**9.1: Explain the Purpose of Adding VBA Code to Enhance Access**

**About VBA**

**Visual Basic for Applications** (or **VBA**) is a programming language used to automate Access, Word and Excel. VBA modules and procedures allow you to build custom applications. MS Access 2013+ uses VBA version 7.1 with the Access database engine

VBA is closely related to Visual Basic version 6 BUT it can only run code within the host application, in this case, MS Access. It is incompatible with VB.net. VBA is object-based but it is NOT object-oriented.

**Why use VBA?**

**Why did you create macros in Excel/Word? Perform some action repeatedly (automated)**

* To create user-defined function to perform complex operations we use repeatedly.
  + Some things you can do easily with macros but VBA makes it possible to customize how you manage the data and how you display it.
* To create complex data management features
  + Make transactions (ie: a number of steps that have to be done together)
* To create more sophisticated user-interface requirements
  + Provide graceful error handling
  + Provide more complex forms.

**9.2: Identify the Different Types of VBA Modules**

## VBA Modules

A VBA **module** is the container for VBA code. A module is a collection of **declarations, statements, and procedures** that are stored together as a unit.

There are four types of modules:

1. **Form module.**  Created with the form and used to respond to events triggered by forms or controls on forms. When you add code to a form object, you create a new class of object in the database. The event-handling procedures you create for the form are its new class’s methods, hence the term class module for the code modules associated with a particular form.
2. **Report module.** Created with the report and used for responding to events triggered by reports, sections of reports, or group headers and footers.
3. **Standard module.**  A standard module is created from the main database window the same way as you create any other new database object. They contain general procedures that aren’t associated with any other Access object (ie: forms/reports). The code in the standard modules can be used anywhere within your application. By default, you can think of them as global or public because they are available to all the elements in your Access application
4. **Class module.** Used to define custom objects with their own properties and methods. A class module is often bound to a form or report. Class modules will not be created in this course.

## Elements of VBA Modules

A module consists of a declarations section and one or more procedures/functions. The declarations section is used to declare variables or constants used by procedures or functions contained in the module.

**procedure** -A series of VBA statements grouped together and used by Access to complete a task. Procedures can either be event or custom procedures

**Procedure Types**

**Event procedures** – are associated with form and report events

**custom procedures** – are not assigned to events but must be called from another procedure or function

**function** – a customer procedure that returns a single value (Boolean, numeric, character, or string value)

# 9.5 Describe VBA Code Window Components

## VBA Editor

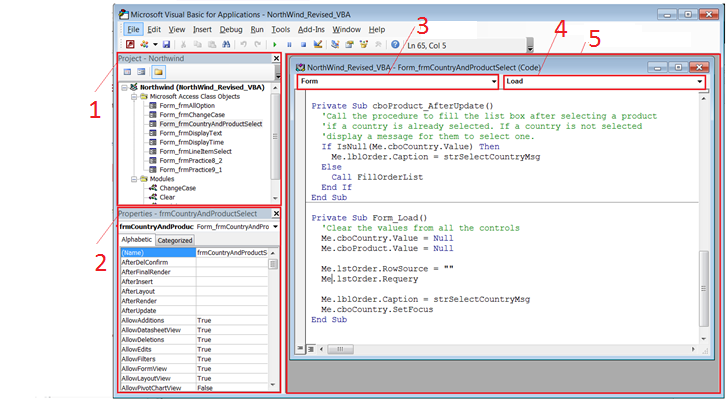
VBA functions and procedures are written in the **VBA Editor**. The editor contains a text editor similar to Notepad. It also color codes keywords and comments. It also provides context-sensitive menus that allow you to pick commands and properties from.

To view the VBA Code:

1. Open up Northwind\_VBA\_Example database
2. Open the form **frmChangeCase** and Go to Design View
3. From the Design tab, select the view code icon which is in the Tools group:

Graphical user interface, application, Word

Description automatically generated



Identify the numbered items in VBA Editor

1. **Project Explorer Window** – Here we can different modules of your project. Double click on Form\_frmDisplayText – this open up the frmDisplayText in the Code Editor Window

2. **Properties Window –** This displays all the properties for the module you currently have selected in the Project explorer window.

3.**Object List –** Works together with the Procedure list. Essentially, they are used for navigation BUT you can also use them to start off your subprocedures

4.**Procedure List –** Similar to above.

5.**Code Editor Window –** Displays code for our current module/procedure. Resizable.

# 9.3: Identify the Different Data Types

**Variables**

What is a variable? The computer creates an area in memory to store things. This area will have a memory address, most likely it could have a range of memory addresses. When writing your code, you can use any variable just by specifying its name. One of those being that you first MUST declare your variable.

You declare a variable using the Dim keyword. Dim stands for dimension which came from BASIC and was originally used to define the dimensions of an array. Now we use it for all variables.

Dim *variableName* As dataType

**Variable names:**

* Must begin with a letter but afterward may contain letters, underscores, and digits in any order.
* Can be up to 255 characters.
* Must be unique inside of the event or procedure, function or module. (Scope)
* Cannot contain spaces or any other punctuation except the underscore character.
* Cannot be a VBA keyword. These keywords are called **reserved words**.

For naming conventions used in this course, refer to the standards document **CDBM190 Standards.pdf (LO2)**

When you declare a variable, it is created with a default value:

* ***0*** for numeric data types such as Byte, Integer, Long, etc.
* **""** (empty string) for String variables.

**Data Types**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Description** | **Range** |
| **Byte** | 1-byte binary data | 0 to 255 |
| **Integer** | 2-byte integer | -32,768 to 32,767 |
| **Long** | 4-byte integer | -2,147,483,648 to 2,147,483,647 |
| **Single** | 4-byte floating-point number | -3.402823E38 to -1.401298E-45 (negative values) |
| 1.401298E-45 to 3.402823E38 (positive values) |
| **Double** | 8-byte floating-point number | -1.79769313486231E308 to -4.94065645841247E-324 (negative values) |
| 4.94065645841247E-324 to 1.79769313486231E308 (positive values) |
| **Currency** | 8-byte number with fixed decimal point | -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |
| **Boolean** | Logical variable evaluates to true or false | True or False |
| **String** | String of characters | Zero to approximately two billion characters |
| **Date** | 8-byte date/time value | January 1, 100 to December 31, 9999 |
| **Variant** | Any of the preceding | Any of the preceding |

A **Variant** is special data type that can contain any kind of data except fixed-length String data. (Variants support user-defined types). A Variant can also contain the special values **Empty**, **Error**, **Nothing**, and **Null.** You can determine how the data in a Variant is treated used the **VarType** function or **TypeName** function

While the use of not explicitly declare variants is not recommended, they can be of use when the needed data type can only be known at runtime, when the data type is expected to vary, or when optional parameters and parameter arrays are desired.

## Explicit Variable Declaration

VBA programming allows implicit typing of variables. Implicit variable typing occurs when a variable has not been declare before it is used in a program. When the VBA program runs, VBA makes the undeclared variable a **Variant** data type by default. Using implicitly typed variables will do nothing but increase the number of errors in our code.

**Implicit typing** – VBA changes any undeclared variables to type Variant. (In VB, you can also cast it)

**Explicitly declared** – programmer declares a datatype for a variable prior to use.

To enforce explicit declaration of variables, the line of code Option Explicit should be added to the top of the procedure, function or module. Or, Option Explicit can be set as a default property of the VBA editor.

## Constants and User-defined data types

**Constant**

A **constant** is a variable that is assigned a value that will not change when VBA code is executed. The constant does not require that the datatype be specified. You precede the variable name with the keywork **Const**

The following is an example of declaring a constant:

Const sngGST = 0.05

To create a global constant, preceded the declaration with the Public keyword. If you want to create a global constant that can be sued from outside the module it is contained in, you can create a “global” constant by adding the “g” prefix to the variable name.

Public Const gsngGST = 0.05

**System Defined Constants**

These are constants that we can use and that are defined by the VBA library. The constants are:

* True
* False
* Null

**User-Defined Data Types**

VBA allows you to create your own data type that consists of one or more of the Access data types. You declare a user-defined type between the Type ... End Type keywords.

Type Person

PersonName As String

PersonBirthDate As Date ' Date variable stores a birthdate.

PersonGender As Integer ' Integer for gender (0 for male, 1 for female)

End Type

These are really handy if you want to work with a record (for example).

## Arrays

**Array** – a variable that stores a *collection* of values, called **elements**, of a single data type in an ordered structure.

Arrays must be explicitly declared in VBA. When you create an array variable, you add the *a* prefix to the variable name.

Arrays in VBA are zero-indexed by default - the first element has an index value of 0.

To declare an array following:

Dim astrName(19) As String

*How many elements can be saved in the above?* ***20****. (starts at 0 – 19)*

To declare an array of 19 elements starting with an index of 1:

Dim astrName(1 TO 19) As String

*How many elements can be saved in the above?* ***19*** *(starts at 1-19)*

Multidimensional array:

Dim alngName(9,9,9) As Long

*How many elements can be saved in the above example? Creates a 3-dimensional array: 10 elements by 10 elements by 10 elements.*

A multidimensional array can have up to 60 dimensions.

Dynamic array:

Dim alngName() As Long

ReDim Preserve alngName(9,9,9) 'in a procedure will preserve prior values.

ReDim alngName(9,9,9) 'in a procedure will reinitialize and resize the array.

## Intrinsic constants and Enumeration

VBA includes pre-defined variables, called **intrinsic constants**.

Pre-defined variables that do not change. Primarily used as arguments for the various subroutines (methods) that are available in the Access library. Access has a number of methods that we can use that accept arguments. These arguments are usually some number that doesn’t really mean anything to a human. So VBA and VB have a number of constants setup that we can use because they have meaningful name that takes the place of some obscure number or string. The meaningful name of the constant represents a number or string expected as an argument by a procedure or function.

Enumeration – collection of related constants. Example: acObjectType

Intrinsic constants: <https://msdn.microsoft.com/en-us/library/t904x0th%28v=vs.90%29.aspx>

## Bang (!) vs Dot (.)

We’ve seen when you’ve been constructing queries that you can use both a dot or a bang.

**Late bound** – The method being called is looked up at runtime (and not at compile time).

**The bang operator** The bang operator is late bound. Traditionally, the bang was used to refer to items implicitly that belong to a collection. Collections can be things like forms, tables, reports all in an access program, controls, etc.

**The Dot operator** is early-bound. It shows that what follows is a property or method of the proceeding object. The dot is the only way you can access a named public member of an object.

Further details at:[*http://bytecomb.com/the-bang-exclamation-operator-in-vba/*](http://bytecomb.com/the-bang-exclamation-operator-in-vba/)

So when I was looking into best practice for !/. I quickly realized there are no best practices. Everyone has their own idea.

<https://msdn.microsoft.com/en-us/library/office/ff191896(v=office.15).aspx>

Now the controls on one form are formed in a collection. Let’s say we wanted to access one control names **NewData** – we could do this in all the ways shown on this page:

NOTE: This will only work as long as you have a control on your form with a Name property of “NewData”

### object syntax

Form Name

Example:

Collection to look in (controls)

Forms!frmMain.Controls!txtName

Collection to look in (in this case forms)

Control Name

In the above example: Forms is the collection. frmMain is the name of a form belonging to the Forms collection. Controls is the collection in a form that contains a number of control objects. txtName is the name of a textbox on the form frmMain.

NOTICE: if you are using collections, the bang follows the collection name BUT a dot must follow the name of a specific object (ie: a specific form name, a specific control name).

BUT Forms.frmMain.txtName will also work

If you want to access properties of a control for example, you MUST use the dot operator.

Forms!frmMain!txtName.Value

OR

Forms.frmMain.txtName.Value

For simplicities sake, I will be and I recommend you use the dot operator – this is because Intellisence doesn’t work with the bang operator.

The general rule for accessing an object is to use the least amount of descriptors you can as long as it is not ambiguous.

The Me self-reference is valid only for the instance of the form open in Form View and is used to get or set the value of a form property in the form’s own class module. So instead of using the form’s name and collection, you can use the keyword **Me**.

Me.ControlName.PropertyName

Not only can you modify a control’s properties at design time (compile time), you can also edit control properties at run time.

Properties of database objects can be treated as variables and assigned values in VBA code.

Example:

To assign “your name” to the value of **txtCustomerName** text box:

1. Specify the type of object or the Collection (Forms),
2. Specify the specific object (**frmCustomer)**
3. Specify the control name (**txtCustomerName)**
4. Specify the property to modify (**value)**

**Forms.frmCustomer.txtCustomerName.value = “Your name”**

**STOPPED HERE 03/28/2022**

If the form or the control object contains a space or other punctuation, you must enclose the name with square brackets [].

For example:

Forms.[frmDisplay Text].txtMessage

Remember: The last part of the named database object is the property:

Forms.frmDisplayText.txtMessage.**SetFocus**

# 9.4 Describe the scope and duration of variables

## Variable Scope

**Can anyone remember what scope refers to in Java?**

**Scope** of a variable in VBA – refers to the portion of code that can access the variable.

Variable scope is dependent on how the variable is declared. There are four scope levels in VBA:

1. **Local** (event or custom procedure level). Variable is only visible during the time when the procedure is running:

Dim varName As Datatype

1. **Form and Report level**. Variable is visible only during the time when the form/report is open:

Private varName As Datatype (preferred)

or

Dim varName As Datatype

1. **Module level.**  Variable is visible to all procedures and functions contained in that module

Private varName As Datatype (preferred )

or

Dim varName As Datatype

1. **Global or public.**  Variable is visible to all procedures and functions within all modules

Public varName As Datatype

**Duration of variables**

**Duration** – A variable characteristic which refers to how long a variable exists. Variables will have a longer or shorter lifetime depending on the declared scope. For example, a variable declared in a custom/event procedure is local in scope with a lifetime equal to the lifetime of the procedure.

**Dim** – exists only during the duration of the event/procedure

**Static** – exists over the lifetime of the application. (Multiple calls of the event/procedure)

Example:

Static varName As Datatype

# 9.7 create a form event procedure using database objects

## Form Event Procedures

Review: What is a procedure? Microsoft Visual Basic code is written in units called **procedures**. A procedure contains a series of Visual Basic statements that perform an operation, calculate a value, etc.

**Event procedure** – is a procedure that is associated with a certain object or Control. It could also be triggered by the system.

**Form event** – an action that is associated with a certain object or Control. Events are most often the consequence of a user action.

example:

* Mouse movement or click(s)
* Changes to data
* Open or closing forms, and other activities

By using an event procedure, you can add your own custom response to an event that occurs on a form, report, or control. When you create an event procedure for an object, Microsoft Access adds an event procedure template named for the event and the object to the form or report module. All you need to do is add code that responds in the way you want when the event occurs for the form or report.

## Create a form event procedure

Example: To demonstrate how to create form event procedures, we will create a new form containing a text box and two command buttons. When one of the buttons is pressed (clicked) it will place a piece of text in the text box. When the other button is pressed (clicked) it will clear the text box.

Step 1. Open the database and set **Option Explicit**

1. Open the database, **Movies.accdb**
2. Create > Form Design
3. Set the default option of the editor to Explicit Variable Declaration:
   1. Open up VBA editor, click on Visual Basic
   2. From the Tools Menu, Select Options….
   3. From the Editor tab, select the **Require Variable Declaration** check box
   4. Click OK to close the Options window. NOTE: You may have to close the database and reopen for the changes to take effect.
   5. At the top of the code editor window you should see a new line of code: “Option Explicit”
4. Switch back to the form. Click on the Access icon View Access button or press ALT-F11 from the keyboard.

Step 2: Create the form:

1. Size the form to 4” wide by 2” high
2. Create the following controls, 2 unbound text boxes with a label “Actor” and “Movie”, and 2 command buttons: “Get Movie” and “Clear Text”
3. Cancel out of the **Command Button Wizard** if it pops up when adding the command buttons. You may wish to turn off the Use Control Wizards option in your controls group
4. Name the objects: lblActor, txtActor, lblMove, txtMovie, cmdGetMovie, cmdClear respectively.
5. Under the form properties, turn off the Navigation Buttons, Record Selectors, and Scroll bars properties.
6. Save the form as **frmActorDisplay**

Step 3: Add the **OnClick** event code to the **Get Movie** button. We will customize the form button’s on click event. When the **Get Movie** button is clicked, a string constant will be displayed in the **txtActor** text box.

1. Select the **Get Movie** button
2. From the **Property Sheet**, select the **Event** tab, click on the **On Click** event property and click on the **Build** button. Code builder
3. From the **Choose Builder** window, select **Code Builder**, and then click OK. This will open the VBA editor and add an event procedure template name for the event: **cmdGetMovie\_Click()**
   1. The procedure name is created by combining the Name property of the control (button) with the name of the event.
   2. If you change the name of the button on the form it will NOT automatically update the event procedure name. You must do this manually (refactoring).
   3. Programming tip: Name your controls before you create the events.
   4. Before we add our code, we should probably do some commenting. Comments are going to be preceded by a single quote character (`) You’ll know that it’s a comment because the text will turn green.
   5. Enter the following code:

|  |
| --- |
| Private Sub cmdGetMovie\_Click()  ' Purpose: This procedure will place a text string into the text box txtActor  ' Parameters: None  ' Return: None    ' Use the "Me" keyword to reference controls/objects from within the forms' class module. BUT  ' if you are trying to refer to a form or report that is from a different module,  ' you must use a full reference (the form name)    ' Before you access the TEXT property on a control, you should set focus to the control.  ' You can access the Value property without setting focus.    Me.txtActor.SetFocus  Me.txtActor.Text = "Sean Connery"  Me.txtMovie.Value = "James Bond"  End Sub |

Step 4: Add the On Click event to the Clear Text button. When this button is clicked, the contents of txtActor text box will be removed

1. Open the VBA Editor
2. From the Object List select the **cmdClear** Command Button
3. The default event for that object will be placed in the code. In this case, it was the On Click event.
4. If the event added is not the event that you want:
   1. From the **Procedure List** select the event desired.
   2. The originally added event will remain in your code, just select it and delete it.
5. In this case, we DO want the On Click event, cmdClear\_Click().
6. Enter the following code:

|  |
| --- |
| Private Sub cmdClear\_Click()  ' Purpose: This procedure will clear the contents of our text boxes (txtMovie, txtActor)    Me.txtMovie.Value = Null  Me.txtActor.Text = ""  End Sub |

1. Save the code. Run – you get an error
2. We forgot to set focus: Add the following line  **Me.txtActor.SetFocus**

## Properties: Text Vs Value

**Text** – for setting / returning the text contained in a textbox/combobox. The textbox/combobox **must have focus** prior to use.

**Value** – set/returns a control’s default property. The control does not need to have focus prior to use.

You can look up these different controls in your Object Browser. Remember to filter on Access objects to reduce the list.

Complete Exercise 9.7. Solution online: Movies\_Mar31\_Exercise97.accdb

# 9.8 Create a user-defined procedure and functions

## Procedures

A series of VBA statements grouped together and used by Access to complete a task. Procedures can either be event or custom procedures. (Think of them as **methods**)

Event procedures are associated with forms and are triggered/called when an event is triggered.

Custom procedures **are not assigned to events** but must be called from another procedure or function.

**Syntax:**

* Can use Public, Private or Static prefix.
* Static preserves variables between calls

Private Sub ProcName (parameters)  
 Lines of code

...  
End Sub

**To call a procedure**:

ProcName (parameters)

OR

Call ProcName (parameters)

In this course we will use the keyword **Call** to invoke the procedure. Prefixing the procedure name with **Call** is good programming practice. It identifies the name that follows as the name of a procedure not just a variable and helps make the VBA code self-documenting.

**Create a new Procedure**

To demonstrate, we will create a public procedure that will change the color of text in a textbox

Step 1: Create a new module:

1. Open the VBA Editor, ALT-F11
2. Create a new module
   1. From the Insert menu, select Module
   2. In the Project Explorer window you will see a new heading called **Modules** and under it will be a new module called **Module1**. It will also have opened up a new code window for the new module.
3. Rename the module:
   1. Select Module1 in the Project Explorer
   2. In the Properties Window (View>Properties), change the **Name** property to **ChangeColor**

Step 2: Create a user-defined procedure

1. Click in the code window
2. From the Insert menu, select Procedure
   1. Enter ClearColorTextBox as the **Name** of the procedure
   2. Select **Sub** as the **Type**. Sub will create a procedure
   3. Select **Public** as the **Scope**. We want it to be available to all forms in our database.
   4. Click the OK button
3. Enter the code below in the ClearColorTextBox() procedure:

|  |
| --- |
| Option Compare Database  Option Explicit  Public Sub ClearColorTextBox(txtTextBoxName As TextBox)  ' This procedure clears and changes the background color of the text box  ' passed in.    txtTextBoxName.Value = Null  txtTextBoxName.BackColor = RGB(255, 255, 0)    End Sub |

**The procedure has a single parameter, the name of the textbox to clear**

1. Save the code

**Call a user-defined Procedure**

We will call the ClearColorTextBox procedure in both forms previously created in this LO.

Step 1: Call ClearColorTextBox procedure from the **frmActorDisplay** form:

1. Add a new button to the form called **Clear with Color**
2. Name the Button **cmdColorClear**
3. Set the event for cmdColorClear to onClick
4. In the VBA Editor. To switch to other code segments just double click on the object name in the Project Explorer window.
5. Double-click on **Form\_frmActorDisplay**
6. Locate the cmdClear\_Click() On Click event code and change it:

|  |
| --- |
| Private Sub cmdColorClear\_Click()  Call ClearColorTextBox(Me.txtActor)  End Sub |

## Functions

Question: What is the different between a function and a method in Java?

A **function** is a custom procedure that returns a single Boolean, numeric, character, or string value.

**Syntax:**

* Can use Public or Private prefix.

Private Function FuncName (parameters) As Datatype  
 Lines of code ...  
 **FuncName = returnValue**   
End Function

**To call a function**:

result = FuncName (parameters)

To demonstrate how to create a user-defined function, we will create a function that will take one string parameter and convert the string to toggle its case (lower case first letter, upper case remaining letters).

Step 1: Create a new module:

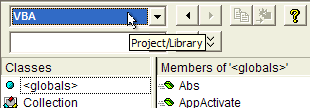
1. Open the form, frmActorDisplay, in Design View
2. Switch to the VBA Editor (ALT-F11)
3. Create a new module:
   1. From the Insert Menu, select Module
   2. In the Project Explorer window, you will see a new module called Module1 under Modules. It will also have opened up a new code window for this new module.
4. Rename the module:
   1. Select Module1, in Project Explorer
   2. In the Properties window, change the Name property to ChangeCase.

Step 2: Create a user-defined function:

The function we will create but with no error checking. We will assume the string being passed is valid.

1. Click in the ChangeCase Code window
2. From the Insert menu, select Procedure.
3. Enter ToggleCase as the Name
   1. Select Function as the Type
   2. Select Public as the Scope. We want the function to be available to all forms in our database.
   3. Click Ok button
4. The function we are creating requires one string parameter, *strToChange*, and returns a string. Change the function signature to the following:
   1. **Public Function ToggleCase(strToChange As String) As String**
   2. **End Function**
5. Save the form.

**Take a look at Access’ built-in functions**. Access has a large number of built-in functions. A number of those functions are for manipulating strings. To view the string function do the following:

1. Open the Object Browser: Object Browser icon
2. From the Project/Library dropdown list, select **VBA**
   1. 
3. In the Classes list, select **Strings**
4. Displayed on the right side in the **Members of Strings** is a list of all VBA string functions.
5. Click on the **LCase** function
6. Click on the **Help** button Help button
7. A window will appear displaying information about the **LCase** function. Help describes the VBA function and also gives an example of how the function is used. The **LCase** function takes a string parameter and returns the string converted to lowercase.
   1. Using the same process, lookup the other VBA functions we need to create the ToggleCase user-define function
      1. UCase
      2. Left
      3. Right
      4. Len
8. Close the help window

Step 3: Create the user-defined function code

1. We will need a temporary variable.
2. Work with the demo to complete the code. Solution:

|  |
| --- |
| Option Compare Database  Option Explicit  Public Function ToggleCase(strToChange As String) As String  'Insert code here  'Purpose: This function takes a string and changes the first letter of a string  ' to a lowercase letter and the rest of the string is converted to uppercase  Dim strTemp As String    'Set first letter to lowercase and store in the temp var  strTemp = LCase(Left(strToChange, 1))    'Take the remaining characters, covert to uppercase and add to temp var  strTemp = strTemp & UCase(Right(strToChange, Len(strToChange) - 1))    'Return the string - NOTE: put the name of the function equal to the value you want  ' to return    ToggleCase = strTemp  End Function |

Step 4: Let’s hook our new user-defined function into a form.

1. Create a form to call the ToggleCase() function:
   1. Size the form to 5” wide by 4” high
2. Create the following controls:
   1. Graphical user interface, application, table

      Description automatically generated
3. Name the objects: txtOriginal, txtResult, cmdToggleCase respectively.
4. Turn off the Navigation Buttons, Record Selectors, and Scroll Bars
5. Save the form as **frmChangeCase**

Step 5: Call the ToggleCase() function when the button is clicked. When the user clicks the button the text that is entered in the textbox txtOriginal will be converted using our ToggleCase call and displayed in the textbox txtResult

1. Switch to the VBA editor
2. From the **Object list** select **cmdToggleCase**
3. The default **On Click** event is the event we will add the VBA code to.
4. Enter the following code:
   1. Me.txtResult.Value = ToggleCase(Me.txtOriginal.Value)
5. Save the code
6. Switch to the form and test

## Using the Locals Windows to Debug Code

In the **Locals** window, Access automatically displays the name, current value, and type of all variables and objects in the current procedure. The values in the **Locals** window are updated each time you suspend code execution. If you step through the code, execution is suspended after each code execution.

# 9.9 write vba code using control structures

## Control Structures

VBA Code only runs in a sequence. It first executes a statement and, if there is no error, moves on to the next statement.

Control structures are used create code that can accomplish advanced tasks such as performing code when conditions are met or repeating code until a situation occurs.

There are two categories of control structures:

1. Decision structures/Conditionals (if/else, case) Case is like a switch in java
2. Reptition structures/Looping (do while, do until, for next)

# 9.10 write vba code using decision structures

A**decision structure (conditionals)** allows the program to make decisions.

* If…Then…Else
* ElseIf
* Select Case
* IIF (Write out the if statement similar to how you do it in Excel. Not really gonna cover this)

**Go ahead a read on your own time, I’ll skip to some demos.**

## If...Then...Else

If condition Then

Lines of code to run if the condition is true

...

End If

If condition Then

Lines of code to run if condition is true

...

Else

Lines of code to run if condition is false

...

End If

## ElseIf

If condition1 Then

Lines of code to run if condition1 is true

...

ElseIf condition2 Then

Lines of code to run if condition1 is false but condition2 is true

...

Else

Lines of code to run if condition1 and condition2 are both false

...

End If

Practice: Create a form with 3 text boxes and a button. It will compare the values in the first 2 text boxes and display a result regarding maximumness and equality regarding those numbers in the third text box:

|  |
| --- |
| Option Compare Database  Option Explicit  Private Sub cmdSubmit\_Click()  'If number one is larger, display it  'Otherwise, say that it isn't larger.  'Note: text in a textbox is Text, not a number, so we need to convert it.    Dim intNumOne, intNumTwo As Integer    intNumOne = CInt(Me.txtNumOne.Value)  intNumTwo = CInt(Me.txtNumTwo.Value)    If intNumOne > intNumTwo Then  Me.txtResult.Value = Me.txtNumOne.Value & " is larger than " & Me.txtNumTwo.Value  ElseIf intNumOne = intNumTwo Then  Me.txtResult.Value = Me.txtNumOne.Value & " is equal to " & Me.txtNumTwo.Value  Else  Me.txtResult.Value = Me.txtNumTwo.Value & " is larger than " & Me.txtNumOne.Value  End If    End Sub |

## Case Statements

**General Syntax**

Select Case varName

Case valueToMatch1

Lines of code to run if valueToMatch1 = varName

Case valueToMatch2

Lines of code to run if valueToMatch2 = varName

Case Else

Lines of code to run if there is no match for varName

End Select

varName can be a string, a numeric value, or an expression.

Case expressions can take one of the following forms where a variable is compared to:

* a number of single value conditions.

Case valueToMatch1

Lines of code to run if valueToMatch1 = varName

* One or more lists of values.

Case valueToMatch1, valueToMatch2, valueToMatch3, ...

Lines of code to run if varName = valueToMatch1, valueToMatch2, valueToMatch3, ...

* A range of values separated by the keyword To.

Case valueToMatch4 To valueToMatch5

Lines of code to run if varName is in the range of valueToMatch4 to valueToMatch5

* The keyword Is followed by a relational operator (<>, < , <=, =, >=, >) and a variable or literal value.

Case Is relationalExpression (example: Case Is < 100)

Lines of code to run if varName = relationalExpression

* The keyword Else. Code in the Else statement will be executed if no prior Case condition is satisfied. This expression should be listed last in the Case statement list.

Case Else

Lines of code to run if there is no match for varName in any of the Case statements

Demo: Create a form with 2 textboxes and a button. Name them txtPlace, txtResult, and cmdEval.

Make an event on click, and use the following code:

|  |
| --- |
| Option Compare Database  Option Explicit  Private Sub cmdEval\_Click()  'Summary: Output feedback to the user based on the place in the race  'Use a case statement to determine the feedback    Dim intPosition As Integer    intPosition = CInt(Me.txtPlace.Value)    Select Case intPosition  Case 1  Me.txtResult.Value = "You won gold!!"  Case 2, 3  Me.txtResult.Value = "You made the podium"  Case 4 To 7  Me.txtResult.Value = "You put in a good effort"  Case Is > 50  Me.txtResult.Value = "No comment"  Case Else  Me.txtResult.Value = "Put in more effort"  End Select    Me.txtPlace.Value = Null  End Sub |

# 9.11 write vba code using repetition structures

A **repetition structure (Looping)** goes through a block of code either a predefined number of times, or until something occurs to cause the program to break out of the loop.

**For…Next**

**Do While Loop**

**Do Until Loop**

## For Next

Repeats a group of statements a specified number of times.

Amount intCounter is changed each time through the loop (integer) Optional.

Exit/End Value of intCounter (integer) Required.

Initial Value of intCounter (integer) Required.

Loop Counter (integer) Required.

For intCounter = intStart To intEnd [Step intIncrement]

statements to execute

Allows loop termination when the surrounding condition code is met. Optional.

[conditional statement

Exit For

end of conditional statement]

Next intCounter

## A picture containing graphical user interface Description automatically generated

Code:

|  |
| --- |
| Option Compare Database  Option Explicit  'Procedure: cmdSubmit\_Click()  'Summary: get a number from the user. Display x multiples of 2 (where x =  ' value from the user).  Private Sub cmdSubmit\_Click()  Dim intXVariable, intMultiple, intTimesThroughLoop As Integer    'Clear the listbox before we do anything  Me.lstOutput.RowSource = ""    'Convert to an integer because the text from a textbox is Text (not a number)  intXVariable = CInt(Me.txtInput.Value)    'Start at 1, and go until you have printed x number of multiples of 2  For intTimesThroughLoop = 1 To intXVariable Step 1  'multiple is equal to the amount of times gone through the loop \* 2  intMultiple = intTimesThroughLoop \* 2    'Output the item to the listbox. Listboxes want strings so convert to a string first  Me.lstOutput.AddItem (CStr(intMultiple))    Next intTimesThroughLoop    End Sub |

## Do While

Repeats a block of statements while a condition is True. Exits when the condition is false.

### Syntax: Pre-test

Do While blnCondition   
 statements to execute

Loop

### Syntax: Post-Test

Do

statements to execute

Loop While blnCondition

#### Demo While PreTest

1. Make a form with 2 textboxes: txtBalance and txtOutput. Add a button cmdCalc

Solution:

|  |
| --- |
| Option Compare Database  Option Explicit  'Procedure: cmdCalc\_Click  'Summary: Obtain user's current bank balance. Calculate how many years it  ' will take the user to become a millionaire if they receive 6% interest  ' each year on their balance.  Private Sub cmdCalc\_Click()  Dim dblBalance As Double, intNumYears As Integer    'sng is short for single  Const sngInterest = 0.06    'obtain the user's balance  dblBalance = CDbl(Me.txtBalance.Value)    'Do While there is less than a million in the account  Do While dblBalance < 1000000  'add the interest for the year to the balance  dblBalance = dblBalance + (sngInterest \* dblBalance)    'increment the number of years  intNumYears = intNumYears + 1  Loop    'output the number of years to become a millionaire  Me.txtOutput.Value = CStr(intNumYears)    End Sub |

Form:

Graphical user interface, application

Description automatically generated

#### Demo While Post-Test

1. Create a form with 1 text box txtActor and 1 button cmdObtain
2. Make a click event for the button

|  |
| --- |
| Option Compare Database  Option Explicit  'Procedure: cmdObtain\_Click  'Summary: Ask the user for a number, display the associated actor  ' repeat until user enters a number that isn't 1,2, or 3  Private Sub cmdObtain\_Click()  Dim strActor As String  Dim intResponse As Integer    intResponse = 0  strActor = ""  Me.txtActor.Value = ""    Do  'Invoke InputBox, which opens an input box for the user  'We must cast whatever value to an integer  intResponse = CInt(InputBox("Enter a number from 1-3"))    Select Case intResponse  Case 1  strActor = "Harrison Ford"  Case 2  strActor = "Mark Hamill"  Case 3  strActor = "Carrie Fisher"  End Select  Me.txtActor.Value = strActor  Loop While intResponse >= 1 And intResponse <= 3    End Sub |

**Do Loop Statement Parts**

* **blnCondition** is any numeric expression or string expression that evaluates to True or False. If blnCondition is Null, then it is treated as False
* Any number of Exit Do statements may be placed anywhere in the Do…Loop as an alternate way to exit the loop. **Exit Do** is often used after evaluating some condition, for example, If..Then, in which case the Exit Do statement transfers control to the statement immediately following the Loop
* When used with nested Do.Loop statements, Exit Do transfers control to the loop that is nested one level above (immediate parent).
* Exit Do statements are considered to be poor programming style. You should always be able to exit a loop gracefully.

## Do Until

Repeats a block of statements while a condition is False (Until True). Exits when the condition is True.

### Syntax: Pre-Test.

Tests the condition prior to entering the loop.

* The loop structure executes the code statements only if **blnCondition** equals False and continues to do so until **blnCondition** becomes True
* Statements in the loop may never be executed.

Do Until blnCondition

statements to execute

Loop

Alter our frmPreTestLoop to use Do Until instead of Do While.

|  |
| --- |
| Option Compare Database  Option Explicit  'Procedure: cmdCalc\_Click  'Summary: Obtain user's current bank balance. Calculate how many years it  ' will take the user to become a millionaire if they receive 6% interest  ' each year on their balance.  Private Sub cmdCalc\_Click()  Dim dblBalance As Double, intNumYears As Integer    'sng is short for single  Const sngInterest = 0.06    'obtain the user's balance  dblBalance = CDbl(Me.txtBalance.Value)    'Do While there is less than a million in the account  'Do While dblBalance < 1000000  **Do Until dblBalance >= 1000000**  'add the interest for the year to the balance  dblBalance = dblBalance + (sngInterest \* dblBalance)    'increment the number of years  intNumYears = intNumYears + 1  Loop    'output the number of years to become a millionaire  Me.txtOutput.Value = CStr(intNumYears)    End Sub |

### Syntax: Post-Test

Tests the condition after entering the loop and executing the statements at least once. Similar to Pre-Test, it repeats iterations until blnCondition is evaluated to True

Do

statements to execute

Loop Until blnCondition

|  |
| --- |
| Option Compare Database  Option Explicit  'Procedure: cmdObtain\_Click  'Summary: Ask the user for a number, display the associated actor  ' repeat until user enters a number that isn't 1,2, or 3  Private Sub cmdObtain\_Click()  Dim strActor As String  Dim intResponse As Integer    intResponse = 0  strActor = ""  Me.txtActor.Value = ""    Do  'Invoke InputBox, which opens an input box for the user  'We must cast whatever value to an integer  intResponse = CInt(InputBox("Enter a number from 1-3"))    Select Case intResponse  Case 1  strActor = "Harrison Ford"  Case 2  strActor = "Mark Hamill"  Case 3  strActor = "Carrie Fisher"  End Select  Me.txtActor.Value = strActor  'Loop While intResponse >= 1 And intResponse <= 3  Loop Until intResponse < 1 Or intResponse > 3    End Sub |

**Do Loop Statement Parts**

* **blnCondition** is any numeric expression or string expression that evaluates to True or False. If blnCondition is Null, then it is treated as False
* Any number of **Exit Do** statements may be placed anywhere in the Do..Until loop as an alternate way. Similar to the statements for Exit Do above.
* When nested, control goes to the parent loop upon exit.
* Again, Exit Do is considered poor programming style.