VISUAL BASIC PROGRAMMING

Introduction:

Visual Basic is an object-oriented language (Object based programming) that was derived from the original BASIC language. It is the fastest and easiest way to create application for MS Windows, whether you are an experienced professional or new to Windows programming.

Visual Basic provides a complete set to simplify Rapid Application Development (RAD).

The **Visual** part refers to the method used to create the Graphical User Interface (GUI). Rather that writing numerous lines of codes to describe the appearance and location of interface elements, simply pre-built objects are added to the screen.

The **basic** refers to the *Beginners All-purpose Symbolic Instruction Code* language. Visual Basic has revolved from the original BASIC language and now contains several hundred statements, functions and keywords, which relate directly to the Windows GUI.

Note

Since Visual Basic 6.0 lacks inheritance it is not an object-oriented language; however, its user-defined types permit its use as an object-based language.

Visual Basic editions

Visual Basic comes in three editions namely:

i) Visual Basic Learning Edition
ii) Visual Basic Professional Edition
iii) Visual Basic Enterprise Edition

Visual Basic Learning Edition is an introductory edition in which Windows applications are created easier. It comes with tools needed to build mainstream Windows application.

Visual Basic Professional Edition is for computer professionals and includes professional features such as tools needed to develop Active-X and Internet controls.

Visual Basic Enterprise Edition is the most advanced edition and is aimed at programmers who build distributed applications in team environment. It includes all features of professional edition plus tools e.g. Visual Source Safe etc.

DEFINITIONS

Program: A set of instructions for the computer to obey and represent a method of processing the data.

Project: This is a collection of files created to make Windows applications i.e. In Visual Basic; each form is treated as a separate file.

Control: An item inside the standard Toolbox and can have a code attached to it.

Events: These are actions that Visual Basic can detect and respond to e.g. A user clicking a command button of a form will generate a click event for that button.

Methods: Are actions or behaviour that can be performed by a class of an object. I.e. something the object can do have done to it. Generally it consists of procedures encapsulated in an object. The syntax to use a method is:

Object. Method

Objects: Refers to a set of code (methods) and data (property) taken as a unit with a well-defined interface. Generally it refers to forms and controls that are included in an application. These objects are manipulated through their properties, events and methods.
Properties: These are attributes/characteristics of an object such as size, caption, color etc. The syntax to use with a property is as follows:

Object. Property=Value

TERMINOLOGIES

Design Time: Any time an application is being developed in a Visual Basic environment.
Run Time: Any time an application is running. Run time, enables the programmer/user to interact with the application.
Event Driven Programming:
When a program is event driven, its code executes in response to events invoked by the user, Operating System or the application.
This differs from procedural programs where the program starts at the first line of code and follows a defined pathway, calling procedures as needed.
***Note*:** List other differences between procedural programming and event driven programming

Object Oriented Programming:
Is approach to software design that differs significantly from structured programming. OOP looks at the real world as made up of objects, where each object belongs to a class of objects. I.e. OOP consists of a web of interacting objects, each housekeeping its own state
A class models a specific kind of object by encapsulating the data (property) that describe the object and the routines (methods) that manipulate the kind of object

Integrated Development Environment (IDE)

It refers to the environment in which V.B application can be developed, tested, and debugged. The Visual Basic IDE is made up of a number of components, namely: Toolbox, Project Explorer, Properties Window, and Form Layout etc.
The following are some of the components of Visual Basic IDE

i) Toolbox

It contains the icons of the controls that are placed on the form to create the application user interface. If control is missing on the Toolbox, it is selected from components on the project menu.

Steps for adding Active-X Controls

From Project menu, click Components and Components dialog box appears. On the Control Tab, select the controls desired and then click OK.

ii) Form Designer Window:

It contains the form that is designed and edited to make the application user interface. The form Designer displays two Windows for each form:

a) The form itself: - Elements of user interface.
b) Code Window: - Code behind the element of the form.

To toggle/switch between the two views; The View Code and View Form icons on the top of the ***Project Explorer*** are clicked for a given form.

iii) Properties Window:

It contains the property settings for selected controls. The Property Window lists all the properties a control has and their values.
The default values of a property can be changed by setting the Property window when the application is being designed or assigned a new value when the application is running.

iv) Project Explorer:

This is the window, which displays the components of the project such as forms, modules, designers etc. Each form in the application is represented by a file in the Project Explorer Window.
A form file contains both the description of the screen layout for the form and the program code associated with it.

v) Form Layout:

This is used to determine the initial position of the form in the application. This window is useful in applications that use multiple forms since it is possible to specify the position of one form in respect to the others.

vi) Immediate Window:

It refers to a debugging aid. While the application is running, the immediate window can be used to execute Visual Basic commands in immediate mode. Other windows are Local window and Watch window.

vii) Code Window:

This is the window used to write, display and edit Visual Basic code. It is a highly specialized word processor with a number of features that make writing Visual Basic code a lot easier. The code window has three parts namely:

a) ***Object List box:*** Displays the name of the selected object. The arrow to the right of the object list box displays a list of the controls/form, which are on the GUI.
b) ***Event Menu or Procedure Menu:*** It lists all available events for selected control/form. It usually changes depending on the type of control selected. The arrow to the right of the box displays all events for the control.
c) ***Main Part***: is used write the code for the event. The code appears between *Private Sub* and *End Sub* statements.

The default Visual Basic colours in the code window is as follows:

- Red for syntax error
- Green for comments
- Blue for Visual Basic keywords

- Black for other executable statements
- Yellow highlight of a statement for break in execution

CREATING AN APPLICATION

There are three steps for creating an application:

i) Designing the interface:
- Form object is drawn as the container
- Appropriate objects/controls are added to the form object from the toolbox

ii) Setting the Property values for forms and controls:

a) Design time:
- Select the object/control
- Using the properties window choose the object properties and settings

b) Run time:
- Access the code window
- Select the object/control and append code to the code event procedure

iii) Writing the program code: Instructions to be executed in response to the objects/controls events is written in the appropriate event procedure within the code window

MODULES

Visual Basic stores code in three kinds of modules:

i) Form Module
ii) Standard Module
iii) Class Module

Each of the *modules* may contain:

a) *Procedures*: A Sub, Function or Property Procedure
b) *Declarations*: Declarations may be constants, types, variables, Dynamic Link Libraries (DLL), Procedure Declarations etc.

i) FORM MODULE (.frm file name extension):

They are foundations of most Visual Basic applications. The form module may contain:
- Property setting for form and its controls.
- Form level declarations
- Event procedures and form level General procedure.

ii) STANDARD MODULE (.bas file name extension)**:**

They are containers for procedures and declarations commonly accessed by other modules within the application. There contain global declaration of variables, constants types, external procedures and global procedures.
Code written in standard module is not necessarily tied to a particular application.

iii) CLASS MODULE (.cls file name extension)

They are codes written to create new objects. The new objects can have their own customised Properties and Methods.

PROCEDURES

A procedure is a block (several Visual Basic statements) of code that is independent from other program code and performs some specific tasks.

There are three types of procedures used in Visual Basic:

- *Sub procedures*: do not return a value.
- *Function procedures*: return a value.
- *Property procedures*: can return and assign values, and set references to objects.

There are two major benefits of programming with procedures:

- A procedure allows breaking programs into discrete logical units, each of which you can debug more easily than an entire program without procedures.
- Procedures used in one program can act as building blocks for other programs, usually with little or no modification.

i) Sub Procedure

This is a block of Visual Basic code that appears between the Sub and End Sub keywords. They can either be *Private* or *Public* Sub procedures.

The syntax for a Sub procedure is:

[Private|Public][Static]Sub *procedurename* (*arguments*)
statements
End Sub

Each time the procedure is called, the *statements* between Sub and End Sub are executed. Sub procedures can be placed in standard modules, class modules, and form modules.

In Visual Basic, there are two types of Sub Procedures:

Event Procedure:

Is a procedure that is invoked (activated) by an event. The name of the procedure is derived from the name of the control (object) that is associated with it and the event for which the procedure is to be written.
When an object in Visual Basic recognizes that an event has occurred, it automatically invokes the event procedure using the name corresponding to the event. Because the name establishes an association between the object and the code, event procedures are said to be attached to forms and controls.
The name of the control and the event are joined by an underscore (_)

All event procedures use the same general syntax.

Syntax for a control event	**Syntax for a form event**
Private Sub controlname_eventname (*arguments*) statements End Sub	Private Sub Form_eventname (*arguments*) statements End Sub

General Procedure:

A general procedure tells the application how to perform a specific task. Once a general procedure is defined, it must be specifically invoked by the application. By contrast, an event procedure remains idle until called upon to respond to events caused by the user or triggered by the system.

Why create general procedures? One reason is that several different event procedures might need the same actions performed. A good programming strategy is to put common statements in a separate procedure (a general procedure) and have your event procedures call it. This eliminates the need to duplicate code and also makes the application easier to maintain.

ii) Function Procedure

Is a block of Visual Basic that appears between Function and End Function keywords. It can be *Private* of *Public*.

The syntax for a Function procedure is:

[Private|Public][Static]Function *procedurename* (*arguments*) [**As** *type*]
statements
End Function

Like a Sub procedure, a Function procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. Unlike *a Sub procedure, a Function procedure* can return a value to the calling procedure. There are three differences between Sub and Function procedures:

- Generally, you call a function by including the function procedure name and arguments on the right side of a larger statement or expression (*returnvalue = function()*).

- Function procedures have data types, just as variables do. This determines the type of the return value. (In the absence of an As clause, the type is the default Variant type.)

- You return a value by assigning it to the *procedurename* itself. When the Function procedure returns a value, this value can then become part of a larger expression.

Example

Write a program that inputs two numbers through textbox controls, Sum them and display the result using a label control

i) <u>Without using a function:</u>

```
Private Sub cmdAdd_Click ()
  A=CInt (txtNum1. Text)
  B=CInt (txtNum2. Text)
  Sum=A+B
  lblResult. Caption="Sum is: " &Sum
End Sub
```

ii) <u>Using a function:</u>

The event Procedure

```
Private Sub cmdAdd_Click ()
  A=CInt (txtNum1. Text)
  B=CInt (txtNum2. Text)
  Sum=Addition (A,B)
  lblResult. Caption="Sum is: " &Sum
End Sub
```

The Function Procedure

```
Private Function Addition (x As Integer, y As Integer) As Integer
        Addition=x+y
End Function
```

NOTE:

a) The code for a procedure should never appear inside another procedure
b) A General Procedure or Function Procedure is normally typed in the General section of a module. The General section is obtained by selecting General in the Object (Or Control) drop down list of the code window.
c) The empty rackets that appear after the procedure name are required and cannot be omitted.
d) After the called procedure has been executed, program execution returns to the Visual Basic statement following the one that invoked the procedure.
e) The numerous Visual Basic in-built functions are invoked in exactly the same way as user defined functions

Calling A Function

Functions are called by name and optionally a list of arguments follow this name in parenthesis.

Argument Passing Mechanism

a) Passing by reference:

This gives the procedure access to the actual variable. The calling procedure passes the address of the variable so that the procedure can change its value permanently. When passing the argument by reference, the argument type must match the declared type.

b) Passing by Value:
The procedure sees only a copy of the argument. This means that changes made to the value in the called procedure are lost when the procedure ends.
In Visual Basic, arguments are passed by value if the arguments being passed have ByVal keyword preceding them.

Example:
Private Sub (ByVal x As Integer)

Advantages Of Using Procedures

a) It makes product development more manageable
b) Software reusability i.e.. Using existing procedures as building blocks for new programs. It enables to avoids repetition of codes in a program. Packaging code as procedures allows it to be executed from several locations in the program by calling the procedure.

VARIABLES

A variable is a named storage location in computer's memory that can contain data which can be modified during program execution.

Variables have a name (the word used to refer to the value the variable contain) and a data type (which determine the kind of data the variable can store)

Declaring Variables
Variable declarations ensure that appropriate memory space is reserved for the variables depending on the data type of the variable.
Variables are declared with the Dim statement, supplying a name for the variable

Syntax:
Dim VariableName [As Type]
e.g.: *Dim Sum As Integer*

Variables declared with the Dim statement within a procedure exist only as long as the procedure is executing. When the procedure finishes, the variable is destroyed.
In addition, the value of the variable in a procedure is local to that procedure i.e. the variable in one procedure cannot be accessed in another procedure. This characteristic allows variables of the same name to be used in different procedures without conflicts or accidental changes.
The optional 'As Type' in the Dim statement allows data type of the variables to be declared. Data types define the type of information the variable stores.

Examples of data types include:
String, Integer, Long, Double, Single, Boolean, Currency

Other ways of declaring variables are:

a) Declaring a variable in the declaration section of the form module rather than within a procedure makes the variable available to all procedures in the module.

b) Declaring a variable using the Public Keywords makes it available throughout the application.
c) Declaring a local variable using the Static Keyword preserves its value even when a procedure ends.

Types of Declaration

i) Implicit Declaration
Implicitly declared variables do not need to be declared before usage. Visual Basic automatically creates a variable with that name, which can be used as if it had been explicitly declared. While this is convenient, it can lead to errors if the variables are misspelled.

ii) Explicit Declaration
This is when all variables are declared before use, this is done by placing the *Option Explicit* statement in the *General Declaration* of the module.
The Option Explicit statement instructs Visual Basic to check the module for any variables that are not explicitly declared. If an undeclared variable is used, an error message appears prompting for correction of the error.

To automatically place the *Option Explicit* statement at the top of each new module:

- On the Tools menu, click Options
- Click the Editor Tab
- Check Require Variable Declaration
- Click Ok.

NB: - The Option applies to new code modules only. Option Explicit statement must be entered manually on the General Declaration Section for existing code modules.
To manually enter the Option Explicit in the General Declaration Section:

- Activate the code window
- Click General from the Object list.
- Click Declarations from the Procedure/Event list
- Type 'Option Explicit'

Comments Statements
Adding a comment to a code makes it easier for somebody else to determine what the code does. Visual Basic ignores anything following a single quote ('). The comments can be placed on their own line or at the end of a line code.
Also preceding any line of code with Rem (the abbreviation for remark) instructs Visual Basic to ignore it.

Line Continuation Character
This is an underscore (_). It is used to break up a single statement into multiple lines. This makes the code statement easier to read because it is fully contained within the code editor window.
The line continuation character is placed after a space in the statement.

Note:
Steps To Make A Start-Up Object

i) From Project menu, click Project Properties
ii) From Project Properties Window, go to Start-Up Object List Box
iii) Choose the required Form then click OK.

VISUAL BASIC IN-BUILT FUNCTION

InputBox Function

This is an in-built function that displays a message in a dialog box, wait for the user to input text or click a button and returns a string containing the contents of the Text Box.

Syntax:
InputBox (Prompt,Title,Default,XPos,YPos,HelpFile,Context)

a) **Prompt:** This is a string expression displayed as a message in the dialog box. If the prompt contains more than one line, it can be separated using a Carriage Return Character Chr (13) or a Line Field Character Chr (10). - It is required.

e.g
InputBox ("My name is: " & Chr (10) & "John")

b) **Title:** This is a string expression displayed in the Title Bar of the Dialog Box. If omitted, the application name is placed in the Title Bar. - It is optional.

c) **Default:** This is a string expression displayed in the Text Box as a default response if no other input is provided. If omitted, the Text Box is displayed empty. It is optional

d) **X-Pos:** A numeric expression that specifies (in twips) the horizontal distance of the left edge of the dialog box from the left edge of the screen. If omitted, the dialog box is horizontally centred. It is optional

e) **Y-Pos:** A numeric expression that specifies (in twips) the vertical distance of the upper edge of the dialog box from the top of the screen. If omitted, the dialog box is approximately positioned a third of the way down the screen. It is optional

f) **HelpFile:** This is a string expression that specifies the help file to be used to provide context sensitive help for the dialog box. If provided, Context must also be provided. It is optional.

g) **Context:** This is a numeric expression assigned to the appropriate help topic. If provided, HelpFile must also be provided. It is optional.

MsgBox Function
This is a function used to display a dialog box, wait for the user to click a command button and returns an integer value indicating which button the user clicked.

Syntax: *MsgBox (Prompt, Button, Title, HelpFile, Context)*

Buttons: This is a number expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button and the modality of the message box. If omitted, the default value is 0. This argument is optional.

a) **Int:** Is a function that returns the integer portion of a given number. It usually converts a numeric value with a decimal to a whole number
Syntax Int (Number)

b) **Str:** Is a function that returns a variant (string) representation of a number
Syntax: Str (Number)

c) **Val:** Is a function that returns numbers contained in a string. The Val function will stop reading the string at the first character it cannot recognise as part of the number Val (String)

Type Conversion Functions
They are functions that convert an expression to a specific data type. The following are some of the type conversion functions

a) **CBool (Expression): -** Converts an expression to a Boolean. If the expression evaluates to a non-zero value, CBool returns True, otherwise False.
b) **CDbl (Expression): -** Evaluates an expression to Double.
c) **CInt (Expression): -** Converts a value to an Integer.
d) **CLng (Expression): -** Converts a value to a Long
e) **CSng (Expression): -** Converts a value to a Single
f) **CStr (Expression): -** Converts a numeric value to a string.
g) **CVar (Expression): -** Converts an expression to a variant i.e. to any given data type.

RGB Function:
The Visual Basic function RGB is a function that returns a color determined by the parameters that have values in the range of 0 to 255.

<u>Syntax:</u>

RGB (Red Component, Green Component, Blue Component)

The computer takes numbers between 0 to 255 as a scale determining the intensity of any of the True colours.

RND Function:
This is a Visual Basic Function that is used to generate random numbers. The random numbers lie between 0 and 1. e.g. Int (1+(Rnd*6)) will generate numbers between 1 and 6.

Example

Write a program that accepts a radius through the Inputbox function and generates a circle of different sizes. The generated circle should be painted with different colours generated by the RGB and Rnd functions

```
Private Sub cmdCircle_Click ()
    Dim Radius As Single, A As Single, B As Single
    Dim X As Integer, Y As Integer, Z As Integer
    frmCircle. ScaleMode = 6
    Radius = CSng (InputBox ("Enter the radius of the circle"))
    A = Me.ScaleWidth / 2
    B = Me.ScaleHeight / 2
    Me.FillStyle = 0
    X = Int ((Rnd * 255) + 1)
    Y = Int ((Rnd * 255) + 1)
    Z = Int ((Rnd * 255) + 1)
    Me.FillColor = RGB (X, Y, Z)
    Me. Circle (A, B), Radius, RGB(0, 0, 255)
End Sub
```

THE USER INTERFACE (Graphical User Interface)

The user interface is the link between the user and capabilities of the application. A well-designed user interface makes easy for the users to learn and use the application.

Elements of the User Interface:

A user interface consists of various elements with which the user can interact and control the application.

The first element is the FORM and it acts as a container object for all elements on the interface (controls).

Form Object

- The **Form** is where the user interface is drawn. It is central to the development of Visual Basic applications, whether for databases or other uses.
- Some of Form Properties:

Appearance	Selects 3-D or flat appearance.
BackColor	Sets the form background color.
BorderStyle	Sets the form border to be fixed or sizeable.
Caption	Sets the form window title.
Enabled	If True, allows the form to respond to mouse and keyboard events; if False, disables form and all controls.
Font	Sets font type, style, size.
ForeColor	Sets color of text or graphics.
Visible	If False, hides the form.

CONTROLS

They are the building blocks of any application. They are categorised into:

- **Intrinsic controls: -** Standard or inbuilt controls
- **ActiveX controls: -** Custom built controls

INTRINSIC CONTROLS

i) Label

It refers a GUI control used to display static text or text that shouldn't be edited by the user and it's often used to label other controls, such as TextBox and list boxes controls.
It can also be used to display texts such as status message and other program information.
Some of the frequently used properties of the Label control are as follows:
Alignment, Autosize, Caption, Name, WordWrap, ToolTipText and Font

- Caption property is used to change the text displayed in the Label control. At design time, this property is set using the control's properties window. At run time, the property is changed using the following syntax:
LabelName.Caption="Any given text"

- Alignment property sets the alignment of the text within a label control either to: -
  -Left justified (0- vbLeftJustify)
  -Right justified (1- vbRightJustify) or
  -Center justified (2- vbCenterJustify)

- AutoSize and WordWrap properties:
  - AutoSize property when set to true causes the control to horizontally expand and adjust to the size of its contents.
  - WordWrap property when set to true causes the control contents to wrap to the next line and expands vertically.

ii) Text Box: -

Is a control used to obtain information from the user or to display the information provided by the application. Unlike a label, the user can edit information displayed in the text box.
Text boxes can be used in conjunction with data control to display information from the database, set up database queries or edit records in a database.
Some properties of text box include: -
Alignment, BackColor, BorderStyle, Font, ForeColor, MultiLine, Name, PasswordChar, ToolTipText, Scrollbar etc

The textbox can either be single-line (for entering simple values) or multiline (for memos and longer notes).

Creating A Multiline Text Box:

The MultiLine property of the text box control, when used in conjunction with the scroll bar property allows multiple lines of text to be displayed. The MultiLine property can be set to True or False and the Scrollbar property can be set to:

-None (0- vbSBNone)
-Horizontal (1- vbHorizontal)
-Vertical (2- vbVertical)
-Both (3- vbBoth)

If the MultiLine property is set to True, the alignment property can be used to set the alignment of text within the text box

The text is LeftJustify by default.

If MultiLine property is False, setting the alignment property has no effect.

iii) Combobox and Listbox Controls

List Box Control

The ListBox control contains a number of items from which the user can select one or more of them (depending on the value of the control's MultiSelect property).

Combo Box Control

The ComboBox control is a combination of a TextBox and a ListBox control, with the difference that the list portion is visible only if the user clicks on the down arrow to the right of the edit area. The user can either choose an item from the list or enter a new one in the edit field. ComboBox controls don't support multiple selections.

NB: - Although the Combo Box allows a new selection to be entered, the new item is not automatically added to the list.
To add to the list, the *AddItem method* is used in the code.

Populating A List

A list can be populated either at design time or at run time.

a) At Design Time:

To add items at design time, use the List property:
i.e. Click List in the properties window then add items, pressing Ctrl+Enter for each item

b) At run time:

To add items to a list at run time, the AddItem method is used in the code.
Syntax:

Object. AddItem *Item Index*

The Item argument is a string that represents the text to add to the list.
The Index argument is an integer that indicates where in the list to add the new item. If not provided, the item is added at the proper sorted position, if Sorted property is set to true or at the end of the list if sorted property is set to False.

Example: -

A code segment to add the east Africa countries to a list control

```
Private Sum Form_Load ()
        lstCountries. AddItem "Kenya"
        lstCountries. AddItem "Tanzania"
        lstCountries. AddItem "Uganda"
End Sub
```

Using End With statement:

```
Private Sum Form_Load ()
  With lstCountries
        . AddItem "Kenya"
        . AddItem "Tanzania"
        . AddItem "Uganda"

    End With
End Sub
```

Setting Or Returning Values

To access items in a list box, the ListIndex and List Properties are used.
The ListIndex property sets or returns the Index number of the currently selected item. The first item has ListIndex 0 and if no item is selected, the ListIndex returns -1.The ListCount property returns the number of items in a list control and is always one value more that the Index number of the last item in the list. The NewIndex property returns the index of the last item entered/added to the list. This property is useful when working with sorted lists i.e. it tells where the item was inserted.

The List property sets or returns the text of an item in a List Box. An Index number is passed to the List property to specify which item to access.

Example

To return the text of a selected item, the ListIndex property is passed as shown below:

```
strSelection=lstMyList. List (lstMyList. ListIndex)
```

To return the string of a Combo Box, the Text property of the Combo Box is used

```
strSelection=cmbMyList. Text
```

Removing Items From A List

The RemoveItem method is used to remove an item from a list. The RemoveItem uses the following syntax:

Object. RemoveItem Index

Note: - The Clear method removes all the in the list and is used as shown below

Object. Clear

The following code segment verifies that an item is selected in the list and then removes the item:

```
Private Sub cmdRemove_Click ()
    If lstCountries. ListIndex > -1 Then
```

```
        lstCountries. RemoveIndex lstCountries. ListIndex
      End if
    End Sub
```
NB: An error occurs if an attempt to remove items is made when no items are selected.

Sorting a List

To cause the list to appear in alphanumeric order when displayed on a form, the Sorted property is set to True.

To display list items in order in which they are added to the list, the Sorted property is set to False.

iv) Option Buttons

They are also called Radio Buttons and they appear in groups. They are used to give a set of choices and when one is selected, the others are automatically deselected. OptionButton controls are useful for offering to the user a number of mutually exclusive selections

To group Option Buttons, a Frame control or Picture Box control is used as a container control. All Option buttons within a container control are functionally distinct.

All Option Buttons directly on the form function as a single group.

v) Check Boxes

They are used to represent On/Off, True/False Options to the user. Check Boxes are independent of each other i.e. Checking one does not affect any other on the form as in the case with Option Buttons.

By default, the Caption is on the right side of the form. This can be changed to RightJustify i.e. the box and not the text is on the left side.

The most common event is the Click event.

Check Boxes Value Properties:

Value of Property	Status of Checkbox
0	Unchecked
1	Checked
2	Grayed

vi) Command Buttons

A Command Button performs a task when the user clicks the button. It can be used to begin, interrupt or end a process. When clicked, the command button appears to be pushed in and is sometimes called a Push Button.

The most common event is the click event.

The Default property of a command button is set to True to allow the user to choose the control by pressing Enter and Cancel property set to True to allow the user to choose the button by pressing Esc.

Note:

It is a good idea to place buttons horizontally along the bottom of the form or vertically along the right side of the form.

vii) Horizontal and Vertical Scroll Bars

Scroll Bars provide easy navigation through a long list of items or large amount of information. It can be used as input devices or indicators of speed or quality e.g. to control the volume of computer game or to view the time elapsed in a timed process.

When using a scroll bar as an indicator of quality or speed, use **Max** and **Min** Properties to set the appropriate range for the control i.e. Set the Maximum and Minimum ranges of the scroll bar using the Max and Min properties. To specify the amount of change in scroll bar, use the **LargeChange** property for clicking in the scroll bar and the **SmallChange** property for clicking the arrows at the end of the scroll bar.

Scroll bar's **Value** Property increases or decreases by the values set to the LargeChange and SmallChange properties.

Example:

Write a program that uses a scroll bar to change the color of 2 labels from
i) Red to Yellow
ii) Yellow to White.

Solution

Scroll Bar properties:

1) Max	255
2) Min	0
3) SmallChange	1
4) LargeChange	1

```
Private Sub hsbColor_Change ()
   lblRed. BackColor=RGB (255,hsbColor. Value, 0)  'Changes from Red to
   Yellow                  lblYellow. BackColor=RGB (255,255,hsbColor. Value)
   'Changes from Yellow 'to White
End Sub
```

viii) Timer Control

The Timer control responds to the passage of time and is independent of the user. It can be programmed to take actions on regular intervals (mainly performs a task in the background). Its main property is the Interval property, which determines how often the timer notifies the application.

<u>Example:</u>

'A program to display the date and time

```
Private Sub Form_Load()
  lblDate1.Caption=Now()
End Sub

Private Sub Timer1.Timer()
  lblDate2.Caption=Now()
End Sub
```

ix) Data Control

The Data control is the key to *data binding*. It is used when connecting an application to an existing database to displays information from the database to the form. It has many properties and methods.

x) Frame Control

The Frame control is typically used as a container for other controls. You rarely write code that reacts to events raised by this control.

xi) Picture Box Control

It is used to display bitmaps, icons, or Windows metafiles, JPEG, or GIF files. It also displays text or acts as a container for other controls.

xii) Image Control

The Image control is similar to the PictureBox control, but it can't act as a container for other controls and has other limitations as well. Nevertheless, you should use an Image control in place of a PictureBox control whenever possible because Image controls consume fewer system resources.

xiii) Line

Adds a straight-line segment to a form.

xiv) Shape

Adds a rectangle, square, ellipse or circle to a form, frame, or picture box

xv) OLE container

The OLE control can host windows belonging to external programs, such as a spreadsheet window generated by Microsoft Excel. In other words, you can make a window provided by another program appear as if it belongs to your Visual Basic application.

MANAGING FORMS

In most applications, the user interface is composed of more than one form. Visual Basic provides a variety of methods for showing, hiding, loading and unloading forms. The following is a list of methods and statements for working with forms.

Statement/Method	*Task To Perform*
a) Load statement	*-Used to load a Form into memory but does not display it*
b) Show method	*-Used to load and display a Form*
c) Show method	*-Display a loaded Form*
d) Hide method	*-Hides a Form from view*
e) UnLoad statement	*-Hides a Form from view and unloads it from memory*

Displaying Forms

Both the Show method and Load statement load forms into memory. However, the Show method displays the form while the Load statement does not.

Syntax for Show method:

Object. Show

e.g. frmForm.Show

Syntax for Load statement:

Load Object

e.g. Load frmForm

Using The Load Event

This event occurs when the Form is loaded into memory either through the use of Load statement or Show method.

The Load event procedure can be used to initialise the Form i.e. The Form_Load event can run code that specifies default settings for controls, initialise the module level variables etc.

Modal And Modeless Forms

The Show method allows the Form to be displayed either as Modal or Modeless.

A Modal Form does not allow the user to interact with other forms in the application at the same time. When a Modal Form is displayed, the application beeps if the user clicks out of the form.

A Modeless Form allows the user to switch to another Form.

The style argument for the Show method determines if the Form is Modal or Modeless.

Example:

```
frmForm. Show vbModeless        'Form is Modeless
frmForm. Show vbModal           'Form is Modal
```

Hiding and Unloading Forms

When you need to remove a Form from the screen, you can either hide or unload it. Hide method and UnLoad statement will both remove the Form from the display. The UnLoad statement also removes the Form from the memory while the Hide method does not.

Syntax for Hide method:

Object. Hide

e.g. frmForm. Hide

Syntax for UnLoad statement

UnLoad Object

e.g. UnLoad frmForm

Using Unload Event

The Unload event procedure is used to verify that the form should be unloaded or to verify actions to take place when the form is unloaded. Any form level validation code needed for closing the form or saving data to a file can be executed after the Unload event.

Unload event occurs when:

a) The form is unloaded using the Unload statement
b) The user closes the form either by clicking the close button on the application title bar or clicking the close command on the application menu.

NB: - Unload event does not occur if the form is removed from the memory by the End statement.

Ending An Application
The execution can be ended by unloading the last form in the application or by using the End Statement.
The End statement terminates the execution of an application and unloads ALL the forms from memory.

TYPES OF APPLICATIONS

In visual basic there are two types of applications: -

i) **Single Document Interface (SDI)**
These are applications that have forms, which are independent of each other.

ii) **Multiple Document Interface (MDI)**
This is the Microsoft term for Windowing environment in which one window, the MDI Container or MDI Parent Form contains many other Windows.
MDI applications have many Child Forms and only one Parent Form.

MDI APPLICATIONS
Every application may have one MDI Form object and many MDI Child Forms. If an MDI has menus, the child forms automatically replaces the MDI Form object Menu Bar when the MDI Child Form is active.
A minimized MDI Child Form is displayed as an icon within the MDI Parent Form.
An MDI Parent Form contains menus and Picture box control and other controls which have alignment property. To place other controls in the MDI Parent Form, you need to draw a Picture Box onto the form and then draw other controls inside the Picture Box.
An MDI Parent Form cannot be Modal i.e. Program controls can move to other forms or dialog boxes.
All Child forms are contained within the MDI Parent Form.

CONTROL STRUCTURES
Control structures used to allow the control of flow of program's execution. They include:

i) Sequence
ii) Decision
iii) Looping

I) Decision Structures (Selection Structures)

Visual Basic procedures can test conditions and then, depending on the results of that test, perform different operations. The decision structures that Visual Basic supports include:

i) If...Then
ii) If...Then...Else
iii) Select Case

i) If … Then (Single Selection) Structure:
This structure selects or ignores an action depending on the condition
Syntax:

```
If condition Then
    statements
End If
```

The condition is usually a comparison or any expression that evaluates to a numeric value. A zero numeric value is False while a non-zero numeric value is True e.g.:

```
If Average>=60 Then
    lblGrade. Caption="Pass"
End If
```

ii) If … Then … Else(Double Selection) Structure

This structure executes one block of statement if the condition is True and another block if the condition is False.

Syntax:

```
If condition Then
    Statements
Else
    statements
End If
```

Example:

```
If Average>=60 Then
    lblGrade.Caption="Passed"
Else
    lblGrade.Caption="Failed"
End If
```

If … Then … ElseIf (Multiple Selection) Structure:-

If...Then…ElseIf is really just a special case of If...Then...Else. Evaluation of the If … Then … ElseIf is the construct that uses several conditions with the *ElseIf* keyword. Notice that you can have any number of ElseIf clauses

Syntax

```
If condition1 Then
[statementblock-1]
ElseIf condition2 Then
[statementblock-2]] ...
Else
[statementblock-n]]

End If
```

iii) Select Case

Visual Basic provides the Select Case structure as an alternative to If...Then...Else for selectively executing one block of statements from among multiple blocks of statements. A Select Case statement provides capability similar to the If...Then...ElseIf statement, but it makes code more readable when there are several choices

A Select Case structure works with a single test expression that is evaluated once, at the top of the structure. Visual Basic then compares the result of this expression with the values for each Case in the structure. If there is a match, it executes the block of statements associated with that Case:

Syntax:

```
Select Case testexpression
    Case Value 1
      Statements block 1
    Case Value 2
      Statements block 2
    Case Value n
      Statements block n
    .
    .
    .
    Case Else
        Statements block n+1
End Select
```

The block of the *Case Else* statement is optional

Some case statements can be followed by multiple values separated by commas

If … Then … Else … structures can only be replaced by the Select Case structure only if the If statement and each ElseIf statement evaluates the same expression i.e. the If … Then … Else statement

NB: If the Case Else is not placed as the last statement, a syntax error occurs.

II) Iteration Structures (Looping)

These structures allow for execution of one or more lines of code repetitively. The loop structures that Visual Basic support include:

i) <u>Do … Loop</u>

Do loop is used to execute a block of statements an indefinite number of times. There are two variations of **Do … Loop** statement but both use the same structure. The loop is executed either While the *condition is true or Until the condition becomes True*. The two variations uses the Key Words ***While*** and ***Until*** to specify how long the statements are executed.

To execute a block of statements while the condition is true, the following syntax is used:

```
Do While Condition
        statements
Loop
```

To execute a block of statements until the condition becomes true, the following syntax is used:

```
Do Until Condition
    statements
Loop
```

In the **Do … While … Loop**, the statements can execute any number of times as long as the condition is true. The number of iterations need not be known before the loop starts. If the condition is initially **False**, the statement may never execute unlike in the **Do … Until … Loop**.
These two are Pre- Condition Loop. For Post- Condition Loop, the *While* or *Until* key words with the condition are placed after the Loop key word.

ii) For … Next

Is a looping structure that is used when the number of repetition are known in advance. This loop uses a variable (counter) that increases or decreases in value during each repetition of the loop.
Syntax:

```
For Counter = Start To End [Step increment]
    Statements
Next Counter
```

The arguments *Counter, Start, End and Increment* are all numeric. The keyword in square bracket with the arguments is optional.
The increment can either be positive or negative. If positive, then Start must be less or equal to End argument and if negative, then Start must be greater or equal to End argument for the loop to execute.

In executing a **For … Loop**, Visual Basic does the following:

i) It sets Counter = Start
ii) It tests to see if Counter is greater than End. If so, Visual Basic exits the loop
iii) It executes the statements
iv) It increments Counter by 1 or value specified in Increment argument
v) It repeats steps 2 to 4

In most cases, the **For … Loop** can be represented with other repetition structures e.g. the Do…While loop equivalent is:

```
Counter = Start
Do While Counter <= End
    statements
    Counter = Counter + Increment
Loop
```

Database programming

It refers to accessing a database via a program as opposed to through the DBMS user interface.

Types of Database Access

I) DAO (Data Access Objects)
- Data Control
- Programming DAO

II) RDO (Remote Data Objects)
- Remote Data Control
- Programming RDO

III) ADO (ActiveX Data Objects)
- ActiveX Data Objects Data Control (ADODC)
- Programming ADO

Data Access Objects

These are database access model that allows programmers to access the jet database engine, as supplied with Microsoft access.

Data Control:
It refers to a control that provides a link between an application program (Visual Basic) and the database file. By itself, the Data Control won't display any data; you must have bound controls to do that. Basically the Data Control is connected to a set of records on the target Data Base. This set of records constitutes the RecordSet Object of the Data Control. Depending on the RecordSetType of the Data Control, the table records can be added, modified or deleted on the target Data Base from Visual Basic form.

Properties Of Data control: -

i) **DataBaseName:** Returns or sets the name of the source database for the data control. Must be a fully qualified path and file name.
ii) **RecordSetType:** This setting specifies the type of RecordSet that a Data Control is connected to. The setting of this property may be one of the following: table, dynaset or snapshot
iii) **RecordSource:** This property holds the name of the table to which the Data Control is connected.
iv) **Connect:** Type of database. Default is Microsoft Access (or Jet).

Recordset Object:
RecordSet object represents the entire set of records from a table (a virtual table via a query). Tables in a database can be accessed directly and the only way to view or manipulate the records is through the RecordSet object. At any time, the RecordSet object refers to a single record within the set as the current record.
Recordsets come in different types established via the **RecordsetType** property. These are:

i) **Table:** A single table in a database. This type of RecordSet is updateable.
ii) **Dynaset:** A dynamic set of records that may be used to add, delete, or alter fields from one or more tables in a database. This type of RecordSet is updateable.

iii) **Snapshot:** This is a static set of records from one or more tables that may be used for searching or report generation. This type of RecordSet is not updateable.

Properties And Methods of RecordSet Objects:

Methods:

i) **AddNew:** Adds a new record to the Recordset. All fields are set to null and this record becomes the current record.
ii) **CancelUpdate:** Cancels any changes made to the current record prior to calling the update method (i.e. either with Edit or AddNew method)
iii) **Update:** Saves any changes made to the current record of the RecordSet object
iv) **Delete:** Deletes the current record or a group of records
v) **MoveFirst:** Moves to the first record in the RecordSet object and makes it the current record
vi) **MoveNext:** Moves to the next record in the RecordSet object and makes it the current record
vii) **MovePrevious:** Moves to the previous record in the RecordSet object and makes it the current record
viii) **MoveLast:** Moves to the last record in the RecordSet object and makes it the current record
ix) **Requery**: Updates the data in a Recordset object by re-executing the query on which the object is based.

Note

These methods are invoked using the double-dot notation. For example, if you have a data control named **datExample**, to invoke a method named **MethodName,** the notation is:

datExample.Recordset.MethodName

Properties:

i) **EOF: (End of File)** returns True or False value that indicates whether the current record position is after the last record in the RecordSet object.
ii) **BOF: (Beginning of File)** returns True or False value that indicates whether the current record position is before the first record in the RecordSet object.
iii) **BOFAction:** sets or returns a value indicating what action the Data Control should take if the BOF property is true.
iv) **BookMark:** a variant that identifies a row in the RecordSet. Each row has its own unique bookmark that is not related to the record's order in the RecordSet. This property is saved to a variant so that one can return to this record later by assigning the variable to the BookMark property.
v) **ReadOnly:** returns or sets a value that determines whether the control RecordSet is opened for read only access.
vi) **RecordCount:** The total number of records in the Recordset

Note

To refer to a Recordset property, use a 'double-dot' notation. For example, if you have a data control named **datExample**, to refer to a property named **PropertyName**, the notation is:

datExample.Recordset.PropertyName

Bound Controls

These are data aware control that can provide access to a specific field or fields in a database through the Data control. The data aware control is typically bound to a Data control through the DataSource and DataField properties

Properties:

i) **DataSource:** Name of a Data control though which the controls (bound controls) are bound to the Data control
ii) **DataField:** Name of a field in the RecordSet that the control displays and updates.

Note: Text box control is the most widely used data bound tool. **Text** property is data bound. Others are labels, checkbox and picture box controls

Steps In Entering New Records

- Click Add command button
- Enter details in the appropriate fields
- Click the Save command button to save the changes

Code

```
'To Add new record
Private Sub cmdAdd_Click ()
        datCourses.RecordSet.AddNew
        txtCourseCode. SetFocus
End Sub

'To save the record
Private Sub cmdSave_Click ()
        datCourses.RecordSet.Update
        MsgBox "Record Updated"
End Sub

'To move to the next record
Private Sub cmdNext_Click ()
        datCourses.RecordSet.MoveNext
        If datCourses.RecordSet.EOF Then
          datCourses.RecordSet.MoveLast
         MsgBox "This is the last record"
      End If
End Sub

'To move to the previous record
```

```
Private Sub cmdPrevious_Click ()
            datCourses.RecordSet.MovePrevious
            If datCourses.RecordSet.BOF Then
               datCourses.RecordSet.MoveFirst
              MsgBox "This is the First record"
        End If
End Sub

'To move to the last record
        Private Sub cmdLast_Click ()
            datCourses.RecordSet.MoveLast
        End Sub

            'To move to the first record
         Private Sub cmdFirst_Click ()
            datCourses.RecordSet.MoveFirst
         End Sub

            'To cancel an update before saving
           Private Sub cmdCancel_Click ()
               datCourses.RecordSet.CancelUpdate
         End Sub

            'To delete a saved record
          Private Sub cmdDelete_Click ()
            If datCourses.RecordSet.RecordCount=0 Then
                 MsgBox "The table is empty"
                 Exit Sub  'It prematurely ends the procedure at this point
            Else
                datCourses.RecordSet.Delete
            End If
            If Not datCourses.RecordSet.EOF Then
             datCourses.RecordSet.MoveNext

         ElseIf Not datCourses.RecordSet.BOF Then
            datCourses.RecordSet.MovePrevious
        Else
            MsgBox "This was the last record"
      End If
            datCourses.RecordSet.Requery
End Sub
```

Find Method:

The Data control provides four methods of finding records in the RecordSet.

i) **FindFirst:** Is a method that finds the first record that meets the specified criteria.

ii) **FindLast:** Is a method that finds the last record that meets the specified criteria.

iii) **FindNext:** Is a method that finds the next record that meets the specified criteria.

iv) FindPrevious: Is a method that finds the previous record that meets the specified criteria.

These methods can find records in the RecordSet based on the user specified criteria
Syntax:
RecordSet. FindFirst Criteria
The criteria argument is a string expression specifying the relation between Field values and Constant.
Example:
Data1.RecordSet.FindFirst "Town='Kitui' "
This locates the first record in the RecordSet in which the town is Kitui.

The **literal** within the criteria string should be within single quotes (')

When FindFirst method is called, Visual Basic locates a record that matches the criteria and repositions the Data control to this record. However, the NoMatch property is examined and it's value set to False if the record is found and True otherwise.
NB: You can only use a FindFirst method with DynaSet RecordSet types.

The following code shows how FindFirst method is used:

```
Private Sub cmdFind_Click ()
   Dim strTitle As String, strSearch As String, varBookMark As Variant
   With datCourses. Recordset  'Reduces repetition of the same statement
      If. RecordCount = 0 Then
         MsgBox "File is empty"
         Exit Sub
      Else
         . MoveFirst
         strTitle = InputBox ("Enter the course title to search")
         If strTitle = "" Then
            MsgBox "Course Title has not been entered"
            Exit Sub
         End If
         varBookMark =. Bookmark
         strSearch = "[Course Title]='" & strTitle & "' "
         . FindFirst strSearch
         If. NoMatch = True Then
            'Unable to find the record
            . Bookmark = varBookMark

            MsgBox "Unable to find Course Title " & strTitle
         End If
      End If
   End With
End Sub
```

<u>Seek Method:</u>
Locates a record in an indexed Table type RecordSet object that satisfies the specified criteria for the current index.

Syntax:

RecordSet.Seek Comparison, Key1, Key2, Key3, …,Key13

Where:

Comparison is one of the string expressions <, <=, >, >= or =

Key1, Key2, … Key13 is one or more values corresponding to the fields in the RecordSet objects current index as specified by its Index property setting

When using the Seek method, the current index must be set with the Index property. If the index satisfies a non-unique key field, Seek locates the first record that satisfies the criteria. Seek method will search through the specified criteria by Comparison and Key. Once found, it makes that record the current record and sets the NoMatch property to False.

If Comparison is equal to (=), greater than (>) or greater than or equal to (>=), Seek

starts at the beginning of the index and searches forward.

If Comparison is less than (<) or less than or equal to (<=), Seek starts at the end of the index and searches backward

The Key1 argument must be of the same field, with DataType as the corresponding field in the current index.

NB: You can only use a Seek method with a Table type RecordSet.

Code using the Seek method

```
Private Sub cmdSeek_Click ()
   Dim strSeek As String, strMessage As String, varBookMark As Variant
   With datCourses. RecordSet  ‘Reduces repetition of the same statement
         If. RecordCount=0 Then
               MsgBox “File is empty”
               Exit Sub
         Else
               . Index = ”idxCourseCode”
               . MoveFirst
               strMessage= “Enter the course code to search”
               strSeek= InputBox(strMessage, “Course Code”)
               If strSeek =”” Then
                     MsgBox “Course Code has not been entered”
                     Exit Sub
               End If
               varBookMark=. BookMark
               . Seek “=”, val (strSeek)

                     ‘Return to the current record in case seek fails
                  If. NoMatch=True Then
                           ‘Unable to find the record
                      . BookMark = varBookMark
                     MsgBox “Unable to find Course Code ” &strSeek
                  End If
          End If
       End With
    End Sub
```

MENU

Menus are used to make applications more users friendly. They are one of the most popular ways of organizing a large number of options. In Visual Basic, menus can go up to 6 levels. They are only attached to forms and are designed using the **Menu Editor.**

Window List:
It lists all the loaded forms (child forms) and checks the currently active form.
In the Menu Editor, select the Menu Item to create a window list and check WindowList.

Arranging Windows:

```
'To cascade windows:
Private Sub mnuWindowsCascade_Click ()
        Me. Arrange vbCascade
End Sub

'To tile windows horizontally:
Private Sub mnuWindowsTileHorizontally_Click ()
        Me. Arrange vbTileHorizontal
End Sub

'To tile windows vertically:
Private Sub mnuWindowsTileVertically_Click ()
        Me. Arrange vbTileVertical
End Sub

'To make a Pop-Up menu:
Private Sub MDIForm_MouseUp (Button As Integer, Shift As Integer, _
X As Single, Y As Single)
If Button = vbRightButton Then
        Call PopUpMenu (mnuAcademic)
End If
End Sub
```

SPLASH SCREEN

This is an attractive form that usually appears when the program is loading and tells the user more about the application. In Visual Basic, this form is pre-designed and given the name frmSplash

Steps To Add A Splash Screen:
i) Select **Project** from the menu
ii) Click **AddForm**
iii) Select **Splash Screen**
iv) Click the **Open** button
v) Change the labels and any graphics in the Splash Screen.

To make the Splash Screen appear as the first item:
i) Make the Splash Screen as the Start-Up object

ii) Insert a Timer control onto the Splash Screen and set its Interval property to 1500.
This implies that the form appears for 1.5 seconds and then disappears
iii) Write the following code into the timer event:

```
Private Sub Timer1_Timer ()
        Load MDIForm
        UnLoad frmSplash
        MDIForm.Show
End Sub
```

LOG ON FORM
Steps To Add a Log On Form:
i) Select **Project** from the Menu Bar
ii) Click on **Add Form**
iii) Select **Log-In** and click **Open**
Change the options appropriately.