KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
DEPARTMENT OF CS, CA & IT
SYLLABUS

Semester – III

18CSU304B PROGRAMMING IN VISUAL BASIC /GAMBAS 3H – 3C

Instruction Hours / week: L: 3 T: 0 P: 0 **Marks:** Internal: **40** External: **60** Total: **100**
End Semester Exam: 3 Hours

Course Objectives

- To introduce computer programming using the VISUAL BASIC programming language.
- To Emphasis on event-driven programming methods, including creating and manipulating objects, classes, and using object-oriented tools such as the class debugger.

Course Outcomes (COs)

1. To identify the differences between the procedural languages and event driven languages.
2. Construct appropriate user interfaces for simple programs, and design systems with minimal complexity and maximal functionality.
3. Analyze and construct efficient and effective algorithms and translate to appropriate control structures in visual basic.

Unit I - GUI ENVIRONMENT

Introduction to graphical user interface (GUI), programming language (procedural, object oriented, event driven), the GUI environment, compiling, debugging, and running the programs.

Controls : Introduction to controls textboxes, frames, check boxes, option buttons, images, setting borders and styles, the shape control, the line control, working with multiple controls and their properties, designing the user interface, keyboard access, tab controls, default & cancel property, coding for controls.

Unit II – OPERATIONS

Data types, constants, named & intrinsic, declaring variables, scope of variables, val function, arithmetic operations, formatting data.

Decision Making : If statement, comparing strings, compound conditions (and, or, not), nested if statements, case structure, using if statements with option buttons & check boxes, displaying message in message box, testing whether input is valid or not.

Unit III -MODULAR PROGRAMMING

Menus, sub-procedures and sub-functions defining/creating and modifying a menu, using common dialog box, creating a new subprocedure, passing variables to procedures, passing argument by value or by reference, writing a function/ procedure.

Unit IV- FORMS HANDLING

Multiple forms creating, adding, removing forms in project, hide, show method, load, unload statement, me keyword, referring to objects on a different forms

Iteration Handling: Do/loops, for/next loops, using msgbox function, using string Function

Unit V- ARRAYS AND GROUPED DATA CONTROL

Arrays - 1-dimension arrays, initializing and array using for each, user-defined data types, accessing information with user-defined data types, using list boxes with array, two dimensional arrays. lists, loops and printing list boxes & combo boxes, filling the list using property window / add item method, clear method, list box properties, removing an item from a list, list box/ combo box operations.

Database Connectivity: Database connectivity of forms with back end tool like mysql populating the data in text boxes, list boxes etc. searching of data in database using forms. Updating/ editing of data based on a criterion.

SUGGESTED READINGS

1. Programming in Visual Basic 6.0 by Julia Case Bradley, Anita C. Millispangh. 2014. Tata Mcgraw Hill Edition.

WEB SITES

1. https://www.tutorialspoint.com/vb.net
2. howtostartprogramming.com/vb-net/
3. https://www.vbtutor.net/lesson1.html
4. gambas.sourceforge.net/

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
DEPARTMENT OF CS, CA & IT
LESSON PLAN

SUBJECT NAME: PROGRAMMING WITH VISUAL BASIC/GAMBAS

SUBJECT CODE: 18CSU304B **SEMESTER: III**

STAFF: K. BANUROOPA **CLASS: II B.Sc. CS A& B**

UNIT I			
SL. No.	**Lecture Duration (Hours)**	**Topics to be Covered**	**Support Materials**
1.	1	Introduction Graphics User Interface (GUI), Programming Language (Procedural, Object oriented, event driven) -	S1:1-4
2.	1	the GUI environment, compiling, debugging, and running the programs	S1:6-30
3.	1	Textboxes, Frames, check boxes, options buttons, Setting a border and style, the shape control, the line control	S1:52-57
4.	1	Working with multiple controls and their properties	S1:57-59
5.	1	Designing the user interface, keyboard access, tab controls. Default & controls property	S1:61-65
6.	1	Coding for Controls	S1:66-70
7.	1	Recapitulation and Discussion of Possible Questions	
Total No. of Hours Planned for Unit I : 7			

UNIT II			
SL. No.	**Lecture Duration (Hours)**	**Topics to be Covered**	**Support Materials**
1.	1	Data types, constants, named & intrinsic, declaring variables, scope of variables,	S1:86-94
2.	1	val function, arithmetic operations, formatting data.	S1:97-100
3.	1	If statement, comparing strings, compound conditions (and, or, not),	S1:123-132
4.	1	nested if statements, case structure	S1:133:136
5.	1	using if statements with option buttons & check boxes	S1:136-138
6.	1	Displaying message in message box, testing whether input is valid or not.	S1:139-143
7.	1	Recapitulation and Discussion of Possible Questions	
Total No. of Hours Planned for Unit II : 7			

UNIT III			
SL. No.	**Lecture Duration (Hours)**	**Topics to be Covered**	**Support Materials**
1.	1	Menus, sub-procedures and sub-functions: defining/creating and modifying a menu	S1:168-173
2.	1	using common dialog box	S1:175-180
3.	1	creating a new subprocedure,	S1:181-182
4.	1	passing variables to procedures	S1:182-183
5.	1	passing argument by value or by reference	S1:183-184
6.	1	writing a function/ procedure	S1:184-186
7.	1	Recapitulation and Discussion of Possible Questions	
Total No.of Hours Planned for Unit III : 7			
UNIT IV			
SL. No.	**Lecture Duration (Hours)**	**Topics to be Covered**	**Support Materials**
1.	1	Multiple forms creating, adding, removing forms in project	S1:208-209
2.	1	hide, show method, load, unload statement, me keyword referring to objects on a different forms	S1:210-213
3.	1	Do/loops	S1:252-256
4.	1	for/next loops	S1:256-260
5.	1	using msgbox function ,using string Function	S1:261-265
6.	1	Recapitulation and Discussion of Possible Questions	
Total No. of Hours Planned for Unit IV : 6			
UNIT V			
SL. No.	**Lecture Duration (Hours)**	**Topics to be Covered**	**Support Materials**
1.	1	Arrays - 1-dimension arrays, initializing and array using for each	S1:293-296
2.	1	user-defined data types, accessing information with user-defined data types using list boxes with array, two dimensional arrays	S1: 297-298 S1:303-308
3.	1	lists, loops and printing list boxes & combo boxes, filling the list using property window / add item method, clear method	S1:246-249
4.		list box properties, removing an item from a list, list box/ combo box operations.	S1:249-252
5.	**1**	**Database Connectivity: Database connectivity of forms with back end tool like mysql populating the data in text boxes, list boxes etc.**	**W3**
6.	1	Searching of data in database using forms Updating/	W3

		editing of data based on a criterion	
7.	1	Recapitulation and Discussion of Possible Questions	
8.	1	Discussion of Previous ESE Question Papers	
9.	1	Discussion of Previous ESE Question Papers	
Total No. of Hours Planned for Unit V : 9			

SUGGESTED READINGS

S1. Programming in Visual Basic 6.0 by Julia Case Bradley, Anita C. Millispangh. 2014. Tata Mcgraw Hill Edition.

WEB SITES

W1. https://www.tutorialspoint.com/vb.net

W2. howtostartprogramming.com/vb-net/

W3. https://www.vbtutor.net

Unit I - GUI ENVIRONMENT

Introduction Graphics User Interface (GUI), Programming Language (Procedural, Object oriented, event driven), The Visual Basic environment IDE, Introduction to VB Controls: Textboxes, Frames, check boxes, options buttons, setting a border and style, the shape control, the line control, working with multiple controls and their properties, designing the user interface, keyboard access, tab controls, Default & controls property, coding for Controls.

Introduction to Visual Basic

VISUAL BASIC is a high level programming language which evolved from the earlier DOS version called BASIC. BASIC means Beginners' All-purpose Symbolic Instruction Code. It is a very easy programming language to learn. Different software companies produced different versions of BASIC, such as Microsoft QBASIC, QUICKBASIC, GWBASIC, and IBM BASICA and so on. However, people prefer to use Microsoft Visual Basic today, as it is a well developed programming language and supporting resources are available everywhere. Now, there are many versions of VB exist in the market, the most popular one and still widely used by many VB programmers is none other than Visual Basic 6. We also have VB.net, VB2005, VB2008 and the latest VB2010. Both Vb2008 and VB2010 are fully object oriented programming (OOP) language. VISUAL BASIC is a VISUAL and events driven Programming Language. These are the main divergence from the old BASIC. In BASIC, programming is done in a text-only environment and the program is executed sequentially. In VB, programming is done in a graphical environment. In the old BASIC, you have to write program code for each graphical object you wish to display it on screen, including its position and its color. However, In VB, you just need to drag and drop any graphical object anywhere on the form, and you can change its color any time using the properties windows. On the other hand, because the user may click on certain object randomly, so each object has to be programmed independently to be able to response to those actions (events). Therefore, a VB Program is made up of many subprograms, each has its own program code, and each can be executed independently and at the same time each can be linked together in one way or another.

Advantages of Visual Basic

- It is not just a language to program in but a whole graphical development environment. This aids your programming skills allowing you to concentrate on developing novel ideas instead of going over old ground.
- It is quick to develop new programs. A newcomer will have a window proudly opening and greeting you with "Hello World!" - always the programmers first program - in less than 5 minutes.

- OLE programming is simple. This allows you to embed objects such as Word documents and Excel spreadsheets with a minimum of fuss.
- It can be used as a front end to SQL (or other databases) allowing the user to enhance the way they access their data.
- It is widely used for in-house application program development andfor prototyping.
- It can also be used to create ActiveX and COM components for useonline or in desktop applications.
- It is very simple to learn. As the name Basic suggests it uses easy to understand and remember terminology.
- Because of its popularity there are many resources available to the user - websites are numerous and books are plentiful for the programmer needing help.
- Strict programming structures can be "turned off" to allow you to quickly develop a program - this could also be seen as a disadvantage when a bug arises!
- Perhaps its strongest advantage is its simplicity. There is hardly any learning curve for programmers to begin learning the language or coming from another language.
- VB is a "component integration language" which utilises Microsoft's Component Object Model ("COM") that allows parts to be bolted onto programs easily. These COM programs can be written in any language.

Disadvantages of Visual Basic

- It is not suited to complex modern programming techniques. Because of its age little is being down to further the VB environment and it has been largely superceded by VB.net and other languages (even by Microsoft!).
- Programs that are written in it tend not to be the quickest, this is largely due to the additional code that is often included that is not really necessary for your program to run.
- VB is an interpreted language which again slows the execution of your program down.
- VB programs require large libraries to be present on your PC to enable them to work. If you do not have them the programmer either has to supply them or you have to download them.
- Because of its over-simplified approach VB can produce programmers that are sloppy in their work leading to workarounds having to be employed.
- Because of its age VB does not allow many modern techniques such asObject Orientated Programming.
- As you can control the checking and warning systems in VB it often enables the programmer to write code that is very difficult to troubleshoot when a bug arises.

- VB consists of features and syntax borrowed from other languages (often ones that are now no longer used). Watch a programmers face when you talk to them about "GoSub" and "On Error" commands!
- OOP - Object Orientated Programming - is missing from VB, this is one of the most common techniques in all new languages allowing code to be easily reused (although this is available in VB.net)
- There is no threading support (although this is also available inVB.net).
- The programs a programmer produces in VB are not portable and cannot be used on non-Windows systems.
- Mathematical performance is poor which slows down the speed of your program.
- Service Packs! Everyone knows about Microsoft's love of service packs to fix the many bugs that have accumulated over time.

Introduction to Graphics User Interface (GUI)

Need of visual basic

Visual Basic is a tool that allows you to develop Windows (Graphic User Interface - **GUI**) applications. Basic denotes method of writing programs code functionality.

Visual Basic is **event-driven**, meaning code remains idle until called upon to respond to some event (button pressing, menu selection ...). Visual Basic is governed by an event processor. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Program control is then returned to the event processor.

Getting Started

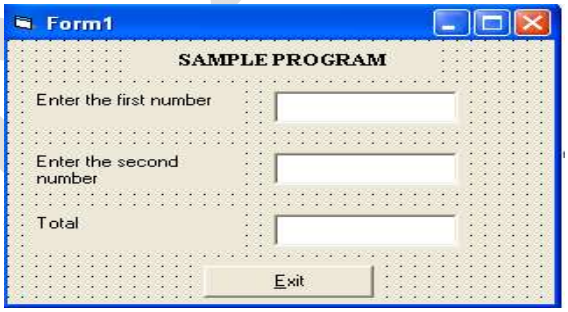


Figure 1.1

The diagram shows a typical Graphical User interface which has been created using Visual Basic. The programming created in Figure 1.1 requires an environment where we can *create* and *modify* the application. This is known as the ***Integrated Development Environment*** and is discussed in the next section below. It Contains Properties, Event Procedures, Forms and Standard Controls

Properties:

The properties describe the appearance of the GUI component. When adding a component, the Name property should be set immediately, according to the three-letter mnemonic naming conventions. The properties are displayed in the Properties Window in Name/Value pairs in alphabetical order.

Event Procedures:

An event procedure is a piece of code that responds to events that can occur for that object. Most of the events are generated by the user, enabling them to dictate the order of execution.

Forms:

The Form is the main stage of your application. By default, the Standard Exe option starts with a form called "Form1". The Name property of the Form should be named with a three-letter mnemonic prefix of "frm". Each Form will be a Window in your application. Controls are added to the form by either double-clicking them in the toolbox, or by selecting the control and drawing a bounding rectangle on the form. Your application may use more than one form.

To add a new Form to the project, either select "Add Form" from the "Project" menu or right-click the Forms folder in the Project Explorer and select, "Add", and then "Form".

To load a new form, use the Show method. The parameter, vbModal, is optional. If used, vbModal means that the form has focus until closed within the application.

The Load command can be used to load a form without showing it. This technique is useful if you want to preload a form, and then use either the "Show" or "Visible" method to make it visible as and when required.

Standard Controls:

Controls are added to the Form from the Toolbox. Each control has a set of properties, and a set of event procedures associated with it. The following lists the control, reading left to right, top to bottom as they appear in the standard Toolbox.

- Pointer.
- PictureBox Control.
- Label Control.
- TextBox Control.
- Frame Control.
- CommandButton Control.
- CheckBox Control.
- OptionButton Control.
- ComboBox Control.

- ListBox Control.
- Horizontal and Vertical Scroll bars.
- Timer Control.
- DriveListBox, DirListBox, and FileListBox.
- Shape Control.
- Line Control.
- Image Control.
- Data Control.
- Object Linkking and Embedding (OLE) Control.

Programming Language (Procedural, Object Oriented, Event Driven)

Programming Language

A program is a set of instructions following the rules of the chosen language.

• Without programs, computers are useless.

• A program is like a recipe.

• It contains a list of ingredients (called variables) anda list of directions (called statements) that tell thecomputer what to do with the variables.

You eventually need to convert your program intomachine language so that the computer can understand it. There are two ways to do this:

– Compile the program

– Interpret the program

Procedural Language

The Procedural Programming Approach to Programming We will begin this course with a brief discussion of the programming methodologies that you are most likely accustom to in your previous Visual Basics.Net introductory courses. Programming as it was done in the past and still being done today in many cases is based on the Event-Driven and Procedural Programming approach. These methods of programming are based on what's known as Structured Programming. Structure programming has been the traditional way of programming.

Procedural Programming If you have taken a course in C, Visual Basic, Pascal, FORTRAN, Cobol etc. the programs you wrote were Procedural. In procedural programming, the focus of the programs was to solve a problem. For example, supposed you were asked to write a program to solve the following problem: Write a Video Management System that will process the rental/return of videos tapes for a retail store such as a program used for Blockbuster Video. Using a language like C or can be done even with VB.NET, this is usually done as follows:

1. Analyze the problem required to be solved: Design flow chart, algorithm etc.

2. Break the problem into smaller manageable pieces, such as the Rental Module, Return Module, Customer Information Module etc.

3. Design the UI for each of the pieces using Forms , controls or any structure supplied by the language to implement the UI

4. Write code to implement each piece using variables, functions & procedures to implement each of the modular pieces. Note that the focus is on solving the problem via the programming code and breaking the problem into smaller manageable pieces.

Object Oriented

- Visual Basic provides full support for object-oriented programming including encapsulation, inheritance, and polymorphism.
- Encapsulation means that a group of related properties, methods, and other members are treated as a single unit or object.
- Inheritance describes the ability to create new classes based on an existing class.
- Polymorphism means that you can have multiple classes that can be used interchangeably, even though each class implements the same properties or methods in different ways.This section describes the following concepts:
- Classes and objects
    - Class members

    Properties and fields

    Methods

    Constructors

    Destructors

    Events

    Nested classes
    - Access modifiers and access levels
    - Instantiating classes
    - Shared classes and members
    - Anonymous types
- Inheritance
    - Overriding members
- Interfaces
- Generics
- Delegates

Access modifiers and access levels

All classes and class members can specify what access level they provide to other classes by using *access modifiers*.

The following access modifiers are available:

Visual Basic Modifier	Definition
Public	The type or member can be accessed by any other code in the same assembly or another assembly that references it.
Private	The type or member can only be accessed by code in the same class.
Protected	The type or member can only be accessed by code in the same class or in a derived class.
Friend	The type or member can be accessed by any code in the same assembly, but not from another assembly.
Protected Friend	The type or member can be accessed by any code in the same assembly, or by any derived class in another assembly.

Event Driven

Visual Basic is an event-driven programming language. Before proceeding to the next chapter, it is very important to have a good concept of event-driven programming. The common events are Click, DblClick, Load, MouseMove, MouseDown, MouseUp, KeyPress, KeyUp, KeyDown, GotFocus, LostFocus, etc.

When you click, press a key, move the mouse or fire other events, the particular block of code of the corresponding event procedure is executed, and then the program behaves in a certain way. This is called event-driven programming.

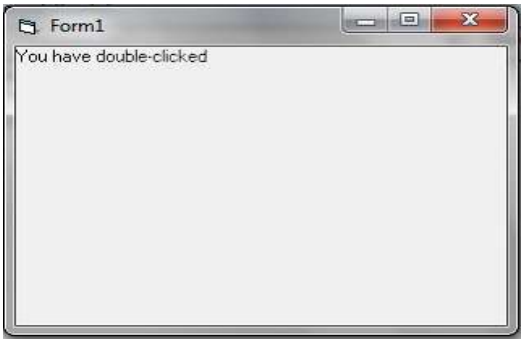
When you fire an event, the code in the event procedure is executed, and then visual basic performs its operations as per the instructions written in the event procedure code. For example, in the first sample program, when you click the 'Print' button, the click event is fired, and then the code in the click event procedure gets executed. The code tells Visual Basic to print a text on the form. So as a result, you see a text printed on the form.

Example:

Write the following code in the DblClick event procedure of the form.

```
Private Sub Form_DblClick()
   Print "You have double-clicked"
End Sub
```

Output:



When you double-click on the form, the DblClick event procedure of the Form object is invoked, and then the code in the DblClick event procedure is executed. Thus, the code instructs Visual Basic to print a text on the form.

The Visual Basic Environment IDE

To launch Visual Basic, on the Taskbar, click Start -> (All) Programs -> Microsoft Visual Studio 6.0 -> Microsoft Visual Basic 6.0. When Microsoft Visual Basic starts, the New Project dialog box comes up:

Figure 1.2 The Visual Basic Start-up Dialog Box

Double Click on the Standard EXE option (or) Select the default highlighted icon and click an open button. You will be then placed in Visual Basic's design environment as illustrated in Figure 1.3.To open the existing project, select File Menu -> open project.

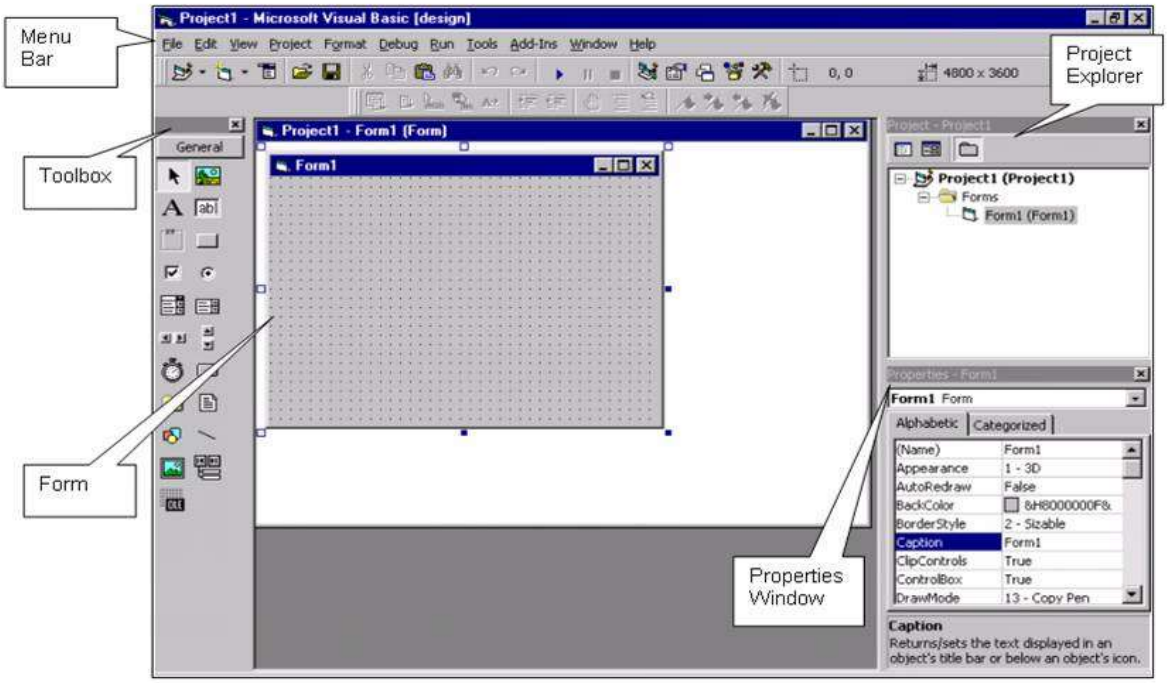


Figure 1.3 Initial Visual Basic Screens

Visual Basic is a high level programming paradigm. Its concepts are based upon Event driven programming. The environment to edit, delete and write code as well as develop windows based applications is known as the **'Integrated Development Environment' (IDE)**.

From the diagram it can be seen that the IDE is divided into separate areas or '*windows*'. We have the Toolbox control which allows us to add objects on to Form window. We can change the properties using the properties windows for all the objects on the form. We can also edit/create the event handlers using the Code Window. When creating applications in Visual Basic it is quite common to use multiple forms, modules etc. The project explorer window is used to keep track of all the additional files used.

The main components of the Integrated Development Environment (IDE) are illustrated in the subsequent text.

Title Bar and Menu Bar

The **title bar** is the horizontal bar located at the top of the screen. It gives the name of the application and is common to all windows application. Everything below the title bar and menu bars in a window application is called the **client area.**

In Visual Basic, the title bar starts out displaying:

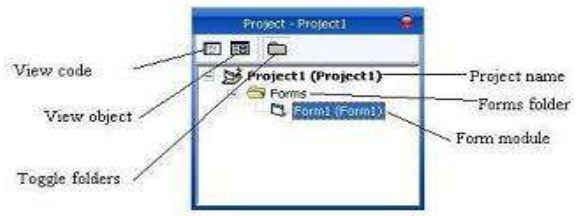
Project1 – Microsoft Visual Basic [design]

The **Menu bar** gives you accesses too many features within the development environment.

1. On the left is the File Menu. It contains the commands such as you can create, open, print and save projects. All of this menu can also be accessed y right-clicking in the project explorer.
2. Next to File is the Edit menu. From here you can perform many of the editing tools that will help you write the code that activates the interface you design for your application, including the search-and-replace editing tools.
3. The View menu gives you fast access to the different parts of your program and to the different parts of the Visual Basic environment.
4. The Project menu is the heart of your project. You can add to and remove forms, code modules, user controls, property pages, as well as ActiveX designers from your projects.
5. The Format menu gives you a way to specify the look of controls that you will place on your forms.
6. The Debug menu contains the tools used to correct (debug) problems, or bugs in your code.
7. The Run menu gives you the tools needed to start and stop your program while in the development environment.
8. The Query and Diagram menu are mostly used in advanced database development. The commands in the Query menu simply the creation of SQL queries. The Diagram menu is used for building database application.
9. The Tools menu gives you access to ways of adding procedures and menus to your programs
10. The Add-Ins menu contains additional utilities called Add-Ins. By default you should have an option for Visual Data Manager and another for the Add-In Manager. Visual Data Manager is a simple but useful tool that allows you to design and populate a database in many popular formats, including Microsoft Access. The Add-In Manager allows you to select other Add-In utilities to be added to the Add-Ins menu.
11. The Window menu lets you control how the windows that make up the visual Basic environment are arranged.
12. Finally, the Help menu is your second stop when you get in a jam.

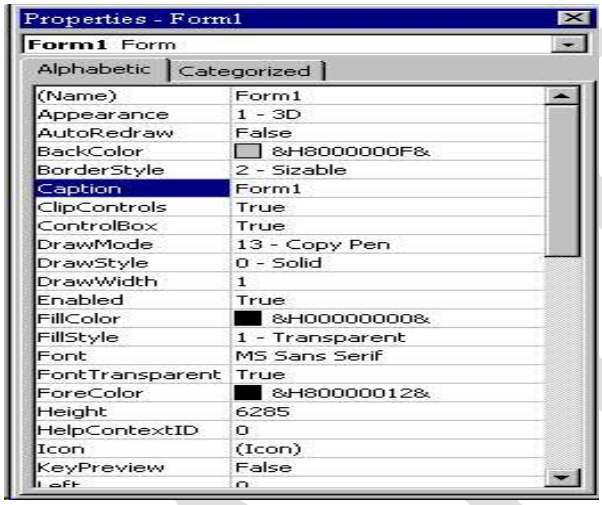
    Notice that all menus have one letter underlined. Pressing ALT and the underlined letter open that menu.

Project Explorer



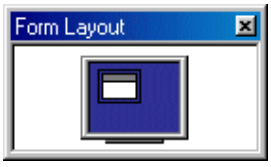
Docked on the right side of the screen, just under the tool bar is the Project Explorer Window. The Project Explorer Window as show above serves as a quick reference to the various elements of a project namely for, classes and modules. The entire object that makes up the application is packed in a project. It looks like a tree-like structure. A simple project will typically contain one form, which is a window that is designed as part of a program's interface. The three tools in the top of the Project Explorer are described in the following table.

Properties Window



The Properties Window is docked under the Project Explorer window. The Properties Window exposes the various characteristics of selected objects. Each and every form in an application is considered an object. Now, each object in Visual Basic has characteristics such as color and size. Other characteristics affect not just the appearance of the object but the way it behaves too. All these characteristics of an object are called its properties. Thus, a form has properties and any controls placed on it will have properties too. All of these properties are displayed in the Properties Window.

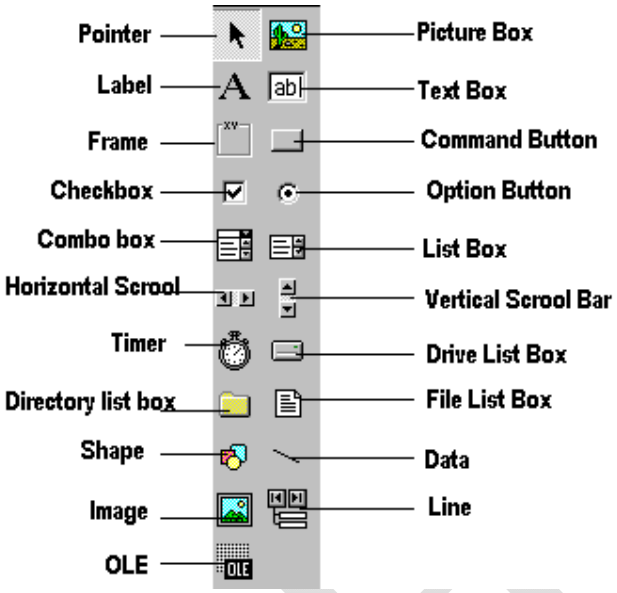
Form Layout Window



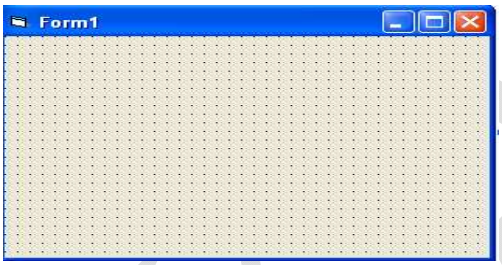
The **Form Layout window** allows you to position the location of the form at run time relative to the entire screen using a small graphical representation of the screen.

Tool Box

The Toolbox contains a set of controls that are used to place on a Form at design time thereby creating the user interface area. Additional controls can be included in the toolbox by using the Components menu item on the Project menu.



Form Designer



It is used to design the interface application. We add controls, graphics and pictures to a form to create the way we want. Each form in the application has its own form designer windows.

Object Browser

The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the key F2. The left column of the Object Browser lists the objects and classes that are available in the projects that are opened and the controls that have been referenced in them. It is possible for us to scroll through the list and select the object or class that we wish to inspect. After an object is picked up from the Classes list, we can see its members (properties, methods and events) in the right column. A property is represented by a small icon that has a hand holding a piece of paper. Methods are denoted by little green blocks, while events are denoted by yellow lightning bolt icon.

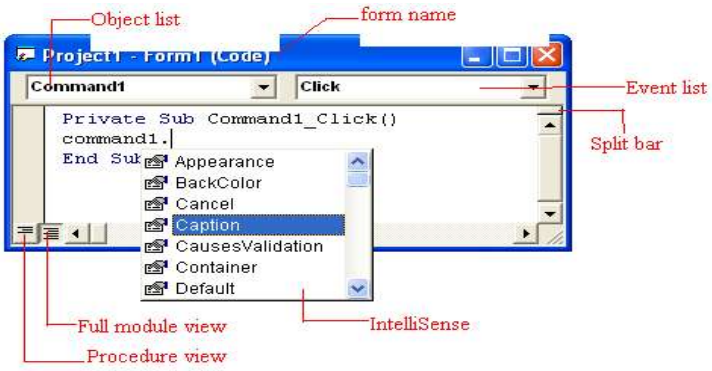
Immediate, Locals, And Watch Windows

These additional windows are provided for use in debugging your application. They are only available when you are running your application within the IDE.

The Code Window

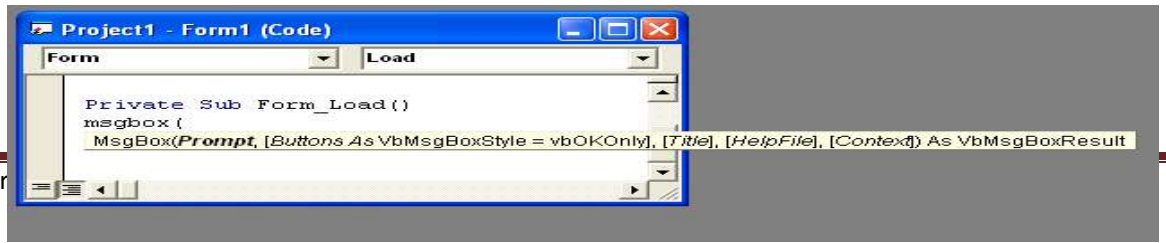
The Code Window is where you write Visual Basic Code for your application. Code consists of languages of statements, constants and declarations. Using the code window you quickly view and edit any of the code in your application. The Code window opens whenever you double-click a control or form or From Project Explorer Window select the name of a form or module or you can choose the View Code icon or View→ Code.

The Code Window has the following sections see figure



Figure

- ***The Split Bar***: The Split bar is located below the title bar at the top of the vertical scroll bar. Split bar is used when complicated codes are performed in your application. That is the window is splitted into two different parts of your code at once.
- ***The Object List Box:*** The left drop-down list box in the code window, called the object box. Lists all the objects on the form.
- ***The Event List Box:*** The right drop-down list box in the code window, called the Event List box. This Box gives the events recognized by the object you have selected in the object list box.
- ***IntelliSense***: IntelliSence saves you a lot of typing work. Its purpose is to display a pop up little boxes with helpful information about the object you are working with. It works with three components.
- ***Quickinfo:*** You get the information about the syntax. Whenever you enter a keyword followed by a space or opening parenthesis, a tip appears and gives the suntax for that element. For Example, Quickinfo feature work for the Msgbox Statement, which pops up a



simple box.

Opening an Existing Visual Basic Project

There are certain default visual basic projects kept in the default directory/samples/for the users. To open them click on File - > Open Project. Double click on the samples folder and then on any project to load it.

Opening a New Visual Basic File and Inserting Source Code

There are three primary steps involved in building a Visual Basic application:

- Draw the user interface
- Assign properties to controls
- Attach code to control.

To create a project, Choose 'New Project' from the 'File' menu. Place the required controls from the toolbox on your form. Write the code in the code.

Running and Viewing the Project in Detail

To run a project, click "Debug" from the menu bar. Click "Start" from the drop down menu. Alternatively, you can press F5 on your keyboard. The project window shows the projects and the forms in them. To view click on them and open.

Figure 3.5: Alternative Way to Start an Application

Making Your First *.exe

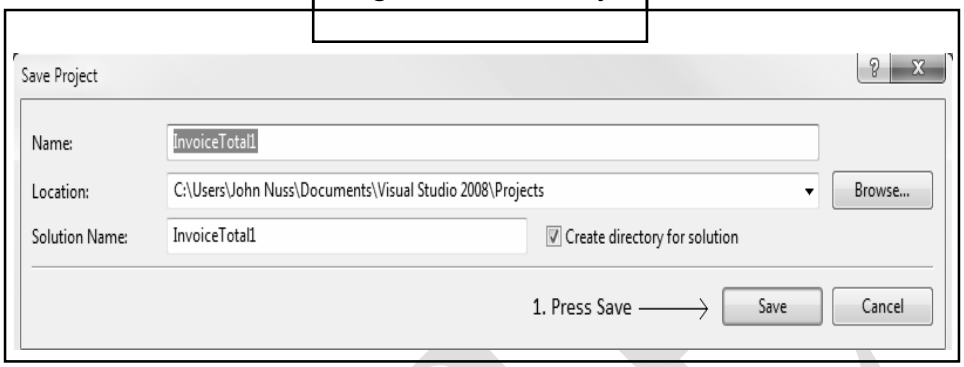
You can create .exe file of your project which a executable file. In windows after clicking on the .exe file project starts running. The simple steps for creating .exe file are as follows:

1. After completing project save it.
2. Then click on file menu.
3. In file menu there is an sub menu 'Make Project'.
4. After clicking on Make Project sub menu it will open a save dialog box to locate the path where .exe file will be created. Select the location and click on Ok then your project will be converted into .exe file at specified location.

Saving Your Visual Basic Project

It is recommended that you save your work at frequent intervals by executing the File ! Save All option from the File Menu. The first time that you perform a File ! Save All operation, Windows will notice that there is no previously saved copy of the project. This will give you the opportunity to

Figure 3.6: Save a Project



save your project under a different name or in a different location. This is especially useful when you are making a copy of a previously written program that you would like to start as a new project. By selecting the *Browse* button, you may make changes as you deem appropriate. At this time, however, for simplicity, we are going to keep all the default values as Windows as selected them for us. Press *Save* as indicated in the diagram below:

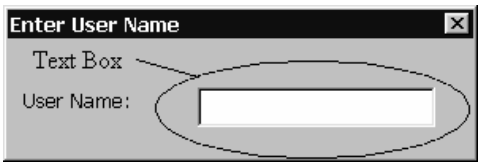
Properties of Controls

All the controls in the Toolbox except the Pointer are objects in Visual Basic. These objects have associated properties, methods and events. Real world objects are loaded with properties. For example, a flower is loaded certain color, shape and fragrance. Similarly, programming objects are loaded with properties. A property is a named attribute of a programming object. Properties define the characteristics of an object such as Size, Color, etc. or sometimes the way in which it behaves. For example, a Textbox accepts properties such as Enabled, Font, MultiLine, Text, Visible, Width, etc.

Textbox Control

Textbox controls offer a natural way for users to enter a value in your program. For this reason, they tend to be the most frequently used controls in the majority of Windows applications. Textbox controls, which have a great many properties and events, are also among the most complex intrinsic controls.

Figure 3.7: Textbox



Some important properties of a textbox are summarized in the table below:

Property	Changes
Name	TextBox Name
Text	Text
MultiLine	Whether the text box can display multiple lines
BackColor	Background Colour
ForeColor	Text Colour
BackStyle	Back style – Opaque or Transparent
SelStart	Start Position of the selection
SelLength	Length of the selection

Appearance Section Properties

The properties that can be used to alter the appearance of a textbox are:

- ***Text Align:*** Gets or sets how text is aligned in a Textbox control. This property has values: Left, Right and Center.

- ***Scrollbars:*** Gets or sets which scroll bars should appear in a multiline Textbox control. This property has values: None, Horizontal, Vertical and Both.

Textbox Event

TextChanged Event is the default event of a textbox.

Protected Sub TextBox1_TextChanged(ByVal sender As Object, _ByVal e As System.EventArgs) Handles TextBox1.TextChanged

Label1.Text = Server.HtmlEncode(TextBox1.Text) End Sub

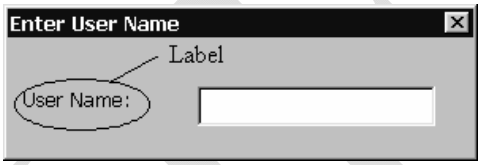
Example: Insert two text boxes into the form together with a few labels. The two text boxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two text boxes. A command button is also programmed to calculate the sum of the two numbers using the plus operator. The program creates a variable, 'sum', to accept the summation of values from Textbox 1 and textbox 2. The procedure to calculate and display the output on the label.

```
Private Sub Command1_Click()
Sum = Val(Text1.Text) + Val(Text2.Text) // To add the values in Textbox 1  and  2
Label1.Caption = Sum // To display the answer on label 1  End  Sub
```

Label Control

The label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs. One of its most important properties is

Figure 3.9: Label



Caption. Using the syntax label. Caption, it can display text and numeric data. You can change its caption in the properties window and also at runtime.

Label Event

Click is the default event of Label. Its syntax is,

```
Private Sub Label1_Click(ByVal sender As System.Object, ByVal e  As  System.EventArgs)_
Handles Label1.Click End  Sub
```

Creating a Label in Code

The following example sets a label on the mouse click.

```
Private Sub Label1_Click(sender As Object, e As EventArgs) _ Handles  Label1.Click
Label1.Location = New  Point(50,  50) Label1.Text = "You have just moved the label" End  Sub
```

Command Button

The Command button is a common control frequently used in VB programming. This is one of the first controls that beginners learn to place onto a form and then to code an "event procedure" for the command control object. An event procedure is something that happens in response to a user action such as a click or a "mouse over".

Figure 3.8: Output of the Code

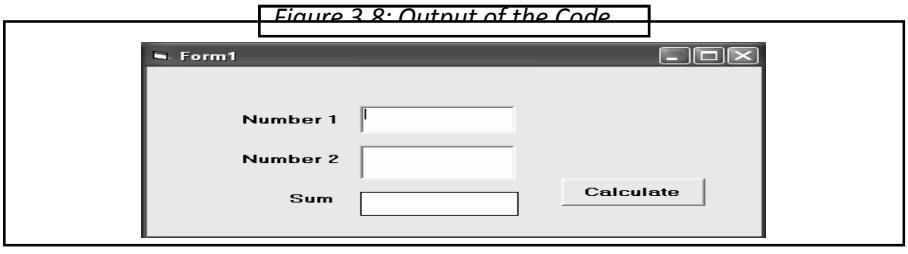
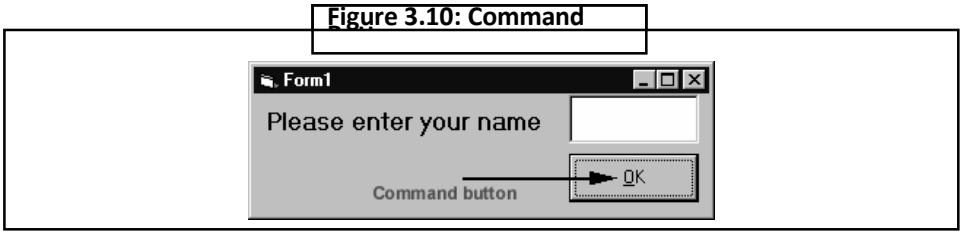


Figure 3.10: Command



Properties of Command Button

The properties of a command button has been discussed in detail in section 3.3.

Methods

Move: The move method changes the position of the command button. We can set the position where we want to move the button to.

SetFocus: You can use this method when you want a particular field or control to have the focus so that all user input is directed to this object.

Events

An event occurs in response to an action, such as the user clicking the mouse or pressing a key in the application. An event handler is a routine that responds to the event. Some events of a command button are:

Click: Determines what action to take when the user clicks a command button.

```
Private Sub CommandButton1_Click ()
If CommandButton1.Caption = "OK" Then 'Check caption, then change it.
  CommandButton1.Caption = "Clicked" End If
End Sub
```

GotFocus: If the control you move to on the sub-form previously had the focus, neither its Enter event nor its GotFocus event occurs, but the Enter event for the sub-form control does occur. If you move the focus from a control on a sub-form to a control on the main form, the Exit and LostFocus events for the control on the sub-form don't occur, just the Exit event for the sub-form control and the Enter and GotFocus events for the control on the main form.

```
Private Sub OptionYes_GotFocus()
  Me!LabelYes.Caption = "Option button 'Yes' has the focus."  End Sub
```

KeyDown and KeyUp: The **KeyDown** event occurs when the user presses a key while a form or control has the focus and the **KeyUp** event occurs when the user releases a key while a form or control has the focus.

```
Private Sub KeyHandler_KeyDown(KeyCode As Integer, _ Shift As Integer)
```

```
Dim intShiftDown As Integer, intAltDown As Integer
Dim intCtrlDown As Integer
‘ Use bit masks to determine which key was pressed. intShiftDown = (Shift And acShiftMask) >
0
‘ Display message telling user which key was pressed.
  If intShiftDown Then MsgBox “You pressed the SHIFT key.” End Sub
```

KeyPress: The **KeyPress** event occurs when the user presses and releases a key or key combination that corresponds to an ANSI code while a form or control has the focus.

```
Private Sub ShipRegion_KeyPress(KeyAscii As Integer)
Dim strCharacter As String
‘ Convert ANSI value to character string. strCharacter = Chr(KeyAscii)
‘ Convert character to upper case, then to ANSI value. KeyAscii = Asc(UCase(strCharacter))
End Sub
```

LostFocus: The **LostFocus** event occurs when the specified object loses the focus.

```
Private Sub OptionYes_LostFocus() Me!LabelYes.Caption = “” ‘ Clear caption.
End Sub
```

MouseDown, MouseUp: The **MouseUp** event occurs when the user releases a mouse button and the **MouseDown** event occurs when the user presses a mouse button.

```
Private Sub Form_MouseDown(Button As Integer, _ Shift As Integer, X As Single, _
Y As Single)
If Button = acLeftButton Then
MsgBox “You pressed the left button.” End If
If Button = acRightButton Then
MsgBox “You pressed the right button.” End If
If Button = acMiddleButton Then
MsgBox “You pressed the middle button.” End If
End Sub
```

CheckBox Control

The CheckBox control is similar to the option button, except that a list of choices can be made using check boxes where you cannot choose more than one selection using an OptionButton. By ticking the CheckBox the value is set to True. This control can also be grayed when the state of the CheckBox is unavailable, but you must manage that state through code.

Notable Properties

S.No.	Property	Description
1.	Appearance	Gets or sets a value determining the appearance of the check box.
2.	AutoCheck	Gets or sets a value indicating whether the Checked or CheckedState value and the appearance of the control automatically change when the check box is selected.
3.	CheckAlign	Gets or sets the horizontal and vertical alignment of the check mark on the check box.
4.	Checked	Gets or sets a value indicating whether the check box is selected.
5.	CheckState	Gets or sets the state of a check box.
6.	Text	Gets or sets the caption of a check box.
7.	ThreeState	Gets or stets a value indicating whether or not a check box should allow three check states rather than two.

CheckBox Event

CheckedChange event is the default event of the CheckBox.

Private Sub CheckBox4_CheckedChanged(sender As Object, _e As EventArgs) Handles CheckBox4.CheckedChanged

Label1.Visible = True TextBox1.Visible = True

End Sub

Working with CheckBoxes

Let us now create a small example using checkboxes.

Private Sub Form1_Load(sender As Object, e As EventArgs) _Handles MyBase.Load

‘ Set the caption bar text of the form.

Me.Text = “tutorialspoint.com” Label1.Visible = False TextBox1.Visible = False

TextBox1.Multiline = True

End Sub

Private Sub Button1_Click(sender As Object, e As EventArgs) _Handles Button1.Click

Dim str As String str = “ “

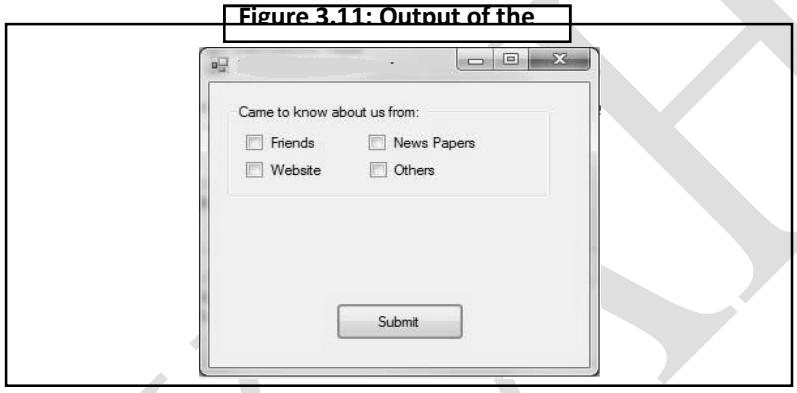
If CheckBox1.Checked = True Then str &= CheckBox1.Text

str &= “ “ End If

If CheckBox2.Checked = True Then str &= CheckBox2.Text

```
 str &= " " End If
 If CheckBox3.Checked = True Then str &= CheckBox3.Text
 str &= " " End If
 If CheckBox4.Checked = True Then str &= TextBox1.Text
 str &= " " End If
If str <> Nothing Then
MsgBox(str + vbLf + "Thank you") End If
End Sub
Private Sub CheckBox4_CheckedChanged(sender As Object, _e As EventArgs) Handles
CheckBox4.CheckedChanged
Label1.Visible = True
TextBox1.Visible = True
End Sub
```

Figure 3.11: Output of the



Frame Control

Frame controls are similar to Label controls in that they can serve as captions for those controls that don't have their own. Moreover, Frame controls can also (and often do) behave as containers and host other controls. In most cases, you only need to drop a Frame control on a form and set its Caption property. If you want to create a borderless frame, you can set its BorderStyle property to 0-None.

Controls that are contained in the Frame control are said to be child controls. Moving a control at design time over a Frame control—or over any other container, for that matter—doesn't automatically make that control a child of the Frame control. After you create a Frame control, you can create a child control by selecting the child control's icon in the Toolbox and drawing a new instance inside the Frame's border. Alternatively, to make an existing control a child of a Frame control, you must select the control, press Ctrl+X to cut it to the Clipboard, select the Frame control, and press Ctrl+V to paste the control inside the Frame. If you don't follow this procedure and you simply move the control over the Frame, the two controls remain completely independent of each

other, even if the other control appears in front of the Frame control.

Frame controls, like all container controls, have two interesting features. If you move a Frame control, all the child controls go with it. If you make a container control disabled or invisible, all its child controls also become disabled or invisible. You can exploit these features to quickly change the state of a group of related controls.

Frame Control Property, Method, and Events

Frame Properties

Some of the properties of the frame control are as follows:

***Caption*:** Text used to label the frame. Font is set in the font tab.

MousePointer: The type of mouse pointer displayed when over the frame. Choose from 16 different types plus a custom pointer.

***Appearance*:** 3D or Flat. 3D is the standard Windows look. Flat results in a rectangular square of background color with a foreground color border and caption.

***Border Style*:** FixedSingle (default) or None. None may be used with Flat Appearance to achieve an uncaptioned rectangle of background color.

***OLEDropMode*:** Set to None (default) if the frame does not accept OLE drops and is to display the No Drop cursor if an OLE drop is attempted. Set to Manual, the frame will trigger OLE drop events, allowing programs to handle the OLE drop operation in code.

***Enabled*:** True or False. True means the object can respond to user-generated events, false prevents it from responding. We can disable a control like a text box if we wish it to simply display information in a read-only way.

Back Color: Color to be used for the frame's background, normally seen only when Flat Style is used.

***Fore Color*:** Color to be used for the frame's foreground, the color of the text caption.

***Color Set*:** Choose from Standard Colors (standard Windows non-dithered colors) or Windows System Colors. The latter will be defined by the user's Control Panel settings and is normally the setting used so that the form changes appearance like the rest of Windows if the Control Panel settings are changed.

***Color Palette*:** Displays available colors. Click on the property to be changed to highlight it in the Properties pane, click on the desired color in the Color Palette pane and then press Apply.

***Edit Custom Color*:** Change the custom color presented in the Color Palette when the Color Set is set to Windows System Colors.

Font: Choose a font installed on this system. It's wise to choose standard Windows fonts such as MS Sans Serif that are universally available.

Size: Size of font, in points.

Effects: Bold, Italic, Underline or Strikeout.

Frame Methods

Some of the frame methods are – Drag , Move, OLEDrag, ShowWhatsThis and ZOrder.

Frame Events

The two major events of the frame control are Click and DblClick.

Option Buttons

They are used to provide a set of mutually exclusive options. The user can select one option button in a group. If you need to place more than one group of option buttons in the same form, you should place them in different container controls like a GroupBox control.

Changing the Background Color

To change the background color of the page as per the option button chosen, we use the following code,

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load ' Set the caption bar text of the form.
  Me.Text = "tutorialspont.com" End Sub
Private Sub RadioButton1_CheckedChanged(sender As Object, _e As EventArgs) Handles RadioButton1.CheckedChanged
  Me.BackColor = Color.Red End Sub
Private Sub RadioButton2_CheckedChanged(sender As Object, _e As EventArgs) Handles RadioButton2.CheckedChanged
  Me.BackColor = Color.Green End Sub
```

Setting a border and Style, The Shape Control, The Line Control

How to Set Border Style

You can use BorderStyle property to add a border to the label control. This property is typically used to differentiate a Label that labels another control from a Label that displays the status of a process in an application.

The following code example demonstrates how to set different type of borde stype to a Label control.

FixedSingle :

```
 Private Sub CreateBorder_FixedSingle()
   Label1.BorderStyle = BorderStyle.FixedSingle
 End Sub
```

Fixed3D :

```
 Private Sub CreateBorder_Fixed3D()
   Label1.BorderStyle = BorderStyle.Fixed3D
 End Sub
```

Using the Shape control

There are four basic controls in VB6 that you can use to draw graphics on your form: the line control, the shape control, the image box and the picture box. To draw a straight line, just click on the line control and then use your mouse to draw the line on the form. After drawing the line, you can then change its color, width and style using the BorderColor, BorderWidth and BorderStyle properties.Similarly, to draw a shape, just click on the shape control and draw the shape on the form. The default shape is a rectangle, with the default shape property set at 0. You can change the shape to square, oval, circle and rounded rectangle by changing the shape property's value to 1, 2, 3 , 4, and 5 respectively. In addition, you can change its background color using the BackColor property, its border style using the BorderStyle property, its border color using the BorderColor property as well its border width using the BorderWidth property.

Example

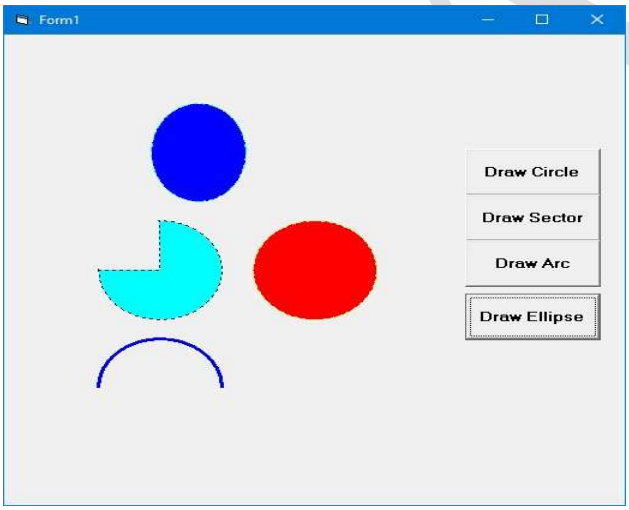
This example will draw a circle, a sector, an arc and an ellipse with different colors and different DrawStyles

The Code

```
Dim pi As Single
Private Sub Form_Load()
'Change the coordinates of the origin
Scale (-4000, 4000)-(4000, -4000)
Me.Width = 8000
Me.Height = 8000
End Sub
Private Sub cmd_Draw_Click(Index As Integer)
   pi = 4 * Atn(1)
Select Case Index
Case Is = 0
Me.FillColor = vbRed
Me.FillStyle = vbSolid
Me.DrawStyle = vbDot
Me.DrawWidth = 1
Me.Circle (0, 0), 800, vbYellow ' Draw a circle with red border
Case Is = 1
Me.FillColor = vbCyan
Me.FillStyle = vbSolid
```

```
Me.DrawStyle = vbDot
Me.DrawWidth = 3
Me.Circle (-2000, 0), 800, vbBlack, -pi, -pi / 2
Case Is = 2
Me.FillColor = vbYellow
Me.FillStyle = vbSolid
Me.DrawStyle = vbDashDot
Me.DrawWidth = 1
Me.Circle (-2000, -2000), 800, vbBlue, 0, pi
Case Is = 3
Me.FillColor = vbBlue
Me.FillStyle = vbSolid
Me.DrawStyle = vbDashDotDot
Me.DrawWidth = 1
Me.Circle (-1500, 2000), 800, vbCyan, , , 1.3
End Select
End Sub
```

The Output



The Line Control

The Line Method

Although the Pset method can be used to draw a straight line on the form, it is a little slow. It is better to use the Line method if you want to draw a straight line faster. The format of the Line command is shown below. It draws a line from the point (x1, y1) to the point (x2, y2) and the color constant will determine the color of the line.

Line (x1, y1)-(x2, y2), color

For example, the following command will draw a red line from the point (0, 0) to the point (1000, 2000).

Line (0, 0)-(1000,2000), VbRed

The Line method can also be used to draw a rectangle. The syntax is

Line (x1-y1)-(x2, y2), color, B

The four corners of the rectangle are (x1-y1), (x2-y1), (x1-y2) and (x2, y2)

Another variation of the Line method is to fill the rectangle with a certain color. The syntax is

Line (x1, y1)-(x2, y2), color, BF

If you wish to draw the graphics in a picture box, you can use the following syntaxes

Picture1.Line (x1, y1)-(x2, y2), color

Picture1.Line (x1-y1)-(x2, y2), color, B

Picture1.Line (x1-y1)-(x2, y2), color, BF

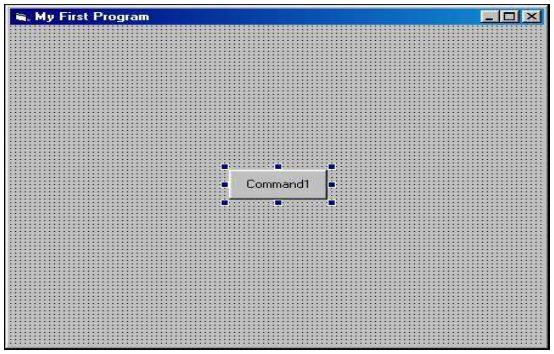
Picture1.Circle (x1, y1), radius, color

Working with multiple controls and their properties

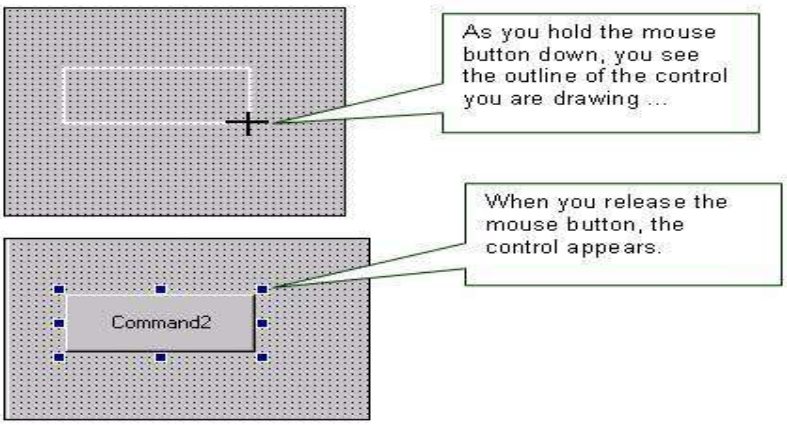
Placing controls on forms

There are two basic ways to place a control on a form:

(1) Double-click the desired control in the toolbox (a command button was used in the example). VB automatically places the control in the middle of the form, as shown below: You can then move and resize the control on the form as desired with your mouse.



(2) Click the desired control in the toolbox once. The mouse cursor becomes a "crosshair" when you move your mouse over the form, and you can draw the control on the form "manually".



Selecting Multiple Controls

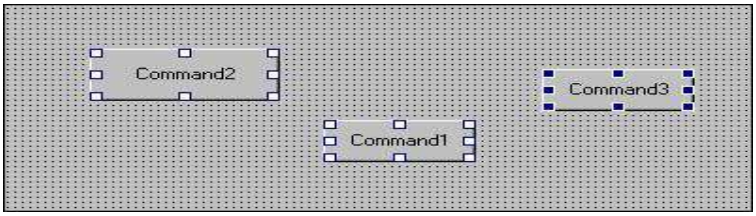
Often, it is convenient to select more than one control at the same time. This enables you to apply the same action to the selected controls (such as moving, formatting, and deleting) simultaneously, which can save time. There are two ways to select multiple controls:

(1) Click on one control with your mouse. Then, hold the **Ctrl** key down on the keyboard, and click the other controls that you want to select.

(2) You can "lasso" the desired controls with your mouse. Hold the mouse button down, and "sweep" the mouse over the controls you want to select. When you release the mouse, the controls should be selected.

Note: To de-select a control in a group of selected controls, Ctrl-click the control.

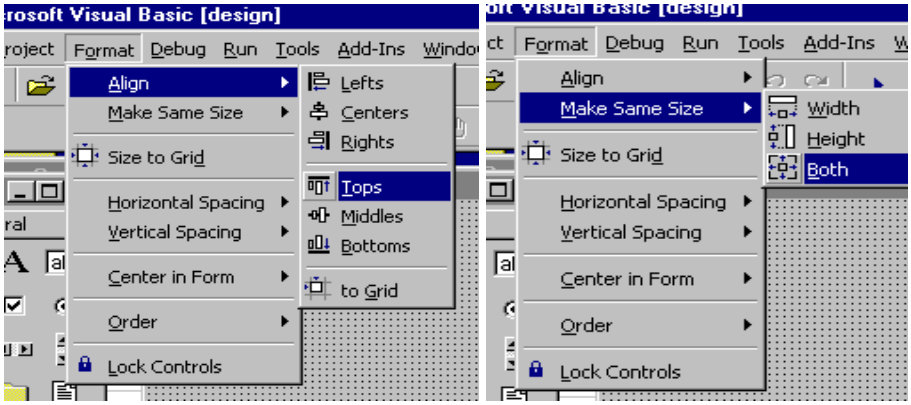
The screen shot below shows three command buttons that have been selected:



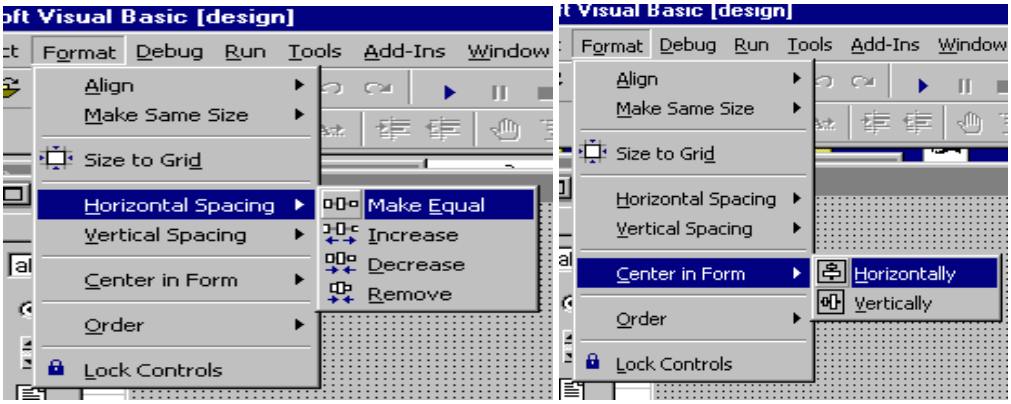
Formatting Multiple Controls

Often, the controls on your form will be "grouped" logically. You may have a set of textboxes in one area, a set of option (radio) buttons in another area, a set of command buttons at the bottom, etc. To create professional-looking forms, these controls should be "lined up" and spaced properly. The options on VB's **Format** menu can help you achieve this with relative ease. In this example, with the three command buttons selected, use the Format options shown in the screen shots below (**Format à Align àTops**, and **Format à Make Same Size à Both**). Note: VB will use the last control you selected as a guide. For example, when you select three controls, the first two will have white

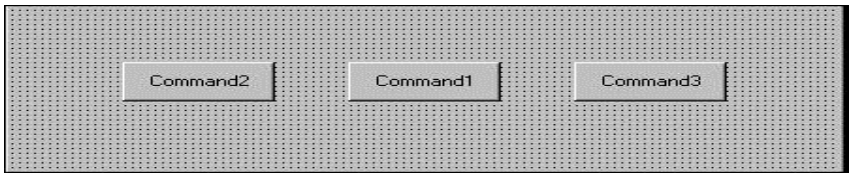
selection handles, and the last one will have blue selection handles. If you then say "align tops", the tops of the first two will be aligned with the third.



Two other Format commands I like are **Horizontal Spacing à Make Equal** (this works with three or more controls – it ensures that the same amount of "white space" exists between all the controls in the selected group) and **Center in Form à Horizontally** (this centers the set of selected controls on the form, giving you an even left and right margin). Note: This example uses a horizontal arrangement of controls, but the same principles apply when you have a vertical arrangement and you use the Vertical Spacing options.



After the formatting commands have been given, the command buttons will look like this:



Controls in Visual Basic

Control	**Description**
Pointer	Provides a way to move and resize the controls form
PictureBox	Displays icons/bitmaps and metafiles. It displays text or acts as a visual container

	for other controls.
TextBox	Used to display message and enter text.
Frame	Serves as a visual and functional container for controls
CommandButton	Used to carry out the specified action when the user chooses it.
CheckBox	Displays a True/False or Yes/No option.
OptionButton	OptionButton control which is a part of an option group allows the user to select only one option even it displays mulitiple choices.
ListBox	Displays a list of items from which a user can select one.
ComboBox	Contains a TextBox and a ListBox. This allows the user to select an ietm from the dropdown ListBox, or to type in a selection in the TextBox.
HScrollBar and VScrollBar	These controls allow the user to select a value within the specified range of values
Timer	Executes the timer events at specified intervals of time
DriveListBox	Displays the valid disk drives and allows the user to select one of them.
DirListBox	Allows the user to select the directories and paths, which are displayed.
FileListBox	Displays a set of files from which a user can select the desired one.
Shape	Used to add shape (rectangle, square or circle) to a Form
Line	Used to draw straight line to the Form
Image	used to display images such as icons, bitmaps and metafiles. But less capability than the PictureBox
Data	Enables the use to connect to an existing database and display information from it.
OLE	Used to link or embed an object, display and manipulate data from other windows based applications.
Label	Displays a text that the user cannot modify or interact with.

Questions	opt1	opt2	opt3	opt4	Answer
visual basic was developed form the programming language _____	FORTRAN	C	BASIC	C++	BASIC
visual bais is developed in the year _____	1978	1980	1970	1972	1970
A complete installation of the most powerful version of visual basic 6.0, the enterprise edition requires _____ of hard disk space	250MB	250GB	460MB	460GB	250MB
Visual Basic 6.0 requires _____ of RAM	18MB	16MB	18GB	16GB	16MB
IDE is _____	Integrated development environment	Integrated developed environment	Integration developme nt environme nt	none	Integrated developmen t environmen t
IDE is also commonly referred to as _____-- environment	interface	developme nt	design	integrated	design
IDE was designed as a _____-	OLE	SDI	MDI	none	SDI
The windows associated with the project will stay with in a single container called _____	child	parent	code	none	parent
quick access to the commonly used commands	toolbars	toolbox	project window	form	toolbars
_____ helps to move and resize the controls and forms	label	shape	OLE	pointer	pointer
action when the user choose it	option button	control	control	button	button
messages and enter text	tool box	text box	button	button	text box
_____ also acts as a coontainer for child forms and some controls	SDI	MDI	IDE	none	MDI
_____ is a window that contains an application code and has other objects place	properties	menu	form	border	form
_____ is an actioon that can be performed on objects	form	move	click	method	method
_____ control enables the user to connect to an existing database and display information from it.	object	properties	data	file	data
_____ serves as a window that can be customized and controls graphics and pictures can also be added to it.	object	properties	form	file	form
Each and every form in an application is considered an _____	object	properties	file	none	object
Characteristics of an object are called its _____	objects	properties	project	none	properties

_____ method is used to display the form object	load	visible	show	display	show
_____ method is used to load a form or control into memory but doesnot display it	load	visible	show	display	load
_____ displays a text that the user cannot modify or interact with	label	text box	timer	line	label
_____ has the zorder event in it.	list box	Ms flex grid	grid	label	list box
_____ displays and operates on tabular data	grid	Msflexgrid	DB grid	none	Msflexgrid
_____ displays information at run time	status bar	option button	text box	rich text box	text box
_____ control is used to display icons, bitmaps, metafiles etc….,	shape control	image control	picture box	check box	image control
the code entered in the _____ event fires when there is a change in the contents of the text box.	change	click	mouse move	none	change
the _____ event fires when the textbox control is clicked.	change	click	mouse move	none	click
the _____ event fires when the mouse is moved over the textbox	change	click	mouse move	none	mouse move
A complete repaint of a form or control can be enforced by _____ method	refresh	set focus	change	repaint	refresh
_____ is used to link or embed object, display and manipulate data from other	.VBP	.FRM	OLE	none	OLE
_____ is the named attribute of a programming object	property	window	object	none	property
_____ is the named attribute of a programming object	window	object	property	none	property
_____ lets windows decide whrer the form should be shown	center screen	manual	center owner	windows default	windows default
_____ indicates whether the window is shown normally, maximized or minimized.	window state	windows default	both	none	windows default
_____ event occurs when the form is closed by user.	load	unload	show	none	unload
Writing a VB program involves _____ steps	2	5	3	4	2
_____ steps involves designing an application with various tools that come along with VB package	visual programming	code programming	both	none	visual programming
_____ steps involves writing programs using text editor.	visual programming	code programming	both	none	code programming
_____ is the name property for the form object	display prog	code programming	form display	none	code programming
_____ is the caption property for the command button if name is cmdexit	&exit	& clear	&display	none	&exit
_____ is the name property for the command button if caption property is &clear.	cmdexit	cmdclear	cmddisplay	none	cmdclear

_____ is the caption property for the command button if name is cmddisplay	&exit	& clear	&display	none	&display
_____ is the name property for the command button if caption is &display	cmdexit	cmdclear	cmddisplay	none	cmddisplay
VB project files are saved with an extension	.VBP	.FRM	.OLE	.EXE	.VBP
form files are saved with an extension	.VBP	.FRM	.OLE	.EXE	.FRM
_____ box displays the name of the selected object associated with the form	object	tool	text	command	object
Intersection of row and column is _____	cell	record	field	none	cell
the cols and rows properties are used to determine the number of columns and rows in a _____ control	flexgrid	ms flexgrid	grid	none	ms flexgrid
_____ event occurs whenever the size of form is ahanged	size	resize	height	width	resize
_____ kinds of rows/columns are created in the msflexgrid contorl.	4	3	2	1	2
Ms flexgrid control is an _____ control	OCX	XCO	COX	XOC	OCX
user can change the current cell at run time using _____	mouse	arrow keys	both	none	both
To display the wrapped text, we need to _____ the cell's column width	increase	decrease	normal	hide	increase
If a cell's text is too long to be displayed in the cell, and the word wrap property is set to _____	FALSE	TRUE	0	1	TRUE
To add the flexgrid, choose complnents from _____ menu	browser	property	project	none	project
After setting the properties, the ms flexgrid control is enlarged vertically and horizontally by _____ its handles.	moving	resizing	hiding	dragging	dragging
Ms flexgrid has the properties _____	row	rows	both	none	both
Ms flexgrid has the properties _____	col	cols	both	none	both
_____ property is set to change the height of a cell	row height	height row	height	none	row height

Unit II – OPERATIONS

Data types, constants, named & intrinsic, declaring variables, scope of variables, val function, arithmetic operations, formatting data.
Decision Making : If statement, comparing strings, compound conditions (and, or, not), nested if statements, case structure, using if statements with option buttons & check boxes, displaying message in message box, testing whether input is valid or not.

Data Types

By default Visual Basic variables are of variant data types. The variant data type can store numeric, date/time or string data. When a variable is declared, a data type is supplied for it that determines the kind of data they can store. The fundamental data types in Visual Basic including variant are integer, long, single, double, string, currency, byte and Boolean. Visual Basic supports a vast array of data types. Each data type has limits to the kind of information and the minimum and maximum values it can hold. In addition, some types can interchange with some other types. A list of Visual Basic's simple data types are given below.

Data Type	Description and Range
Boolean	A data type that takes on one of two values only: True or False.
Byte	8 bits storage capacity, having the range of any character (from 0 to 255) from the ASCII character set.
Currency	Used to hold decimal values used for currency functions. The range is from - $922,337,206,685,477.5808 to $922,337,203,685,477.5807. Note: four decimal places ensure that proper rounding can occur.
Date	Holds the date and time values. The date can range from January 1,100, to December 31,9999.
Double	Double precision used for decimal values. The range for this data type is – 1.7976931348623E+308 to 1.79769313486232E+308.
Integer	Whole number which range from -32,767 to 32,768. The storage capacity for this data type is 2 bytes. Note: cannot be used for decimal calculations.
LongInteger	The data range for an integer is limited, due to its storage capacity. When an integer data type is above its range, there is an over flow and erroneous results occur. LongInteger is more suitable with additional storage capacity and increased range to -2,147,483,648 to 2,147,483,647.
Object	A special data type which references objects such as controls and forms.
Single	Used as alternate data type for double precision known as Single-precision It requires less memory and lower data range from -3.402823E+38 to 3.402823E+38.
String	String data type is an array of characters. Data that consists of 0 to 65,400 characters of alphanumeric data (numeric and alphabetic).
Variant	Data of any data type and used for control and other values for which the data type is unknown.

Visual Basic Data Types

Let us now summarize the storage capacity and range of these data types.

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	–32,768 to 32,767
Long	4 bytes	–2,147,483,648 to 2,147,483,648 –3.402823E+38 to – 1.401298E–45 for negative values.
Single	4 bytes	1.401298E–45 to 3.402823E+38 for positive values. –1.79769313486232e+308 to – 4.94065645841247E–324 for negative values.
Double	8 bytes	4.94065645841247E–324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	–922,337,203,685,477.5808 to 922,337,203,685,477.5807 +/- 79,228,162,514,264,337,693,543,950,335 if no decimal is use
Decimal	12bytes	+/- 7.9228162514264337593543950335 (28 decimal places).

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

Variables

Variables are the memory locations which are used to store values temporarily.

A defined naming strategy has to be followed while naming a variable. A variable name must begin with an alphabet letter and should not exceed 255 characters. It must be unique within the same scope. It should not contain any special character like %, &, !, #, @ or $.

A variable is scoped to a procedure-level (local) or module-level variable depending on how it is declared. The scope of a variable, procedure or object determines which part of the code in our application are aware of the variable's existence. A variable is declared in general declaration section of e Form, and hence is available to all the procedures. Local variables are recognized only in the procedure in which they are declared. They can be declared with Dim and Static keywords. If we want a variable to be available to all of the procedures within the same module, or to all the procedures in an application, a variable is declared with broader scope.

Local Variables

A local variable is one that is declared inside a procedure. This variable is only available to the code inside the procedure and can be declared using the Dim statements as given below:

Dim sum As Integer

The local variables exist as long as the procedure in which they are declared, is executing. Once a procedure is executed, the values of its local variables are lost and the memory used by these variables is freed and can be reclaimed. Variables that are declared with keyword Dim exist only as long as the procedure is being executed.

Static Variables

Static variables are not reinitialized each time Visual Invokes a procedure and therefore retains or preserves value even when a procedure ends. In case, we need to keep track of the number of times a command button in an application is clicked, a static counter variable has to be declared. These static variables are also ideal for making controls alternately visible or invisible. A static variable is declared as given below:

Static intPermanent As Integer

Variables have a lifetime in addition to scope. The values in a module-level and public variables are preserved for the lifetime of an application whereas local variables declared with Dim exist only while the procedure in which they are declared is still being executed. The value of a local variable can be preserved using the Static keyword. The following procedure calculates the running total by adding new values to the previous values stored in the static variable value.

Function RunningTotal ( ) Static Accumulate

Accumulate = Accumulate + num RunningTotal = Accumulate

End Function

If the variable Accumulate was declared with Dim instead of static, the previously accumulated values would not be preserved across calls to the procedure, and the procedure would return the same value with which it was called. To make all variables in a procedure static, the Static keyword is placed at the beginning of the procedure heading as given in the below statement.

Static Function RunningTotal ( )

Example: The following is an example of an event procedure for a CommandButton that counts and displays the number of clicks made.

Private Sub Command1_Click ( ) Static Counter As Integer Counter = Counter + 1

Print Counter End Sub

The first time we click the CommandButton, the Counter starts with its default value of zero. Visual Basic then adds 1 to it and prints the result.

Module Level Variables

A module level variable is available to all the procedures in the module. They are declared using the Public or the Private keyword. If you declare a variable using a Private or a Dim statement in the declaration section of a module—a standard BAS module, a form module, a class module, and so on—you're creating a private module-level variable. Such variables are visible only from within the module they belong to and can't be accessed from the outside. In general, these variables are useful for sharing data among procedures in the same module:

```
//In the declarative section of any module
Private LoginTime As Date ' A private module-level variable
Dim LoginPassword As String ' Another private module-level variable
```

You can also use the Public attribute for module-level variables, for all module types except BAS modules. (Public variables in BAS modules are global variables.) In this case, you're creating a strange beast: a Public module-level variable that can be accessed by all procedures in the module to share data and that also can be accessed from outside the module. In this case, however, it's more appropriate to describe such a variable as a property:

```
//In the declarative section of Form1 module Public CustomerName As String ' A Public property
```

You can access a module property as a regular variable from inside the module and as a custom property from the outside:

```
//From outside Form1 module... Form1.CustomerName = "John Smith"
```

The lifetime of a module-level variable coincides with the lifetime of the module itself. Private variables in standard BAS modules live for the entire life of the application, even if they can be accessed only while Visual Basic is executing code in that module. Variables in form and class modules exist only when that module is loaded in memory. In other words, while a form is active (but not necessarily visible to the user) all its variables take some memory, and this memory is released only when the form is completely unloaded from memory. The next time the form is recreated, Visual Basic reallocates memory for all variables and resets them to their default values (0 for numeric values, "" for strings, Nothing for object variables).

Public versus Local Variables

A variable can have the same name and different scope. For example, we can have a public variable named R and within a procedure we can declare a local variable R. References to the name R within the procedure would access the local variable and references to R outside the procedure would access the public variable.

Variables Declaration

Declaring a variable tells Visual Basic to reserve space in memory. It is not must that a variable should be declared before using it. Automatically whenever Visual Basic encounters a new variable, it assigns the default variable type and value. This is called implicit declaration.

The Dim Statement

Though implicit declaration is easier for the user, to have more control over the variables, it is advisable to declare them explicitly. The variables are declared with a Dim statement to name the variable and its type. The As type clause in the Dim statement allows to define the data type or object type of the variable. This is called explicit declaration.

Syntax

Dim variable [As Type]

Example:

Dim strName As String

Dim intCounter As Integer

Selecting Variable Types

To create a variable we can use any data type as per our need as discussed in the sections above.

Converting Between Data Types

The process of changing a value from one data type to another type is called *conversion*. Conversions are either *widening* or *narrowing*, depending on the data capacities of the types involved. They are also *implicit* or *explicit*, depending on the syntax in the source code.

Implicit Conversion: Visual Basic provides two options that can significantly change the behavior of the editor and compiler. Not using these two options, Option Explicit and Option Strict, can make coding somewhat easier but provide opportunities for hard-to- find errors and very sloppy programming.

If you are converting a value from a narrower data type to a wider type, where there is no danger of losing any precision, the conversion can be performed by an implicit conversion. For example, the statement:

bigNumberDouble = smallNumberInteger

does not generate any error message, assuming that both variables are properly declared. The value of smallNumberInteger is successfully converted and stored in bigNumberDouble. However, to convert in the opposite direction could cause problems and cannot be done implicitly.

The following list shows selected data type conversions that can be performed implicitly in VB:

From	To
Byte	Short, Integer, Long, Single, Double, or Decimal
Short	Integer, Long, Single, Double, or Decimal
Integer	Long, Single, Double, or Decimal
Long	Single, Double, or Decimal
Decimal	Single, Double
Single	Double

- ***Explicit Conversion – casting:*** If you want to convert between data types that do not have implicit conversions, you must use an explicit conversion, also called casting. But beware: If you perform a conversion that causes significant digits to be lost, an exception is generated. Use methods of the Convert class to convert between data types. The Convert class has methods that begin with "To" for each of the data types: ToDecimal, ToSingle, and ToDouble. However, you must specify the integer data types using their .NET class names, rather than the VB data types.

For the VB data type	Use the method for the .NET data type
Short	ToInt16
Integer	ToInt32
Long	ToInt64

Example: The following are examples of explicit conversion. For each, assume that the variables are already declared following the textbook naming standards. You should perform a conversion from a wider data type to a narrower one only when you know that the value will fit, without losing significant digits. Fractional values are rounded to fit into integer data types, and a single or double value converted to decimal is rounded to fit in 28 digits.
numberDecimal = Convert.ToDecimal(numberSingle) valueInteger = Convert.ToInt32(valueDouble)
amountSingle = Convert.ToSingle(amountDecimal)
Variables
Using Option Explicit Statement

When Option Explicit is turned off, you can use any variable name without first declaring it. The first time you use a variable name, VB allocates a new variable of Object data type. For example, you could write the line:

Z = myTotal + 1

without first declaring either Z or myTotal. This is a throwback to very old versions of Basic that did not require variable declaration. In those days, programmers spent many hours debugging programs that had just a small misspelling or typo in a variable name.

You should always program with Option Explicit turned on. In VB .NET, the option is turned on by default for all new projects. If you need to turn it off (not a recommended practice), place the line:

Option Explicit Off

before the first line of code in a file.

Providing Names for Your Variables

Some rules need to be followed while naming your variables. Some of them are:

- A name must begin with a letter.
- May be as much as 255 characters long (but don't forget that somebody has to type the stuff!).
- Must not contain a space or an embedded period or type-declaration characters used to specify a data type; these are ! # % $ & @.

- Must not be a reserved word (that is part of the code, like Option, for example).
- The dash, although legal, should be avoided because it may be confused with the minus sign. Instead of First-name use First_name or FirstName.

Using Option Explicit Statement

When Option Explicit is turned off, you can use any variable name without first declaring it. The first time you use a variable name, VB allocates a new variable of Object data type. For example, you could write the line:

Z = myTotal + 1

without first declaring either Z or myTotal. This is a throwback to very old versions of Basic that did not require variable declaration. In those days, programmers spent many hours debugging programs that had just a small misspelling or typo in a variable name.

You should always program with Option Explicit turned on. In VB .NET, the option is turned on by default for all new projects. If you need to turn it off (not a recommended practice), place the line:

Option Explicit Off

before the first line of code in a file.

Providing Names for Your Variables

Some rules need to be followed while naming your variables. Some of them are:

- A name must begin with a letter.
- May be as much as 255 characters long (but don't forget that somebody has to type the stuff!).
- Must not contain a space or an embedded period or type-declaration characters used to specify a data type; these are ! # % $ & @.
- Must not be a reserved word (that is part of the code, like Option, for example).
- The dash, although legal, should be avoided because it may be confused with the minus sign. Instead of First-name use First_name or FirstName.

Select the appropriate scope for variables and constants.

- A variable (or constant) can be only be accessed from within areas of your VB project depending on the **scope** that you chose for the variable when it was declared. There are three levels of scope, **global**, **module level**, or **local**.
    - A **global** variable may be used in ALL procedures of a project. Usually **it is wise to avoid overusing global variables** even though they seem convenient. It is a conventional to begin a global variable's identifier with a lowercase g.

- A variable with **module level** scope can be used anywhere within the module (a form or a code module). That is it can be used in any number of procedures that happen to be located in the particular module. It is conventional to begin a module level variable's identifier with a lowercase m.
- A **local** variable can only be used within the specific procedure where it is declared.

- The scope of the variable determines its **visibility**. The visibility of a variable refers to whether or not a variable "can be seen" by a particular procedure.
- The scope of a variable is determined by where you place its Dim statement. It is conventional to place the Dim statement at the top of the procedure if its a local variable and in the general declarations section of a module if it has module level scope.

Modify the environment to require Option Explicit.

- Choose the menu option, Tools/Options..., and then checkmark the Require Variable Declaration under the Editor tab in order to have VB automatically place the statement **Option Explicit** in every module that you insert into every project.
- Making this change requires you to declare all named variables and constants using Dim statements. This will prevent a number of logical and other errors.

constants

- If you know that a number will be used within your program's algorithm many times and would not change like the freezing point of water (O degrees Celsius) or the number of units in a dozen (12) then you should treat that number as a **constant**. Constants cannot change once they are assigned within a program.
  - There are 2 types of constants actually. **Named constants** are ones that you define within your program. You create the name for a named constant and you assign it the proper value.
  - **Intrinsic constants** are built into Visual Basic. These constants can neither be changed or renamed. For example, the color constants (vbBlue, vbRed, etc.) are intrinsic ones.

vbNewLine
vbNullChar
vbNullString
vbTab
vbVerticalTab
vbBlack (0)
vbBlue (16711680)
vbCyan (16776960)

vbGreen (65280)

vbMagenta (16711935)

vbRed (225)

vbWhite (167772115)

vbYellow (65535)

vbFalse (0)

vbTrue (-1)

- Intrinsic constants use two-character prefixes to indicate that they are intrinsic as opposed to named constants. **Named constants** use the three-letter, lowercase character prefixes that represent data types as in the chart above however you should type the rest of the identifier with uppercase letters. For example, a named constant that stands for the PA sales tax rate might be named sngPA_TAX_RATE. Notice that the underscore character should be used to separate multiple words within the identifier for named constants.
- Controls are more complicated objects that have properties, that respond to user events, and that can use built-in methods. They are much more complicated than variables and constants. While a text box could have "12" in its Text property, this is not identical to creating a variable and storing the value 12 there.

Convert text input to numeric values using the Val function.

- When a user inputs alphanumeric characters into a text box on a form, he or she is actually placing the characters into the Text property of that text box. The Text property of a text box is treated as a character string. Even if the user is asked to enter a number such as the number of items purchased the entered digits are treated as one string. Unfortunately you probably would want to perform arithmetic on the value entered by the user but it is an error to perform arithmetic on a string.
- The Text property of the text box must be converted to a value which can be used in an arithmetic expression. Use the **Val function** to convert a string into a numerical value. The statement **intTotalItems = Val(txtItems)** would make the conversion. Remember that the Text property of the text box txtItems is the default property so txtItems.Text is unnecessary in the reference. The variable intTotalItems will contain an actual numerical value that can be added, subtracted, multiplied, etc. Note that if the Text property of txtItems cannot be turned into a valid number, the Val function may return a zero. Actually the Val function reads the Text property from left to right turning digits, decimal points, and positive or negative signs into numbers until it reaches an invalid character. Examples: Val("123.4#") = 123.4 Val("-13") = -13 Val(#903) = 0

Perform calculations using variables and constants.

- When performing calculations within an expression in your code you must take into account the **order of operations**. Many people learned "Please Excuse My Dear Aunt Sally" in elementary or junior high school. This reminds you to execute the following mathematical operations in this order: Parentheses, Exponentiation, Multiplication, Division, Addition, and Subtraction. However, multiplication and division should be evaluated from left to right within the expression as well as addition and subtraction. For example, 10 - 3 + 4 = 11 not 3.
- You should add parentheses if necessary in order to make sure that your expression evaluates correctly.
- **Assignment statements** evaluate from right to left. So you must evaluate the expression to the right of the **assignment operator** (the equal sign, =) and realize that the resulting value will be stored in the variable to the left of the assignment operator. However, you may assign a value to a label Caption property or a text box Text property.

Format values for output using the formatting functions.

- It is important to make sure that results of an expression are **formatted** nicely if the output is to be displayed to the user. Visual Basic provides a number of formatting functions that can be used to display values nicely.
- The **FormatCurrency function** returns a string of characters formatted as a monetary amount with a dollar sign, commas if necessary, and two positions to the right of a decimal point. Formerly with older version of BASIC this sometimes was much more difficult than using one simple function. For example, FormatCurrency(1345.236) = $1,345.24
- The **FormatNumber function** works similarly as the FormatCurrency function except that the dollar sign is not outputted. It is easy to prevent the numbers to the right of the decimal from being displayed by adding a second argument. For example, FormatNumber(1345.236, 0) = 1,345
- The **FormatPercent function** can be used to automatically multiply the argument by 100 and add a percent sign. For example, FormatPercent(.8241, 1) = 82.4% where the second argument indicates that one decimal place is to be displayed.
- The **FormatDateTime function** works along with predefined formats to display dates and/or times. For example, FormatDateTime("9/10/98", vbGeneralDate) = 9/10/98 5:23:12 PM Other named formats are vbLongDate, vbShortDate, vbLongTime, and vbShortTime.

If-else Blocks

The essence of computer programming relies on telling the computer what to do when something occurs, and how to do it. This is performed by setting conditions, examining them and stating

what decisions the computer should make. Microsoft Visual Basic uses various conditional statements for almost any situation your computer can encounter. As the application developer, it is up to you to anticipate these situations and make your program act accordingly.

If Block

The If...Then statement examines the truthfulness of an expression. Structurally, its formula is:

```
If (condition) Then
        statement(s)
[ElseIf (condition) Then
        statement(s)]
[Else
        statement(s)]
End If
```

Therefore, the program examines a condition, in this case ConditionToCheck. This ConditionToCheck can be a simple expression or a combination of expressions. If the ConditionToCheck is true, then the program will execute the Statement.

There are two ways you can use the If...Then statement. If the conditional formula is short enough, you can write it on one line, like this:

If ConditionToCheck Then Statement

If there are many statements to execute as a truthful result of the condition, you should write the statements on alternate lines. Of course, you can use this technique even if the condition you are examining is short. In this case, one very important rule to keep is to terminate the conditional statement with End If.

If ConditionToCheck Then

Statement

End If

Example:

If Condition Then Statement 1

Statement 2 Statement n

End If

Example:

Sub Main()

Dim age As Integer

Console.WriteLine("Please enter your age") age = Console.ReadLine().ToString()

If age = 15 Then

Console.WriteLine("You are not old enough to drink") End If

Console.ReadLine()

End Sub

First we created a variable called age with the data type integer. We then print to the console "Please enter your age" . Then we read the users value. If the user enters 15 they will see the message "You are not old enough to drink" and if they enter any other value your computer will explode. just kidding nothing will happen and the program will terminate.

This is a simple if Statement the problem with this if statement is that if condition evaluates to false (so age is not 15) the program simply won't do anything. In this case we can use an If Else statement.

ElseIf Clauses

The If. Then statement offers only one alternative: to act if the condition is true. Whenever you

would like to apply an alternate expression in case the condition is false, you can use the If...Then. Else statement. The syntax of this statement is:

If ConditionToCheck Then Statement1

Else

Statement2 End If

When this section of code is executed, if the ConditionToCheck is true, then the first statement, Statement 1, is executed. If the ConditionToCheck is false, the second statement, in this case Statement 2, is executed.

```
Dim message As String
message = "Welcome " & username
If username = "Asim" And password = 243 Then Console.WriteLine(message)
Else
If username = "Dave" And password = 12345 Then Console.WriteLine(message)
Else
If username = "Admin" And password = 2012 Then Console.WriteLine(message)
Else
Console.WriteLine("Oops could not find you!") End If
End If
End If
```

Nested If Statements

Example:

```
If tempInteger > 32 Then
        If tempInteger > 80 Then
                commentLabel.Text = "Hot"
        Else
                commentLabel.Text = "Moderate"
        End If
Else
        commentLabel.Text = "Freezing"
End If
```

Using If Statements with Radio Buttons & Check Boxes

- Instead of coding the CheckedChanged events, use If statements to see which are selected
- Place the If statement in the Click event for a Button, such as an OK or Apply button

```
Private Sub testButton_Click(ByVal sender As System.Object, _
 By Val e As System.EventArgs) Handles testButton.Click
 ' Test the value of the check box.
    If testCheckBox.Checked Then
        messageLabel.Text = "Check box is checked"
    End If
End Sub
```

Another example:

```
If maleRadioButton.Checked And _
 Integer.Parse(ageTextBox.Text) < 21 Then
    minorMaleCountInteger += 1
End If
If juniorRadioButton.Checked Or seniorRadioButton.Checked Then
    upperClassmanInteger += 1
End If
```

MESSAGE BOXES

Message boxes used to display the information. Message boxes should be used for short messages or to give transient feedback. It can hold 1024 characters. Message boxes can be used in either two ways, depending on your needs. You can use for display information and also for decision making from the user. In either case the MsgBox Function is used by the following syntax:

MsgBox(*prompt*[, *buttons*] [, *title*] [, *helpfile*, *context*])

The **MsgBox** function syntax has these parts:

Part	Description
prompt	Required. String expression displayed as the message in the dialog box.
buttons	Optional. Numeric 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 for ***buttons*** is 0 (which causes only an OK button to be displayed with no icon).
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
helpfileand ***context***	Both optional. These arguments are only applicable when a Help file has been set up to work with the application.

Eg.1

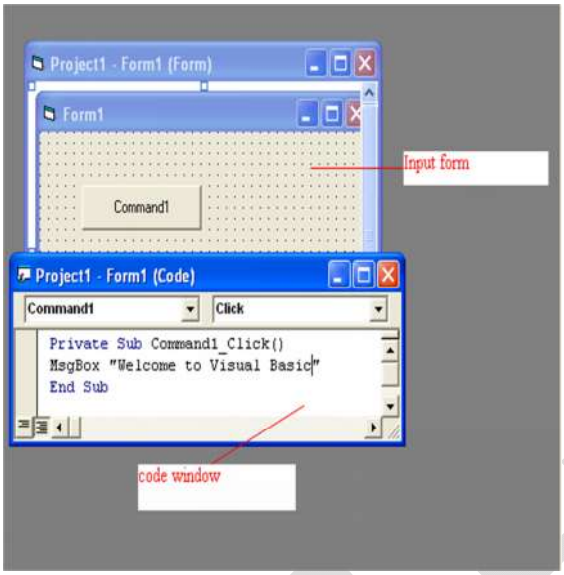


Figure 3.1 Input Form

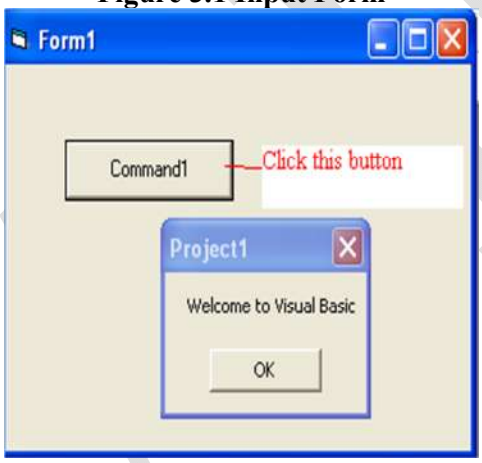


Figure3.1 Output Form

In the running screen click on the command1 button you get an Msgbox named "Welcome to Visual Basic" followed by **OK** button. Now, Click on the **OK** button you will be back to Visual Basic screen.

Eg.2

MsgBox "The Last Name field must not be blank.", _ VbExclamation, _ "Last Name"

➔causes the following box to be displayed:

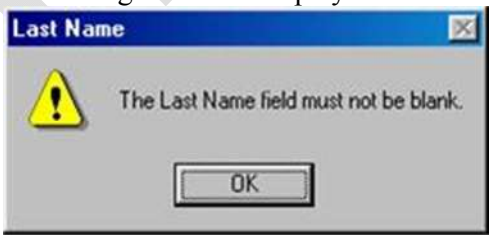


Figure 3.3

The buttons argument

The Buttons displayed in a message here

Button Layout	Value	Short Description
vbOKonly	0	Displays the OK button.
vbOKCancel	1	Displays the ok and cancel button.

vbAbortRetryIgnore	2	Displays the Abort , Retry , Ignore
vbYesNoCancel	3	Displays Yes , No and Cancel button
vbYesNo	4	Displays the Yes / No button
vbRetryCancel	5	Displays the retry and Cancel buttons.

The Icons displayed in the message box are here

Icon on message	Value	Short Description
vbCritical	16	Displays critical message icon
vbQuestion	32	Displays question icon
vbExclamation	48	Displays exclamation icon
vbInformation	64	Displays information icon

The Default button displayed in a message form

Default Button	Value	Short Description
vbDefaultButton1	0	Button 1 is default
vbDefaultButton2	256	Button 2 is default
vbDefaultButton3	512	Button 3 is default

Msgbox Return Value

Return Value	Value	Short Description
vbOk	1	The User Clicked OK
vbCancel	2	The User Clicked Cancel
vbAbort	3	The User Clicked Abort
vbRetry	4	The User Clicked Retry
vbIgnore	5	The User Clicked Ignore
VbYes	6	The User Clicked Yes
VbNo	7	The User Clicked No

Input Validation

- Check to see if valid values were entered by user before beginning calculations
- Check for a range of values (reasonableness)
  - If Integer.Parse(hoursTextBox.Text) > 10 Then
    - MessageBox.Show("Too many hours")
- Check for a required field (not blank)
  - If nameTextBox.Text <> "" Then ... Performing Multiple Validations
- Use nested If statement to validate multiple values on a form
  - Use Case structure to validate multiple values
  - Simpler and clearer than nested If
  - No limit to number of statements that follow a Case statement
  - When using a relational operator must use the word **Is**

- Use the word **To** to indicate a range of constants
- Sharing an Event Procedure
- Add events to the Handles clause at the top of an event procedure
- Allows the procedure to respond to events of other controls
- Key to using a shared event procedure is the *sender* argument
- Cast (convert) *sender* to a specific object type using the CType function

SUBJECT : Programming with visual basic

UNIT-2

S.No	Questions	opt1	opt2	opt3	opt4	Answer
1	VB uses _____ blocks such as variables, data types etc…,	constructing	building	fundamental	specified	building
2	Code in VB is stored in the form of _____	modules	subroutines	variables	standards	modules
3	There are _____ kinds of modules	5	2	3	4	3
4	A simple application may contain a single form and the code resides in the _____ itself	form	module	form module	none	form module
5	_____ is the foundation of object oriented programming in VB	standard	class	form	none	class
6	___ is the extension for class module	.frm	.CLS	.std	none	.CLS
7						
8	_____ may include constant,type,variable and DLL procedure declarations.	declaration	procedure	variable	statement	declaration
9	By default VB variables are of the _____ data type.	const	static	variant	none	variant
10	_____ is the value range of byte	0 to 255	1 to 255	0 to 266	1 to 266	0 to 255
11	_____ is the value range of integer	-32767 to 32768	-32768 to 32767	32767 to -32768	32768 to -32767	-32768 to 32767
12	_____ are used for storing values temporarily.	character	constant	variable	module	variable
13	From the following which character is allowed while declaring variables	%	@	_	$	_
14						
15	There are _____ ways for declaring variables	1	3	2	4	3
16						
17	_____ statement checks in the module for usege for any undeclared variables.	option explicit	option implicit	int count	none	option explicit

18						
19	_____ variable is one that is declared inside a procedure	global	local	static	scope	local
20	Variables that are declared with keyword ___ exist only as long as the procedure is being executed	public	private	dim	double	dim
21	To make all variables in a procedure static, _____ keyword is placed at the beginning of the procedure.	public	private	dim	static	static
22	The module-level variable is available to all the procedures in the module. They are declared using _____ keywords	public	private	a and b	a or b	a or b
23	VB programs can be broken into smaller logical components called _____	procedures	programs	sub	event	procedures
24	Event procedures acquire the declaration _____ by default	public	private	static	global	private
25	_____ window is opened for the module to which the procedures is to be added.	code	add	type	sub	code
26	Functioons are especially useful for taking one or more pieces of data called	modules	arguments	procedures	programs	arguments
27	_____ statements are used to control the flow of program execution	control	property	while	do-while	control
28	_____ block is used to define several blocks of statements, in order to execute one block	if then ….. Else	if	then	if … then	if then ….. Else
29	_____ block is used for conditional execuion of one or ore statements	if then … else	if	then	if … then	if … then
30	_____ is an alternative to If…Then….Else.	select…case	case…select	select	none	select…case
31						
32	Select…case structure evaluates an expression ___ at the top of the structure	twice	once	thrice	none	once
33						
34	The _____ statement first executex the statemetn and then test the condition after each execution	do….while	while….do	select….case	while	do….while
35	_____ structure executes the statements until the condition is satisfied	do…loop	do..loop until	do while…loop	none	do..loop until

36	do…loop until is --------- loop	finite	infinite	long	small	infinite
37	_____ function retrives only date	for…next	next…for	exit for	exit do	for…next
38	A _____ loop can be terminated by an exit for statement	for…next	next…for	exit for	for exit	for…next
39	do….while loop is terminated using _____ statement	exit for	for exit	exit do	do exit	exit do
40	A sequence of variables by the same name can be referred using _____	arrays	modules	sub-routines	none	arrays
41						
42	The individual element of an array are identified using ___	modules	index	base	none	index
43	Fixed size array always _____	remains same	changes	. increased	decreased	remains same
44	Size of dynamic array is changed at _____ time	execution	run	break	stop	run
45	Variables of different datatypes when combined as a single variable to hold several related information is called a _____ datatype	user defined	user like	user string	none	user defined
46	_____ statement is used to create a constant.	constant	const	consta	constan	const
47	The _____ function retrieves the date and time	Now	Date	Time	none	Now
48	_____ function retrives only time	Now	Date	Time	none	Time
49	_____ function retrives only date	Now	Date	Time	none	Date
50	_____ function returns the intervals between 2 dates in terms of years, months or dates.	format	diffdate	datediff	relational	datediff
51	_____ function accepts a numeric value and converts it to a string in format	specified	relational	logical	comparison	specified
52						
53	_____ returns the variant (string) containing a specified number of characters from a string	left()	right()	mid()	instr()	mid()

54	The _____ function returns the length of the string	strcmp()	strrev()	len()	format()	len()
55						
56	_____ is a group of controls that share the same name and type	control arrays	arrays	format arrays	none	control arrays
57	There are _____ ways to create a control array at design time	3	2	1	5	3
58						
59	_____ is the value of the index in the array	TRUE	FALSE	0	index%	index%
60	Control arrays are added at run time using _____ statement.	load	unload	format	unformat	load
61	_____ statement removes a control from an array	load	unload	format	unformat	unload
62	_____ is a named sequence of statements executed as a unit	property	procedure	project	browser	procedure
63	A procedure name is always defined at _____ level	routine	subroutine	procedure	project	procedure
64						
65	_____ is the list box at the upper-right corner of the code window and the debug window that displaya the procedures recognized for the object displayed in the object box.	project file	property page	procedure box	none	procedure box
66	VB offers different types of _____ to execute small sections of coding in applications	project	property	procedure	none	procedure
67	Procedures used in one program can act as a _____- for other programs with slight modifications	building blocks	buildings	construction	module	building blocks
68	A _____ procedure is a procedure block that contains the control's actual name, an under score (_) and avent name.	sub	event	general	function	event
69	_____ procedure dialog box from the tools menu and by choosing the required scope and type	add	sub	mul	div	add
70	Boolean has _____ function	cbool	cbol	cboolean	cboolea	cbool
71	Decimal has _____ function	cdecimal	cdeci	cdemal	cdec	cdec

Unit III -MODULAR PROGRAMMING

Menus, sub-procedures and sub-functions defining/creating and modifying a menu, using common dialog box, creating a new subprocedure, passing variables to procedures, passing argument by value or by reference, writing a function/ procedure.

Introduction

Menu bar is the standard feature of most windows applications. The main purpose of the menus is for easy navigation and control of an application. Some of the most common menu items are File, Edit, View, Tools, Help and more. Each item on the main menu bar also provide a list of options or in the form of a pull-down menu. When you create a Visual Basic program, you need not include as many menu items as a full fledge Windows application such as Microsoft Words. What you need is to include those menu items that can improve the ease of using your program by the user, and not to confuse the user with unnecessary items. Adding menu bar is relatively easy to accomplish in Visual Basic. There are two ways to add menus to your application, one way is to use the Visual Basic's Application Wizard and the other way is to use the menu editor.

Designing Menus Using Menu Editor

The Menu Editor lets you quickly and easily place menu bar items into your application by pushing command buttons and typing a few property values. The Menu Editor contains menu description tools that let you create the application's menu bar, menu commands, and shortcut access keys. The Menu Editor is a dialog box that you access from the Form window by pressing Ctrl+E or by selecting Tools | Menu Editor from Visual Basic's own menu bar.

To Display the Menu Editor

Press Ctrl+E to display the Menu Editor. Each menu bar command requires a caption (specified by the Caption property) and a name (specified by the Name property). The other Menu Editor items are optional. The additional Menu Editor properties, such as the Enabled property that determines whether the menu item is grayed out and unavailable for certain procedures, as well as a Visible property, which determines when the user can see the menu bar command, are not needed for every option. You'll rarely change these extra property values from their default values for menu bar commands.

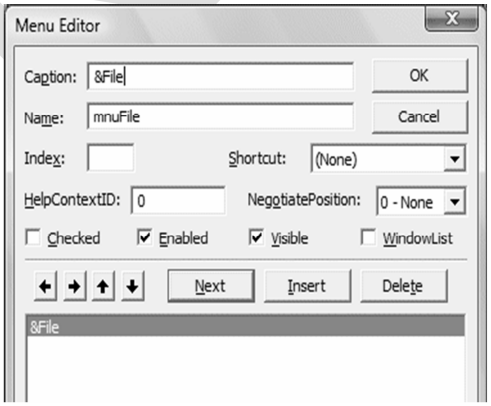
Using the List Box in the Menu Editor

In the list box at the bottom of the Menu Editor, all top-level menu items are flush with the left side of the list box border. Sub-level menu items have four small dots preceding the menu caption; sub– sub-menu items have eight dots. The menu items can also be reorganized according to position in the menu. To control the level and position of the menu item being entered, just use the four direction arrows in the Menu Editor. The up and down arrows reposition menu items for order. The left and right arrows allow menu items to be top-level, sub-level or sub– sub-level. You can implement a separator bar in a menu (a horizontal line that visually defines the boundary between two groups of menu items) by putting a single dash (-) as the entire caption for a menu item. This item then becomes the separator bar on the runtime menu.

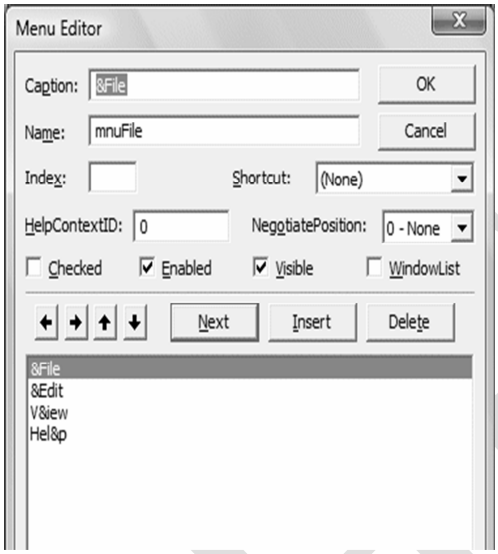
To Create Menu Controls in the Menu Editor

To start adding menu items to your application, open an existing project or start a new project, then click on Tools in the menu bar of the Visual Basic IDE and select Menu Editor. When you click on the Menu Editor, the Menu Editor dialog will appear. In the Menu Editor dialog, key in the first item File in the caption text box. You can use the ampersand (&) sign in front of F so that F will be underlined when it appears in the menu, and F will become the hot key to initiate the action under this item by pressing the Alt key and the letter F. After typing &File in the Caption text box, move to the name textbox to enter the name for this menu item, you can type in mnuFile here. Now, click the Next button and the menu item &File will move into the empty space below.

Add Menu Controls in Menu Editor

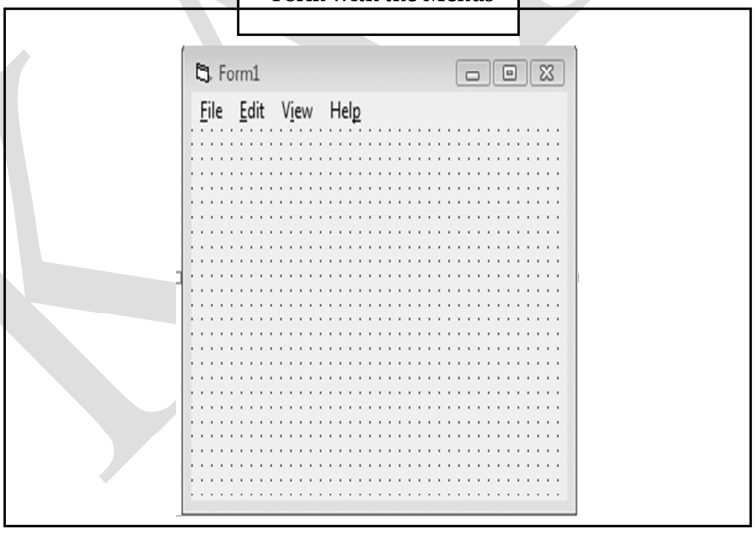


The Menu Editor

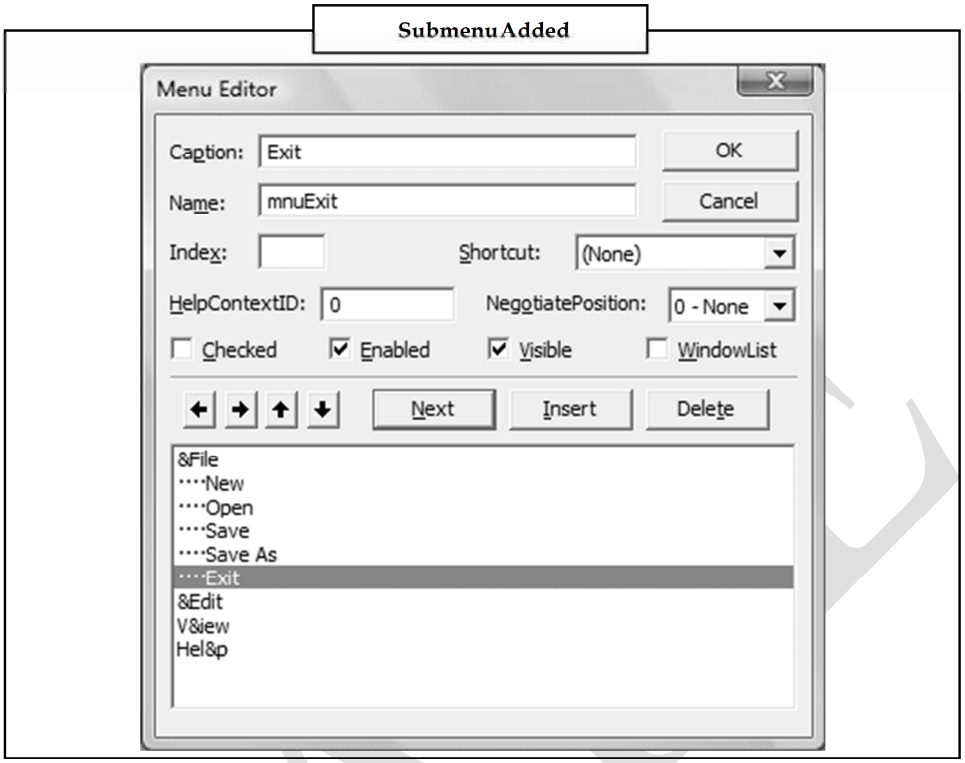


You can then add in other menu items on the menu bar by following the same procedure. When you click Ok, the menu items will be shown on the menu bar of the form.

Form with the Menus

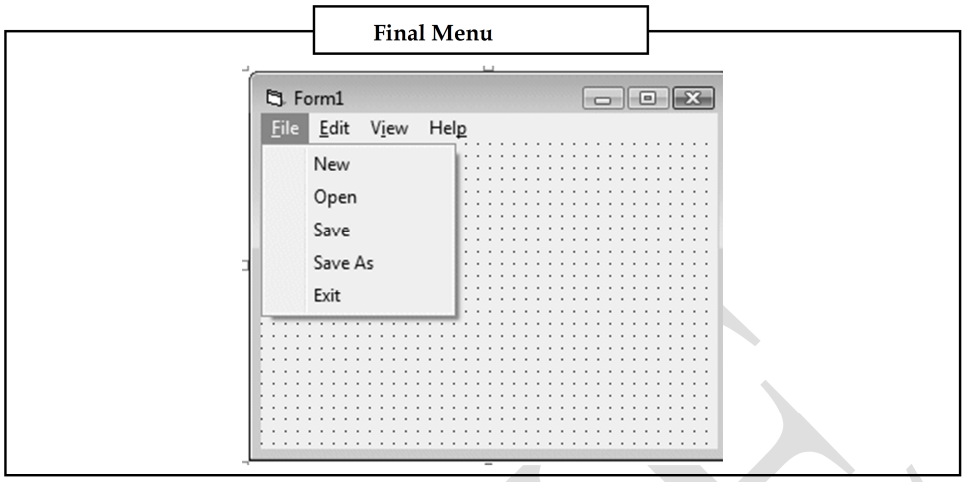


Now, you may proceed to add the sub menus. In the Menu Editor, click on the Insert button



between File and Exit and then click the right arrow key, and the dotted line will appear. This shows the second level of the menu, or the submenu. Now, key in the caption and the name. Repeat the same procedure to add other submenu items. Here, we are adding New, Open, Save, Save As and Exit.

Now, click the OK button and go back to your form. You can see the dropped down submenus when you click on the item File, as shown.



Finally, you can enter the code by clicking on any of the submenu items.

Assigning Access Keys and Shortcut Keys

To improve keyboard access to menu commands we make use of access keys and shortcut keys.

Access Keys

You can use an access key instead of the mouse to move the focus to a menu or command. To do so, follow the steps below:

1. Build the Visual Basic form for your application, using Windows standards for menu items, buttons and other standard controls. Windows standards are implemented in every Microsoft application so you have to look no further than Microsoft Visual Basic itself for most of them. Any underlined letter in the text of a menu item, button or other control identifies the access key for that control.
2. Determine the controls on your Visual Basic form for which you want to assign access keys. Decide which key will be your access key for each control. Of course, each control must have a different key for your application to determine which control your access key combination is to invoke.
3. Click on a control such as the "File" menu item to select it. Scroll down in the Properties window to find the Text property for the control. Insert an ampersand (&) before the character that you want to be the access key. For example, the Text property of the File menu item would be "&File" to make Alt-f the access key combination.
4. Click the Debug menu item at the top of the Visual Basic screen to test your access

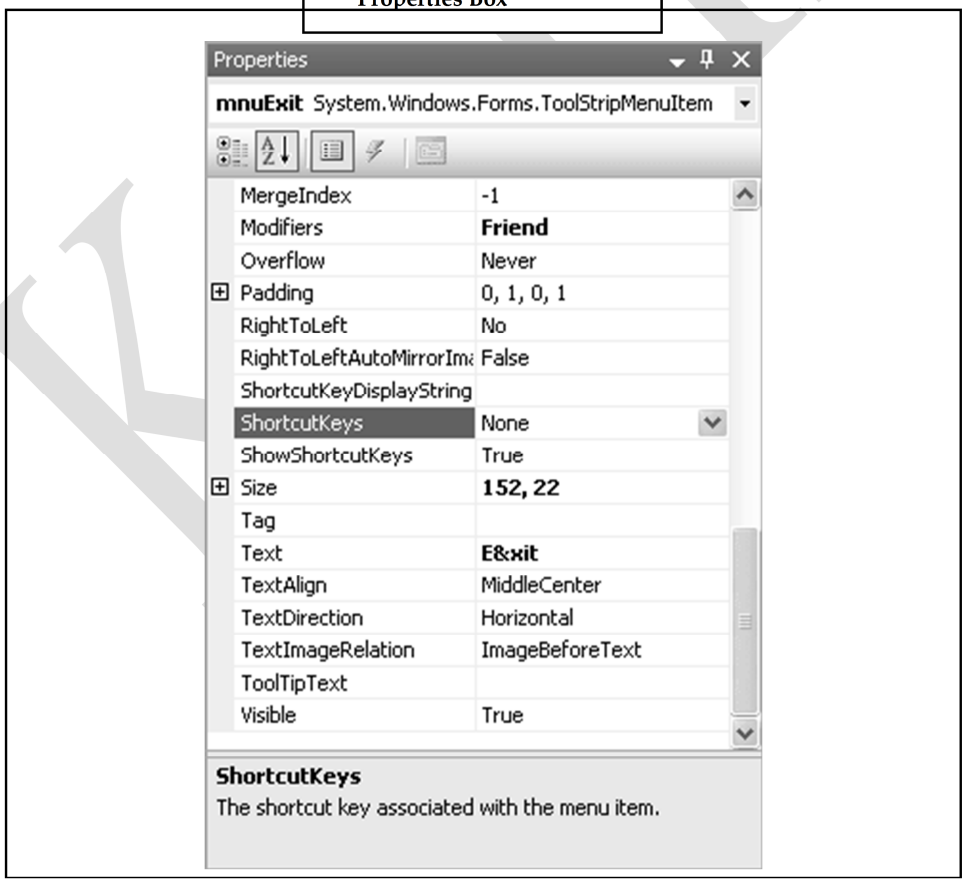
keys. When your form appears press the Alt key to see an underline under the access key in the File menu item. Hold down the Alt key and press "f" to activate the File menu item. This indicates that you have successfully set the access key.

Shortcut Keys

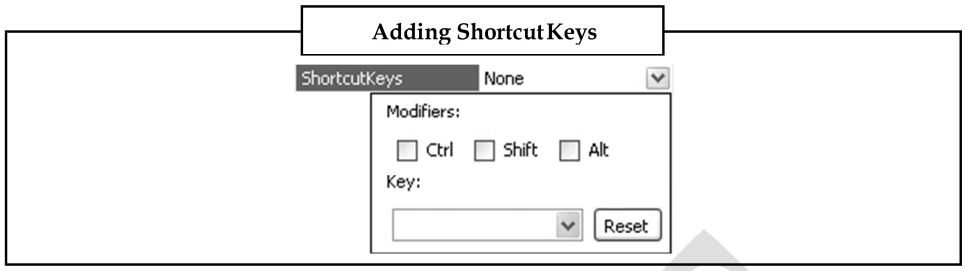
Short-cuts are the key combinations you press to active a menu item when the menu does not have focus. The three most commonly used short-cuts are Ctrl-C, Ctrl-X, and Ctrl-V. These are typically bound to Copy, Cut, and Paste, but you can bind them to any of your menu items if you choose. If you want to set the short-cut key for a menu item, you have the options listed in the Shortcut: list. To do so, follow the steps below:

- In Design time, select the Exit item on your menu
- Look at the properties box on the right
- Locate the ShortcutKeys item:

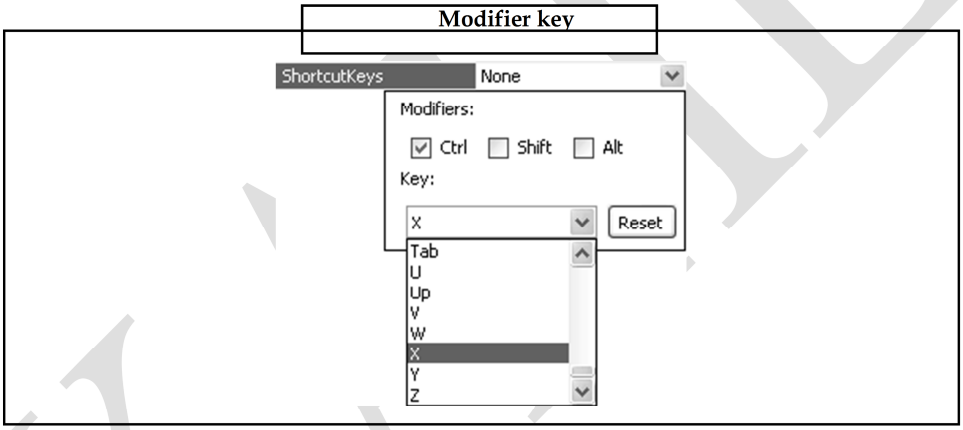
Properties Box



- Click the down arrow to reveal the following:



The Modifier is the key you press with your shortcut. For example, the CTRL key then the "X" key on your keyboard. Place a check inside the Ctrl box. Then select the letter "X" from the Key dropdown list, as in the next image:



Click back on your menu to see what it looks like:

Run your program and test out the shortcut. Don't click the File menu. Just hold down the Ctrl key on your keyboard. Then press the letter X. Again, the program will close down.

Separating Menu Items

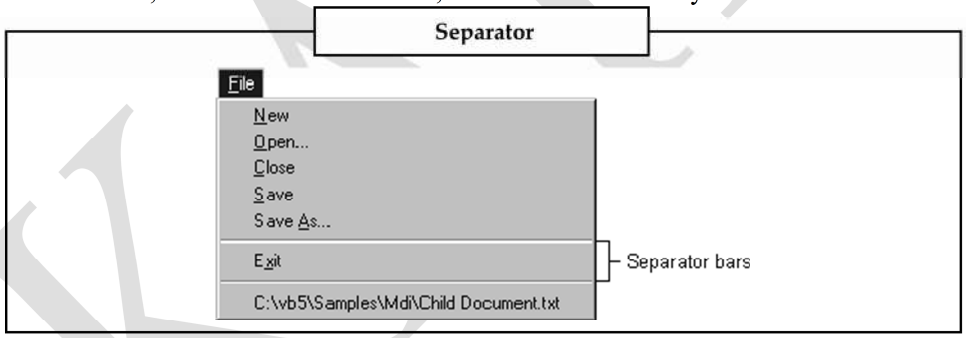
A separator bar is a horizontal line between items on a menu. You can use a hyphen (-) as the Caption property for a menu control to create a separator bar. On menus that contain multiple items, you can use separator bars to divide items into logical groups.

To Create a Separator Bar in the Menu Editor

To create a separator bar in the Menu Editor:

1. Click Insert tab
2. Click the right arrow button to indent the new menu item to the same level as the menu items it will separate.
3. In the Caption text box. , enter a hyphen (-)
4. Set the Name property.
5. Click OK

Although separator bars are created as menu items, they do not respond to the Click event, and users cannot choose them. A menu item cannot be a separator bar if it is a menu title, has submenu items, is checked or disabled, or has a shortcut key.



Common Dialog Boxes

- You can use set of predefined standard dialog boxes in your projects for such tasks as specifying colors and fonts, printing, opening and saving.
- The common dialog boxes control, which is a custom control, allows your project to use the dialog boxes that are provided as a part of the Windows environment.
- To use the common dialog control, you first place a control, and its location doesn't matter, you cannot change size of the control, since it will be invisible when your program will run.
- In code when you wish to display one of the standard dialog boxes, you refer the

properties and methods of the control.

- You don't need more than one dialog control on your form.

Steps to add Common Dialog Control

1. Select Project -> Components in Visual Basic IDE

Visual Basic displays a window that contains the list of ActiveX controls that are installed in your machine.

2. Make sure Control tab is selected.

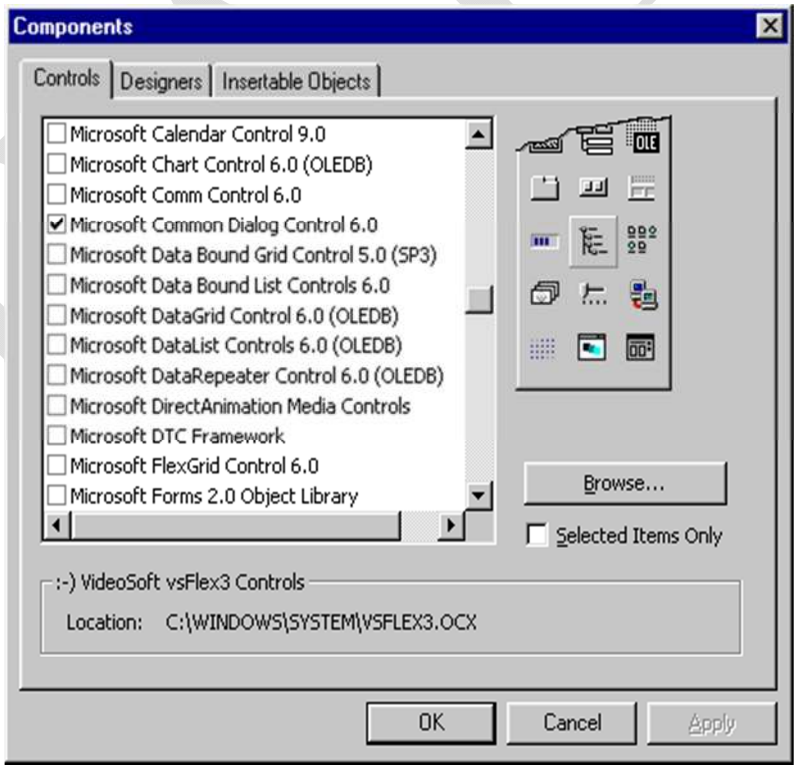
3. Go to the control that you want to load. In this case we want to load Microsoft Common Dialog Control 6.0, so select that control.

4. Turn on the checkbox on the left of the control.

At this stage, the name and path of the selected .OCX file will be displayed at the bottom of Component dialog.

5. Click on Ok button.

Once an ActiveX control is loaded, one or more icons are displayed depending upon whether the selected component contains one or more controls.



What does Common dialog control have?

Microsoft common dialog control allows you to access a set of dialog boxes through its methods and properties.

The following are the dialog boxes that can be accessed through Common dialog control.

Dialog box	Meaning	Method to Invoke
Open	Displays the list of files and directories and allows you to select a file	. ShowOpen
Save As	Displays the list of files and directories and allows you to select a file	. ShowSave
Color	Displays a list of standard colors and System colors	. ShowColor
Font	Displays a dialog box with the list of available fonts and styles	. ShowFont
Print	Displays print dialog box	. ShowPrinter
Help	Invokes help window using the given file	. ShowHelp

Dialog boxes that can be accessed through common dialog control.

Now, let us understand properties related to each of these common dialog boxes.

Open and Save As dialog boxes

- Open dialog box is used to let user select a file. It contains list of files, directories and also a text box into which you can enter filename.
- Save As dialog box is same as Open dialog box except the caption. Both the dialog boxes display list of files, directories and drives so that user can select a file from anywhere from the file system.

The following are the properties that are related to Open and Save As dialog boxes.

Property	Meaning
CancelError	Specifies whether a trappable runtime error (with number 32755) occurs when user selected Cancel button or no error occurs.
DefaultExt	Specifies the default extension such as .txt or .frm for files. When a file with no extension is entered the default extension is automatically added.
DialogTitle	Contains the title of the dialog box. This is ignored for Font Color and Print dialog boxes.
FileTitle	Returns only the filename without path.
Filter	Specifies the filters to be displayed in Type list box of dialog.
FilterIndex	Specifies the default filter.
InitDir	Specifies the directory to be used initially. If not specified then current directory is used.
FileName	Returns the path and filename of the selected file.

Properties that are specific to Open and Save As dialog boxes.

The following code listing shows you how to use Open dialog box.

```
‘ Open button displays Open dialog box and takes the filename
Private Sub CmdOpen_Click()
With CommonDialog1
‘ set two filters; one is *.txt another is *.*
.Filter = "Text Files|*.txt|All files|*.*"
.FilterIndex = 0 ' first filter is default
.DialogTitle = "Select a text file"
‘ invoke Open dialog box
.ShowOpen
' print filename on the form
Print .FileName
End With
End sub
```

Font Dialog box

- Font dialog box displays the list of available font names and other font styles. This dialog box can display the fonts supported by either Printer or Screen or Both.
- You must specify whether you want to list fonts supported by Screen or Printer or Both. Unless you specify this, no fonts will be displayed in Font dialog box.

The following are the properties related to Font dialog box.

Property	Meaning
FontBold	Whether bold was selected
FontItalic	Whether italic was selected.
FontStrikethru	Whether strikethrough was selected.
FontUnderline	Whether underline was selected.
FontName	The selected font name
FontSize	The selected font size.

Properties that are specific to Font dialog box.

Flags property for Font Dialog box

This property is used to set options related to Font dialog box. You can set more than one flag using OR operator. The following is the list of flags related to Font dialog box.

Flag Constant	Value	Meaning

CdlCFBoth	&H3	Causes fonts supported by Screen and Printer to be listed.
CdlCFEffects	&H100	Causes the dialog box to display Underline and Strikethrough effects also.
CdlCFPrintersFonts	&H2	Causes only fonts supported by printer to be listed.
CdlCFScreenFonts	&H1	Causes only fonts supported by screen to be listed.

Flags related to Font dialog box

The following listing illustrates how to use Font dialog box to allow user to select Font related options and set those options to current form.

```
Private Sub cmdFont_Click()
With CommonDialog1
'Display screen fonts with effects
.Flags = cdlCFScreenFonts Or cdlCFEffects
.ShowFont
Me.FontName = .FontName
Me.FontSize = .FontSize
Me.FontBold = .FontBold
Me.FontItalic = .FontItalic
Me.FontUnderline = .FontUnderline
Me.FontStrikethru = .FontStrikethru
Print "This is sample text"
End With
End Sub
```

Color Dialog box

- Color dialog box displays color palette from which user can select a color.
- Color dialog box also allows users to create a custom color and select it.
- Color property returns the selected color.
- Flags property can be used to specify whether user should be allowed to define custom colors (cdlCGPreventFullOpen) and whether full dialog box (including Define custom colors section) is to be displayed.

The following code displays Color dialog box and changes the background of the form with the color selected by user. Color dialog box with Define custom color section open.

```
Private Sub cmdBKColor_Click()
With CommonDialog1
.Flags = cdlCCFullOpen
.ShowColor
Me.BackColor = .Color
End With
End Sub
```

Print Dialog Box

The Print dialog box allows the user to specify how output should be printed. The user can specify a range of pages to be printed, print quality, number of copies to be printed, and so on. The following are the properties related to Print dialog box.

Property	Meaning
Copies	Specifies the number of copies to be printed.
FromPage	Specifies the value to be displayed in FromPage text box Print dialog box.
ToPage	Specifies the value to be displayed in ToPage text box Print dialog box.
Min	Specifies the minimum value for page range.
Max	Specifies the maximum value for page range.
PrinterDefault	Specifies whether the options selected in Print dialog box change the settings of Printer system object.

Properties of Print dialog box.

The following code allows user to select printing related options and also changes the corresponding settings in Printer system object.

```
Private Sub CmdPrint_Click()
With CommonDialog1
' enable changes to printer object
.PrinterDefault = True
' you must set Min and Max properties otherwise
' Frompage and ToPage are ignored.
.Min = 1
.Max = 10
.FromPage = 5
.ToPage = 10
```

```
' select Pages as the default
.Flags = cdlPDPageNums
.ShowPrinter
' handle printing using Printer Object
End With
End Sub
```

Help Dialog Box

Unlike other dialog boxes, it doesn't take any input from user. Instead it runs WINHLP32.EXE program to display help file that is set using HelpFile property.

The following are the properties related to Help dialog box.

Property	Meaning
Helpfile	Specifies the help file to be used to display help.
Helpkey	Specifies the keyword of the topic that is to be displayed.
Helpcommand	Specifies the type of online help requested.

Properties related to Help dialog box of Common dialog control

The following listing displays the help file CDPLAYER.HLP that is found in Windows\Help folder.

```
Private Sub Cmdhelp_Click()
With CommonDialog1
.FileName = "c:\windows\help\cdplayer.hlp"
' display contents of the file initially
.HelpCommand = cdlHelpContents
.ShowHelp
End With
End Sub
```

Procedures in VB

A *procedure* (including an event procedure) is a self-contained group of Visual Basic commands that can be accessed from a remote location within a Visual Basic program. The procedure then carries out some specific action. Information can be freely transferred between the "calling" location (i.e., the command which accesses the procedure) and the

procedure itself. Thus, it is possible to transfer information to a procedure, process that information within the procedure, and then transfer a result back to the calling location. Note, however, that not all procedures require an information transfer – some merely carry out an action without any information interchange. Large modules are customarily decomposed into multiple procedures, for several reasons. First, the use of procedures eliminates redundancy (that is, the repeated programming of the same group of instructions at different places within program). Secondly, it enhances the clarity of a program by allowing the program to be broken down into relatively small, logically concise components. And finally, the use of independent procedures allows programmers to develop their own libraries of frequently used routines.

Visual Basic supports three types of procedures – *Sub* procedures (sometimes referred to simply as *subroutines*), *Function* procedures (also called *functions*), and *Property* procedures. Sub and function procedures are commonly used in beginning and intermediate level programs. The *shell* (beginning and ending statements) for a new sub or function procedure can be added to a project by selecting Add Procedure... from the Tools menu.

SUB PROCEDURES (SUBROUTINES)

In its simplest form, a sub procedure is written as

Sub *procedure name* (*arguments*)

. . . .

statements

. . . . .

End Sub

The *procedure name* must follow the same naming convention used with variables. In addition, a procedure name cannot be identical to a constant or variable name within the same module.

The list of *arguments* is optional. Arguments represent information that is transferred into the procedure from the calling statement. Each argument is written as a variable declaration; i.e.,

argument name As *data type*

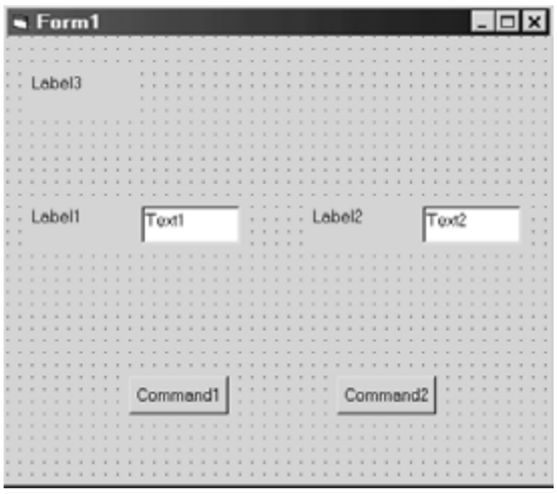
The data type can be omitted if the argument is a variant. Multiple arguments must be separated by commas. If arguments are not present, an empty pair of parentheses must appear in the Sub statement.

EXAMPLE

Here is a sub procedure that determines the smallest of two numbers.

```
Sub Smallest(a, b)
Dim Min
If (a < b) Then
Min = a
MsgBox "a is smaller (a = " & Str(Min) & ")"
ElseIf (a > b) Then
Min = b
MsgBox "b is smaller (b = " & Str(Min) & ")"
Else
Min = a
MsgBox "Both values are equal (a, b = " & Str(Min) & ")"
End If
End Sub
```

This procedure has two arguments, a and b. Both are variants. The procedure compares the values of the arguments, determines which is smaller, and then displays the value of the smaller argument in a message box. Note that the variable Min is a variant that is defined locally within the procedure. It represents the smallest value among the arguments. This variable is not required in this example (we could simply use a or b instead). However, it is a good idea to include this variable, in case the procedure should be expanded to process the minimum value in some manner without altering the given values of the arguments.



Also, note that we could also have included explicit data typing in the first two lines; i.e.,

```
Sub Smallest(a As Variant, b As Variant)
Dim Min As Variant
```

or, if we choose a different data type,

```
Sub Smallest(a As Single, b As Single)
Dim Min As Single
```

etc., if we wished.

Calling the sub procedures

A sub procedure can be accessed from elsewhere within the module via the Call statement. The Call statement is written

Call *procedure name* (*arguments*)

The list of arguments in the Call statement must agree with the argument list in the procedure definition.

The arguments must agree in *number*, in *order*, and in *data type*. However, the respective names may be different. Thus, if the procedure definition includes three arguments whose data types are single, integer, and string, the Call statement must also contain three arguments whose data types are single, integer, and string, respectively. The names of the arguments within the procedure definition need not, however, be the same as the names of the arguments in the Call statement. For example, the arguments within the procedure definition might be named a, b and c, whereas the corresponding arguments within the Call statement might be called x, y and z. Here is another way to access a sub procedure.

procedure name arguments

Note the absence of the keyword Call, and the absence of parentheses. When the procedure is accessed, the values of the arguments within the calling portion of the program become available to the arguments within the procedure itself. Thus, the values of the arguments are transferred from the calling portion of the program to the procedure. Moreover, if the value of an argument is altered within the procedure, the change will be recognized within the calling portion of the program. (Actually, it is the *addresses* of the arguments that are shared; hence, the *contents* of those addresses can be accessed from either the calling portion of the program or from within the procedure itself.) This type of transfer is called passing by *reference*.

EXAMPLE

The program determines the smallest of two numbers and then displays the result. Figure shows the preliminary control layout.

The required procedures (a sub procedure and two event procedures) are shown below.

```
Sub Smallest(a, b)
Dim Min
If (a < b) Then
Min = a
MsgBox "a is smaller (a = " & Str(Min) & ")"
ElseIf (a > b) Then
Min = b
MsgBox "b is smaller (b = " & Str(Min) & ")"
Else
Min = a
MsgBox "Both values are equal (a, b = " & Str(Min) &
")"
End If
End Sub
Private Sub Command1_Click()
Dim x As Variant, y As Variant
x = Val(Text1.Text)
y = Val(Text2.Text)
Call Smallest(x, y)
‘or:
‘Smallest x, y
End Sub
Private Sub Command2_Click()
End
End Sub
```

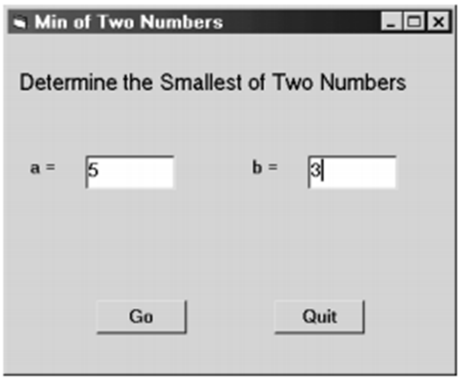
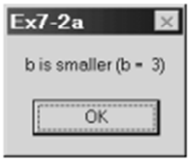


Fig. 7.3



When the user clicks on the Go button, event procedure Command1_Click() is activated. This causes the two values entered in the text boxes to be converted to numerical values and assigned to the variants x and y, respectively. These values are then transferred to the sub procedure Smallest when the sub procedure is accessed via the Call statement. Within Smallest, the arguments (i.e., the values of x and y within the event procedure) are referred to

as a and b. The sub procedure then determines which argument represents the smallest value and displays an appropriate message indicating the result.

Passing Value by reference and by value

When passing an argument by reference, the argument name may be preceded by the reserved word ByRef within the procedure definition; i.e.,

ByRef *argument name* As *data type*

The ByRef designation is not essential, however, because this is the default mode of transfer in Visual Basic. An argument passed by reference is usually written as a single variable within the calling statement. It may be possible, however, to write an argument as an *expression* within the calling statement and still pass its value to a procedure by reference (most programming languages do not allow expressions to be passed by reference). This works because the expression is assigned its own address, which is accessible from within the procedure.

Note, however, that information cannot be transferred back to the calling portion of the program when the calling argument is written as an expression (see below).

Arguments can also be passed to a procedure by *value*. In this case, the value assigned to each argument in the calling statement (rather than the argument's address) is passed directly to the corresponding argument within the procedure. This is strictly a one-way transfer; that is, the argument values are transferred *from* the calling statement *to* the procedure. If any of these values is altered within the procedure, the new value will *not* be transferred back to the calling statement. Passing arguments by value can be useful, however, since the arguments in the calling statement can always be written as expressions rather than single variables. In order to pass an argument by value, the argument name within the procedure must be preceded by the reserved word ByVal; i.e.,

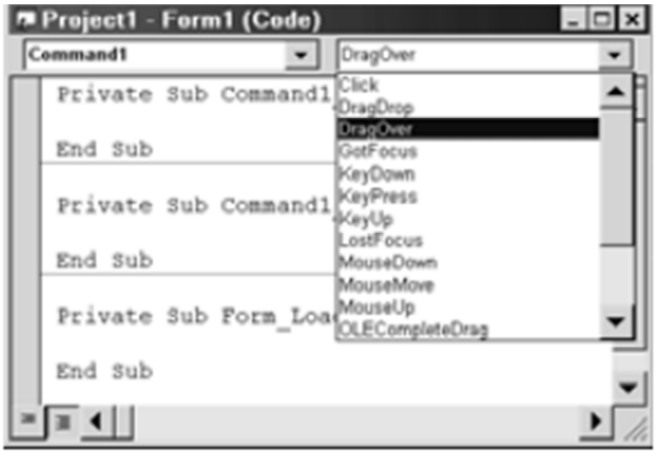
ByVal *argument name* As *data type*

If a procedure includes multiple arguments, some may be passed by reference and others by value.

```
Sub Smallest(ByVal a, ByVal b, ByRef c)
If (a < b) Then
c = a
```

```
Else
c = b
End If
End Sub
Private Sub Command1_Click()
Dim x, y, z, min
x = Val(Text1.Text)
y = Val(Text2.Text)
z = Val(Text3.Text)
Call Smallest(x, y, min)
Call Smallest(z, min, min)
Text4.Text = Str(min)
End Sub
Private Sub Command2_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
End Sub
Private Sub Command3_Click()
End
End Sub
```

EVENT PROCEDURES

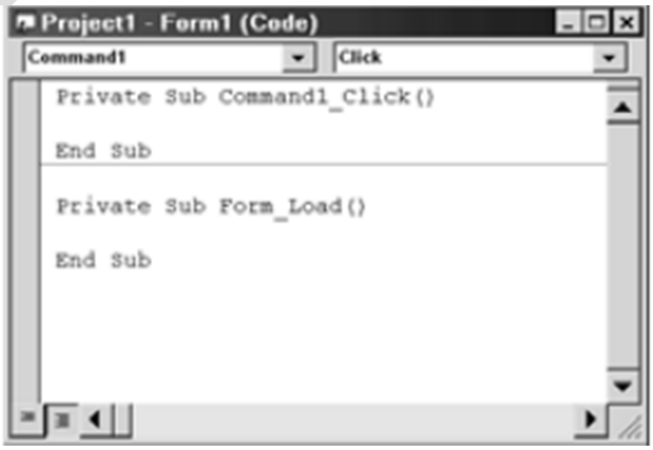


An event procedure is a special type of sub procedure. It is accessed by some specific action, such as clicking on an object, rather than by the Call statement or by referring to the procedure name. The particular action associated with each event procedure is selected from the upper-right drop-down menu within the Code Editor Window. The object name and the activating event collectively make up the event procedure name. Thus, Command1_Click() is the name of an event procedure that is activated by clicking on command button Command1.

Like any other sub procedure, arguments may be used to transfer information into an event procedure. An empty pair of parentheses must follow the procedure name if arguments are not present.

EXAMPLE- DEFINING AN EVENT PROCEDURE

Suppose we double click on command button Command1 within the Form Design Window, the Code Editor Window will then be displayed. The object in this case is Command1 and the desired action is a mouse click, as indicated by the two menu selections at the top. If a different action is desired, it can be selected by clicking on the down arrow in the upper right window and then selecting from the resulting menu.



Once the object and the action have been selected, the first and last lines of the event procedure are generated automatically within the Code Editor Window, as shown in above figures. The user must then provide the remaining Visual Basic statements, thus completing the event procedure. The term Private appearing in the first line determines

the *scope* of the event procedure; i.e., the portion of the program in which the event procedure is recognized.

FUNCTION PROCEDURES

A function procedure is similar to a sub procedure, with one important difference: a function is intended to return a single data item, just as a library function returns a single data item. Each function name therefore represents a data item, and has a data type associated with it. Within a function definition, the function name must be assigned the value to be returned, as though the function name were an ordinary variable. In its simplest form, a function procedure is written as

Function *procedure name* (*arguments*) As *data type*

. . . . .

statements

. . . . .

procedure name = ...

. . . . .

End Function

As with a sub procedure, the list of *arguments* is optional. Arguments represent information that is transferred into the procedure from the calling statement. Each argument is written as a variable declaration; i.e.,

argument name As *data type*

Remember that the data type can be omitted if the argument is a variant. Multiple arguments must be separated by commas. If arguments are not present, an empty pair of parentheses must appear in the Function statement. The *data type* designation in the Function statement refers to the data item being returned. This designation is not essential – the returned data item will be considered to be a variant if the designation is not included. Notice that the procedure name is assigned a value at some point within the procedure (multiple assignments are permitted, in accordance with the required program logic). This is the value being returned by the function. Thus, within a function, the procedure name is used as though it were an ordinary variable. (Contrast this with a sub procedure, where the procedure name does *not* represent a data item.)

EXAMPLE - DEFINING A FUNCTION PROCEDURE

Here is a function procedure that determines the factorial of a positive integer quantity. The function is based upon logic similar to that given in Example 4.11.

```
Function Factorial(n As Integer) As Long
Dim i As Integer
If n < 1 Then
Beep
MsgBox ("ERROR - Please try again")
Else
Factorial = 1
For i = 1 To n
Factorial = Factorial * i
Next i
End If
End Function
```

This procedure has one integer argument, n, which represents the value whose factorial will be determined. Thus, the value of n is transferred into the procedure, and its factorial is returned as a long integer. Note that the factorial is referred to by the function name, Factorial. Notice also that the function name (Factorial) is assigned a value at two different places within the procedure, as required by the program logic. A function procedure is accessed in the same manner as a library function, by writing the function name and its required arguments as an expression. Thus, the function name (and its arguments) can be assigned to another variable, etc. The list of arguments in the function access must agree with the argument list in the function definition in *number*, in *order* and in *data type*. As with sub procedures, however, the names of the arguments in the function access may be different than the argument names used in the function definition.

EXAMPLE

Let us now consider a complete Visual Basic program that determines the factorial of a positive integer n.

```
Function Factorial(n As Integer) As Long
Dim i As Integer
```

```
If n < 1 Then
Beep
MsgBox ("ERROR - Please try again")
Else
Factorial = 1
For i = 1 To n
Factorial = Factorial * i
Next i
End If
End Function

Private Sub Command1_Click()
Dim n As Integer, nFact As Long
n = Val(Text1.Text)
nFact = Factorial(n)
Text2.Text = Str(nFact)
End Sub

Private Sub Command2_Click()
Text1.Text = ""
Text2.Text = ""
End Sub

Private Sub Command3_Click()
End
End Sub
```

Note the manner in which the function procedure Factorial is accessed within event procedure Command1_Click;

i.e.,

```
nFact = Factorial(n)
```

Thus, the value of n is transferred into Factorial as an argument. The factorial of n is then

returned by the function and assigned to the long integer variable nFact. The value of nFact is then converted to a string and displayed within text box Text2. In the above code, two separate statements are used to access Factorial and to display its returned value; i.e.,

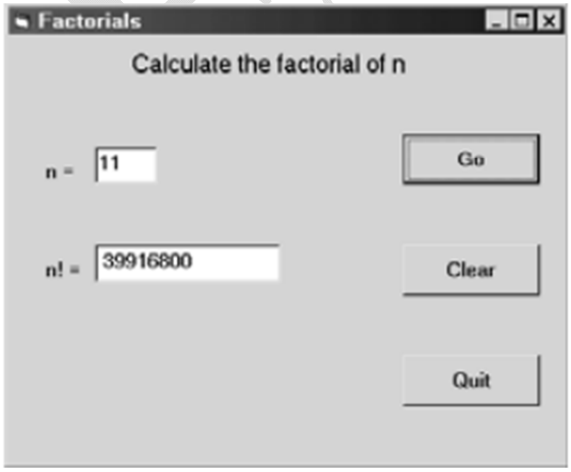
nFact = Factorial(n)

Text2.Text = Str(nFact)

This was done in order to clarify the program logic. The two statements can be combined, however, by simply writing

Text2.Text = Str(Factorial(n))

The result obtained with a representative value of n = 11 is shown in Figure below



A function reference may appear within a more complex expression, as though the function name were an ordinary variable. However, any required arguments must follow the function name, enclosed in parentheses and separated by commas.

Possible Questions

2 marks

1. Differentiate sub procedure and event procedure.
2. What is a menu?
3. List the flags and their meaning for font dialog box
4. Define a function
5. How do you separate menus?
6. What is a short cut key?
7. What is an access key?
8. What are the two ways of passing parameters in VB?
9. Write the general syntax for an event procedure.
10. List the color common dialog box properties.

6-marks

1. Illustrate the steps involved in creating a menu with example.
2. How to assign access key and short cut keys to a menu item? Explain with example.
3. Describe the various types of procedures In VB
4. Write a note on common dialog box control and its uses.
5. Write a program in VB to calculate factorial of a number using functions
6. Elucidate the open and print dialog box with examples
7. Explain two ways of passing the parameters to procedures in VB with examples.

Sno	Questions	opt1	opt2	opt3
1	Which is a type of procedure found in VB?	Event	Function	Sub
2	When using a procedure the calling code sends data via the:	actual argument to the formal parameter of the procedure.	formal argument to the actual parameter of the procedure.	actual parameter to the formal argument of the procedure.
3	Which is an optional element of an event procedure?	End Sub	Handles	Object_Event
4	What happens when a parameter in a procedure is declared ByVal?	Only arguments of numeric data types are allowed.	A reference to the argument is sent to the procedure.	A copy of the argument is sent to the procedure.
5	Which is not an optional element of a sub procedure declaration?	Parameters	Public	Private
6	How many return statements are allowed in a Function Procedure?	0	1	2
7	Which menu object property places a check mark in the display of the menu text?	Check	Checked	CheckMark
8	Which symbol creates an access key in the text of a menu item?	@	#	$
9	Which is not a standard dialog box?	ColorDialog	FontDialog	OpenDialog
10	How are dialog boxes implemented in a program?	A dialog box is generated programmatically.	A dialog control is placed in the component tray.	A dialog control is placed on the form.
11	What is the method used to activate the color dialog box?	ActivateDialog	DisplayDialog	ExhibitDialog
12	Which OpenFileDialog control property specifies the choices in the “Files of type” dropdown box?	FileName	FileNames	FileType
13	The function procedures are _____ by default.	Public	Private	Protected
14	Every optional argument in the procedure definition must specify a _____ value which must be a constant expression.	Constant	Default	Integer

15	To add the commondialog control to any project one has to include it from	File menu->component->Microsoft Common Dialog Control 6.0	Project menu->Component-> Microsoft Common Dialog Control 6.0	Component menu->project-> Microsoft Common Dialog Control 6.0
16 17	A _____ can be used to add new commands to the existing menus, create new menus and menu bars, change or delete existing menus and menu bars	Tool editor	Menu interface	Menu editor
18 19	A menu editor can be added oly after _____ a project.	creating	opening	closing
20 21	The _____ includes all the menu controls of the current form.	Tool editor	Menu interface	Menu editor
22	A _____ bar is a line which seperates the menu item.	join	include	divide
23	Pop-up menus are also called _____ menus, because the items displayed on the pop-up menu depend on where the pointer is located when the right mouse button is clicked.	menu controls	check marks	context menus
24	When the menu item is ___ it is dimmed and cannot be selected, but viewed.	able	disable	view
25	To place a check mark in a menu item, the checked property is set to _____	TRUE	FALSE	0
26	_____ is a named sequence of statements executed as a unit	property	procedure	project
27	A procedure name is always defined at _____ level	routine	subroutine	procedure
28 29	_____ is the list box at the upper-right corner of the code window and the debug window that displaya the procedures recognized for the object displayed in the	project file	property page	procedure box
30	VB offers different types of _____ to execute small sections of coding in applications	project	property	procedure
31	Procedures used in one program can act as a _____ for other programs with slight modifications	building blocks	buildings	construction
32	A _____ procedure is a procedure block that contains the control's actual name, an under score (_) and avent name.	sub	event	general
33	A function can also be written by selecting _____ procedure dialog box from the tools menu and by choosing the required scope and type	add	sub	mul

34	The _____ statement first executex the statemetn and then test the condition after each execution	do….while	while….do	select….case
35	_____ structure executes the statements until the condition is satisfied	do…loop	do..loop until	do while…loop
36	do…loop until is --------- loop	finite	infinite	long
37	_____ function retrives only date	for…next	next…for	exit for
38	A _____ loop can be terminated by an exit for statement	for…next	next…for	exit for
39	do….while loop is terminated using _____ statement	exit for	for exit	exit do
40	Do...Loop is an iterative statement because it:	selects a block of statements to run.	runs the same block of statements repeatedly.	selects a block of statements and runs it repeatedly.
41	Which Do…Loop statement should be used to process test scores where a test score over 100 is a signal to stop the processing?	Do While Score > 100	Do Until Score > 100	Loop While Score > 100
42	In the For…Next statement the default value for the Step is:	-1	0	1
43	The For…Next Loop is used when:	a choice is made based on a Boolean condition.	a block of statements is executed an unknown number of times.	a block of statements is executed a known number of times.
44	Which is not a valid Exit statement?	Exit Do	Exit For	Exit Form

opt4	Answer
All of the above.	All of the above.
formal parameter to the actual argument of the procedure.	actual argument to the formal parameter of the procedure.
Statements	Statements
Both a and b.	A copy of the argument is sent to the procedure.
Sub	Sub
There is no limit.	There is no limit.
CheckOn	Checked
&	&
ZoomDialog	ZoomDialog
All of the above	A dialog control is placed on the form.
ShowColor	ShowColor
Filter	Filter
Inherited	Public
Character	Default

None of the above	Project menu->Component-> Microsoft Common Dialog Control 6.0
none	Menu editor
maximizing	opening
none	Menu editor
seperator	seperator
none	context menus
visible	disable
1	TRUE
browser	procedure
project	procedure
none	procedure box
none	procedure
module	building blocks
function	event
div	add

while	do….while
none	do..loop until
small	infinite
exit do	for…next
for exit	for…next
do exit	exit do
selects a block of statements and runs it a specified number of times.	runs the same block of statements repeatedly.
Loop Until Score > 100	Do Until Score > 100
2	1
None	a block of statements is executed a known number of times.
Exit Select	Exit Form

LOOPING WITH For-Next

The For-Next structure is a block of statements that is used to carry out a looping operation; that is, to execute a sequence of statements some predetermined number of times. The structure begins with a For-To statement and ends with a Next statement. In between are the statements to be executed.

In its simplest form, a For-Next structure is written as

For *index* = *value1* To *value2*

. . . . . .

executable statements

. . . . . .

Next *index*

The For-To statement specifies the number of passes through the loop. Within this statement, *index* is a variable whose value begins with *value1*, increases by 1 each time the loop is executed, until it reaches *value2*.

Note that the value of *index* will be *value2* during the last pass through the loop.

The Next statement identifies the end of the loop. It consists simply of the keyword Next, followed by the *index*. The *index* appearing in the For-To and the Next statements must be the same. (Visual Basic allows the *index* to be omitted from t he Next statement in single For-Next loops, though this is considered poor programming practice.)

The *executable statements* refer to one or more consecutive statements that are executed during each pass through the loop. These statements are usually indented, so that the structure can easily be identified. The indentation is not required, though it is considered good programming practice.

EXAMPLE

A typical For-To loop structure is shown below.

```
sum = 0
For i = 1 To 10
sum = sum + i
Next i
```

This structure will result in 10 passes through the loop. During the first pass, i will be assigned a value of 1; i will then increase by 1 during each successive pass through the loop, until it has reached its final value of 10 in the last pass. Within each pass, the current value of i is added to sum. Hence, the net effect of this program segment is to determine the sum of the first 10 integers (i.e., 1 + 2 + . . . + 10).

Note the indentation of the assignment statement within the loop structure.

A more general form of the For-Next structure can be written as

For *index* = *value1* To *value2* Step *value3*

. . . . . .

executable statements

. . . . . .

Next *index*

Within the For-To statement, *value3* determines the amount by which *value1* changes from one pass to the next. This quantity need not be restricted to an integer, and it can be either positive or negative. If *value3* is negative, then *value1* must be greater than *value2* (because the value assigned to *index* will *decrease* during each successive pass through the loop). Note that *value3* is understood to equal 1 if it is not shown explicitly (i.e., if the Step clause is omitted).

EXAMPLE

```
sum = 0
For count = 2.5 To -1 STEP -0.5
sum = sum + count
```

Next count

The loop structure will cause count to take on the values 2.5, 2.0, 1.5, . . ., 0.0, -0.5, -1.0. Hence, the final value of sum will be 6.0 (because 2.5 + 2.0 + 1.5 + 1.0 + 0.5 + 0.0 – 0.5 – 1.0 = 6.0). Note that this structure will generate a total of eight passes through the loop.

The For-Next structure is one of the most widely used features in Visual Basic. It is most often used when the number of passes through the loop is known in advance.

The following rules apply to For-Next loops.

1. The index variable can appear within a statement inside the loop, but its value cannot be altered.
2. If *value1* and *value2* are equal and *value3* is nonzero, the loop will be executed once.
3. The loop will not be executed at all under any of the following conditions:
   (*a*) *value1* and *value2* are equal, and *value3* is zero.
   (*b*) *value1* is greater than *value2*, and *value3* is positive.
   (*c*) *value1* is less than *value2*, and *value3* is negative.
4. Control can be transferred *out* of a loop, but *not in*.

Visual Basic includes an Exit For statement. This statement permits a transfer out of a For-Next loop if some particular condition is satisfied. For example, we may wish to jump out of a loop if an error or a stopping condition is detected during the execution of the loop.

The Exit For statement is generally embedded in an If-Then structure that is included within the loop. When the Exit For statement is encountered during program execution, control is immediately transferred out of the For-Next loop, to the first executable statement following Next.

EXAMPLE

Here is a variation of Example 3.14, illustrating the use of a typical Exit For statement.

```
sum = 0
For i = 1 To 10
sum = sum + i
If sum >= 10 Then
Exit For
Next i
```

This loop is set up to execute 10 times, but the execution will be terminated if the current value of sum equals or exceeds 10. In this particular case, the execution will terminate within the fourth pass (because 1 + 2 + 3 + 4 = 10).

LOOPING WITH Do-Loop

In addition to For-Next structures, Visual Basic also includes Do-Loop structures, which are convenient when the number of passes through a loop is not known in advance (as, for example, when a loop is required to continue until some logical condition has been satisfied).

A Do-Loop structure always begins with a Do statement and ends with a Loop statement. However, there are four different ways to write a Do-Loop structure. Two of the forms require that a logical expression appear in the Do statement (i.e., at the beginning of the block); the other two forms require that the logical expression appear in the Loop statement (at the end of the block).

The general forms of the Do-Loop structure are shown below.

First form:	***Second form:***
Do While *logical expression* . . . . . . *executable statements* . . . . . . Loop	Do Until l*ogical expression* . . . . . . *executable statements* . . . . . . Loop

Third form:	***Fourth form:***
Do . . . . . . *executable statements* . . . . . . Loop While *logical expression*	Do . . . . . . *executable statements* . . . . . . Loop Until *logical expression*

The first form continues to loop as long as the *logical expression* is true, whereas the second form continues to loop as long as the *logical expression* is *not* true (until the *logical expression* becomes true).

Similarly, the third form continues to loop as long as the *logical expression* is true, whereas the fourth form continues to loop as long as the *logical expression* is *not* true.

Note that there is a fundamental difference between the first two forms and the last two forms of the Do Loop block. In the first two forms, the logical test is made at the *beginning* of each pass through the loop; hence, it is possible that there will not be *any* passes made through the loop, if the indicated logical condition is not satisfied. In the last two forms, however, the logical test is not made until the *end* of each pass; therefore, at least one pass through the loop will always be carried out.

EXAMPLE

Consider the following two Do-Loop structures.

flag = "False" . . . . . . Do While flag = "True" . . . . . . Loop	flag = "False" . . . . . . Do . . . . . . Loop While flag = "True"

The left loop will not execute at all, because the logical test at the beginning of the loop structure is false. The right loop will execute once, however, because the logical test is not carried out until the end of the first pass through the loop. Moreover, if the string "True" is assigned to flag during this first pass through the right loop, then the execution will continue indefinitely, until flag is reassigned.

Note that a Do-Loop structure does not involve a formal index. Thus, the programmer must provide the logic for altering the value of the *logical expression* within the loop. Typically, an initial assignment is made before entering the loop structure. The logical expression is then altered at some point within the loop.

EXAMPLE

Here is a Do-While loop that is comparable to the For-Next loop

```
sum = 0
count = 1
Do While count <= 10
sum = sum + count
```

```
count = count + 1
Loop
```

This structure will result in 10 passes through the loop. Note that count is assigned a value of 1 before entering the loop. The value of count is then incremented by 1 during each pass through the loop. Once the value of count exceeds 10, the execution will cease. Here is another way to accomplish the same thing.

```
sum = 0
count = 1
Do
sum = sum + count
count = count + 1
Loop While count <= 10
```

If we choose to use an Until clause rather than a While clause, we can write the control structure in either of the following ways.

sum = 0 count = 1 Do Until count > 10 sum = sum + count count = count + 1 Loop	sum = 0 count = 1 Do sum = sum + count count = count + 1 Loop Until count > 10

Note that the logical expression in these two structures (count > 10) is the opposite of the logical expression in the first two structures (count <= 10).

Control can be transferred out of a Do-Loop block using the Exit Do statement. This statement is analogous to Exit For, which is used with For-Next blocks. Thus, when an Exit Do statement is encountered during program execution, control is transferred out of the Do-Loop block to the first executable statement following Loop.

EXAMPLE

Here is a variation of Example 3.16, illustrating the use of a typical Exit Do statement.

```
sum = 0
count = 1
Do While count <= 10
sum = sum + count
If sum >= 10 Then
Exit Do
count = count + 1
Loop
```

LOOPING WITH While-Wend

Visual Basic supports While-Wend structures in addition to Do-Loop structures. This structure also permits conditional looping. The structure begins with the While statement (analogous to Do While), and ends with the Wend statement (analogous to Loop).

The general form of a While-Wend structure is

```
While logical expression
. . . . . .
executable statements
. . . . . .
Wend
```

The loop created by the While-Wend structure continues to execute as long as the *logical expression* is true.

Thus, While-Wend is analogous to a Do While-Loop structure. Note that the logical expression is tested at the *beginning* of each pass through the loop. The While-Wend structure, like the Do-While structure, does not involve a formal index. Therefore, you must assign an initial value to the *logical expression* before entering the loop. This value will then be altered within the loop, in accordance with the program logic.

EXAMPLE

Here is a While-Wend loop that is comparable to the Do-While loop

```
sum = 0
count = 1
While count <= 10
sum = sum + count
count = count + 1
Wend
```

THE Stop **STATEMENT**

The Stop statement is used to terminate the execution at any point in the program. The statement consists simply of the keyword Stop. This statement may appear anywhere in a Visual Basic program except at the very end. Multiple Stop statements may appear in the same program, as dictated by the program logic. However, modern programming practice tends to avoid the use of the Stop statement.

The _____ statement first executex the statemetn and then test the condition after each execution	do….while	while….do	select….case	while		do….while
_____ structure executes the statements until the condition is satisfied	do…loop	do..loop until	do while…loop	none		do..loop until
do…loop until is --------- loop	finite	infinite	long	small		infinite
_____ function retrives only date	for…next	next…for	exit for	exit do		for…next
A _____ loop can be terminated by an exit for statement	for…next	next…for	exit for	for exit		for…next
do….while loop is terminated using _____ statement	exit for	for exit	exit do	do exit		exit do
Do...Loop is an iterative statement because it:	selects a block of statements to run.	runs the same block of statements repeatedly.	selects a block of statements and runs it repeatedly.	selects a block of statements and runs it a specified number of times.		runs the same block of statements repeatedly.
Which Do…Loop statement should be used to process test scores where a test score over 100 is a signal to stop the processing?	Do While Score > 100	Do Until Score > 100	Loop While Score > 100	Loop Until Score > 100		Do Until Score > 100
In the For…Next statement the default value for the Step is:	-1	0	1	2		1
The For…Next Loop is used when:	a choice is made based on a Boolean condition.	a block of statements is executed an unknown number of times.	a block of statements is executed a known number of times.	None		a block of statements is executed a known number of times.
Which is not a valid Exit statement?	Exit Do	Exit For	Exit Form	Exit Select		Exit Form
_____ statements are used to control the flow of program execution	control	property	while	do-while		control
_____ structure executes the statements until the condition is satisfied	do…loop	do..loop until	do while…loop	none		do..loop until
do…loop until is --------- loop	finite	infinite	long	small		infinite
_____ function retrives only date	for…next	next…for	exit for	exit do		for…next
A _____ loop can be terminated by an exit for statement	for…next	next…for	exit for	for exit		for…next
do….while loop is terminated using _____ statement	exit for	for exit	exit do	do exit		exit do
A sequence of variables by the same name can be referred using _____	arrays	modules	sub-routines	none		arrays

The individual element of an array are identified using ___	modules	index	base	none		index
Fixed size array always _____	remains same	changes	. increased	decreased		remains same
Size of dynamic array is changed at _____ time	execution	run	break	stop		run
Variables of different datatypes when combined as a single variable to hold several related information is called a _____ datatype	user defined	user like	user string	none		user defined
_____ statement is used to create a constant.	constant	const	consta	constan		const
The _____ function retrieves the date and time	Now	Date	Time	none		Now
_____ function retrives only time	Now	Date	Time	none		Time
_____ function retrives only date	Now	Date	Time	none		Date
_____ function returns the intervals between 2 dates in terms of years, months or dates.	format	diffdate	datediff	relational		datediff
_____ function accepts a numeric value and converts it to a string in format	specified	relational	logical	compariso n		specified
_____ returns the variant (string) containing a specified number of characters from a string	left()	right()	mid()	instr()		mid()
The _____ function returns the length of the string	strcmp()	strrev()	len()	format()		len()
_____ is a group of controls that share the same name and type	control arrays	arrays	format arrays	none		control arrays
There are _____ ways to create a control array at design time	3	2	1	5		3
_____ is the value of the index in the array	TRUE	FALSE	0	index%		index%
Control arrays are added at run time using _____ statement.	load	unload	format	unformat		load
_____ statement removes a control from an array	load	unload	format	unformat		unload
_____ is used for opening many windows at the same time	MDI	IDM	DIM	SDI		MDI
All the document windows are contained in a _____	child	parent	both	mdi		parent
VB application can have only _____ MDI form.	1	2	3	4		1
Each time the user clicks new from the file menu, a new _____ window is created and displayed.	parent	form	standard	child		child

____ removes the form from the memory as well as from the screen.	unload	hide	remove	delete
___ removes the form from screen only but not from the memory	unload	hide	remove	delete

_____ is a group of controls that share the same name and type	control arrays	arrays	format arrays	index	control arrays
____ binds data-aware controls to Microsoft access and other ODBC data sources.	DAO	ADO	Data control	Data	Data control
UDT refers to	User defined datatypes	User declared data type	Undefined data types	Undecided data types	User defined datatypes
Which property of the list box makes multiple selection possible?	Addmoreitem	Mulselect	Multiple sel	Select	Multiple sel
_____ communicates with data sources through the JET database engine	Data Access Objects	data control	cointainer	relations	Data Access Objects
___- binds data-aware controls to microsoft access and other ODBC data sources.	DAO	ADO	Data control	Data	Data control
_____ provides a framework for using code to create and manipulate components of a remote ODBC DataBase system	RDO	RDC	ODBC	VBSQL	RDO
_____ binds the control of a ODBC remote DataBase .	RDO	RDC	ODBC	VBSQL	RDC
_____ is an API call interface to the open DataBase connectivity libraries and drivers to provide data access to Oracle	RDO	RDC	ODBC	VBSQL	ODBC
_____ is an implementation of the DataBase library API specifically designed to provide access to an SQL server through a VB application.	RDO	RDC	ODBC	VBSQL	VBSQL
_____ is a programming model that eliminates the need to choose from among DAO and RDO	ADO	RDC	ODBC	DB	ADO
A data access object is a collection of _____ classes that model the structure of a relational database system.	module	procedure	object	programe	object
The _____ object corresponds to a stored table definition.	DataBase	TableDef	QueryDef	Record set	TableDef
The _____ is a stored query definition , which is a pre compiled SQL statement.	DataBase	TableDef	QueryDef	Record set	QueryDef
There are _____ types of DAO libraries supported by Visual Base 6.0.	1	2	3	4	2
To open an existing database , the _____ method of the workspace object is used.	open database	create database	database	none	open database
A _____ is an object that contains a set of records from database.	set	records	record set	none	record set
The _____ method moves to the first row in the record set.	move first	move next	move previous	move last	move first
The _____method moves to the last row in the record set.	move first	move next	move previous	move last	move next
The _____ method moves to the previous row in the record set.	move first	move next	move previous	move last	move previous
The _____ method moves to the last row in the record set.	move first	move next	move previous	move last	move last
The _____ property is True when the user moves beyond the last record in the record set.	EOF	BOF	Both	none	EOF
The _____- property is True when the user has moved to a position before the first record in the record set.	EOF	BOF	Both	none	BOF
Modifying and Deleting records are used to _____ a record in a record set.	manipulate	edit	change	modify	manipulate
The _____ method can be used to locate a record in a table type record set.	open	close	store	seek	seek
The _____ method is used to perform action queries.	open	close	store	execute	execute
The _____- command is used to add rows to a table.	add	insert	sub	delete	insert
_____ supports page and report headers , detail lines and many other common features , including a variety of graphics and font features.	Data view window	Data report designer	Query designer	Data environment designer	Data report designer
The _____ designer enables us to design queries save them in our DataBase.	Dataview window	Data report	Query	Data Environment	Query
With the help of _____ Designer, we can link it to all our databases, tables and Queries with a single object.	Dataview window	Data report	Query	Data Environment	Data Environment
The _____ can actually write real ADO code that is designed to browse records in the same way as the old data contsl could.	Data form wizard	Data view window	Query	Data Environment	Data form wizard
Syntax to declare a multidimensional Array	Dim var_name(element's row,element's column)	Dim var_name(element's row,element's column);	Dim var_name(element's row,element's column)	Dim var_name [element's row][element's columns]	Dim var_name(element's row,element's column)
What is the value of the index for the first element in a VB.NET array?	0	1	2	3	0
Which method will return the number of elements in an array?	Dimension	Length	Number	Size	
In the statement, Dim Days(7) as String, what part of the array does the number 7 refer to?	Array name	Datatype	Lowerbound	Upperbound	Upperbound
What is required to reference an element in an array?	Array name	Index value of the element	Element value	Both array name and index	Both array name and index
Which method will arrange the elements of an array in alphabetical order?	Arrange	Assemble	Order	Sort	Sort
The number of variables allowed in an array is:	0	1	2	Any number of variables can be declared in an array.	Any number of variables can be declared in an array.
what is the number of elements in the following array: arr(6,6)	36	66	25	12	36
The array is resized using ___ statement.	Dim	Redef	Redim	Declare	Redim
User defined data types are similar to ____	structures	unions	functions	procedures	structures