESSFUL COMPLETION"

ElseIf xmessage = vbNo Then

    'this is from the xmessage question you answered after being warned you are
    'about to process a large data set.
    xmessage = MsgBox("You have chosen not to process this large data set.", vbOKOnly, "Goodbye")

End If

End Sub
```

Save and Compile to check for errors.
Resave.
Test the program.
Deactivate Design mode
Click on the command button.
Test the option where the user chooses not to proceed after the caution message box.
Click on No.
Success.
Repeat the test.  This time select **Yes** to continue after the caution message.

```vba
Dim intPercentTrucks As Integer

' dashboard reporting
'total records
Sheets("bridges").Cells(18, 12).Value = 15532

' caution user that data is very large and processing will
' consume significant computer resources
Dim xmessage, xmessage As String

xmessage = MsgBox("You are about to process a very large
& vbCrLf & "variable. Do you want to continue?",
It xmessage = vbYes Then

For I = 2 To 15532

'dashboard reporting
'current record
Sheets("bridges").Cells(21, 12).Value = I
'dashboard reporting
```

An **Overflow Error** occurs. This is a problem with a variable not being able to hold the data given to it.
Click **Debug**

The line where the error occurred is highlighted.

Study the code carefully.

```vba
For i = 2 To 15332

    'dashboard reporting
    'current record
    Sheets("bridges").Cells(21, 12).Value = i
    'dashboard reporting
    'number of records left
    Sheets("bridges").Cells(24, 12).Value = 15332 - i

    'assign values from the data
    intYearBuilt = Sheets("bridges").Cells(i, 5).Value
    intAveDailyTruckTraffic = Sheets("bridges").Cells(i, 7).Value
    intPercentTrucks = Sheets("bridges").Cells(i, 8).Value

    'calculate bridge age
    intAge = 2014 - intYearBuilt
    Sheets("bridges").Cells(i, 10).Value = intAge

    'calculate truck traffic
    intAveDailyTruckTraffic = intAveDailyTraffic * intPercentTrucks
    Sheets("bridges").Cells(i, 9).Value = intAveDailyTruckTraffic

    'assign correct county name from census code
```

The data from column 7 is actually the Average Daily Traffic, not Average Daily Truck Traffic. Average Daily Truck Traffic will be calculated subsequently. Correct the variable name to `intAveDailyTraffic`.

Click on Reset
Save and Compile to check for more bugs and errors. If none, Resave. Re-Run.
The program proceeds. The dashboard updates rapidly as the program proceeds.
Hold down the Ctrl key and press Break. (In some operating systems it is Shift + Break, or Fn + Break)
The program pauses and goes in the Break mode
Select Debug

As Integer

'declare integer
'on

'get

12).Value = 15332

very large and processing will use resources
'string

about to process a very large data set which may tie up computer resources" + & "Do you want to continue?", vbYesNo + vbExclamation, "CAUTION")

.nue, start the For loop
Click on the spreadsheet window to view it.
Review the calculated columns.
Review the digital dashboard

The results so far are satisfactory. It can be concluded that the program is working as intended.
Select **VBA window**.
Click on **Run/ Resume** to resume execution of the program from where it was paused.
The program successfully runs to completion.
Save and close the workbook.
All project requirements have been met.
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 the topics, conditional statements, message boxes and alerts, and looping structures were covered in detail. Several examples from engineering were used to illustrate and demonstrate the concepts and methods learned in this class. Two mini-projects were used to demonstrate these 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


Images were all drawn/ prepared by K. Ofosu