

UNIT-I

TWO MARKS

1. Write any four properties of command button. (Apr'16, Nov'15, Nov'13)

- a. Name: Name of the command button
- b. Caption: The text displayed on the form
- c. Visible: It determines whether the command button displayed on the screen or not.
- d. Enabled: Determines whether the button is responsive or not.

2. List any four standard controls in VB.

Label, TextBox, CommandButton, CheckBox, OptionButton

3. Write any four properties used to control the appearance of a form.

Form BackColor, Height, Width, FillColor

4. State the different data types in VB. (Nov'15, Nov'12)

Byte, integer, long, single, double, currency, variant

5. List the various controls available in VB "Tool Box"

Label, TextBox, CommandButton, CheckBox, OptionButton, PictureBox, ComboBox, ListBox, ImageBox etc.

6. Define FORM.

A form is a container in visual basic IDE which is used to place various controls based on the application requirements. Also it is a blank window in visual basic contains the grid of dots.

7. What is a variable?

A variable is a name given to a memory location where a data value will be stored. The value of a variable may be changed at any time during program execution.

8. Give the purpose of a command button

The command button control is used to begin, interrupt, or end a process. When clicked, it invokes a command that has been written into its Click event procedure. Most Visual Basic applications have command buttons that allow the user to simply click them to perform actions.

9. What are labels? (Apr'12)

A label is a graphical control, which is used to display text. It is commonly used to display help information. It is also used to display information such as status messages and other program information.

10. List down common form properties.

Caption, name, borderstyle, enabled, font, height, width, left, top, mousepointer, visible, windowstate

11. List the names of more general event procedures

Click, Gotfocus, LostFocus, MouseMove, MousUp, MouseDown, KeyPress, KeyDown, KeyUp

12. List the scale properties in a form.

ScaleMode, ScaleHeight, ScaleWidth, ScaleLeft, ScaleTop

13. What are the properties for labels?

Labels have 34 properties. Commonly used properties are caption, wordwrap, autosize, backcolor, alignment etc.

14. Write a note on Variant data type in VB

Variant data type allows the user to store any type of data and is the default type of Visual Basic data type. If a variable is declared as variant, then it will have the ability to store any type of values.

15. What is IDE?

An integrated development environment (IDE), also known as integrated design environment is a software suite that consolidates the basic tools developers needs to write and test software. Typically, an IDE contains a code editor, a compiler or interpreter and a debugger.

16. Write a note on properties window.

Properties window displays the list of design time properties for the selected objects and their current settings. A property is a characteristic of an object such as name, size caption.

17. What is the toolbox?(Apr'12, Apr'11)

A toolbox contains a set of tools that are used at design time to place controls on a form. It contains more than 20 controls.

18. How to declare a variable in VB?

A variable can be declared with the help of "Dim" statement.

Syntax: Dim <variable_name> as <datatype>

Eg: Dim rollno as Integer.

19. Define the Project Explorer.

The project explorer window displays a hierarchical list of the projects and all other elements in the project. The elements of project explorer window are: view code, view object, Toggle folders.

20. What is message box?

Message boxes display information in a dialog box superimposed on the form.

They wait for the user to choose a button before returning to the application.

Message boxes can hold maximum of 1024 characters.

Syntax:

Msgbox Message, TypeofBox, TitleofBox

FIVE MARKS

1. Explain the data types supported by VB? Explain with example.(Apr'16, Apr'12, Nov'11)

There are five categories of data types of VB6:

- a) Integer
- b) String
- c) Date
- d) Boolean
- e) Variant

i. Integer Data Type

But these Data type categories are divided into more sub data types.

Byte	Store integer values in the range of 0 - 255
Integer	Store integer values in the range of (-32,768) - (+ 32,767)
Long	Store integer values in the range of (- 2,147,483,468) - (+ 2,147,483,468)
Single	Store floating point value in the range of (-3.4x10 ⁻³⁸) - (+ 3.4x10 ³⁸)
Double	Store large floating value which exceeding the single data type value
Currency	store monetary values. It supports 4 digits to the right of decimal point and 15 digits to the left

ii. String Data type:

Array of characters can be stored by using a string data type.. A string variable can be of two types, one is fixed length string variable and other is variable length variable. This data type is useful, when we collect the data from the text box.

iii. Date Data Type:

The date data type is used to store date and time values.

Example:

```
Dim DateOfBirth As Date
```

```
Dim Kickoff Time As Date
```

iv. Boolean Data Type:

Boolean data types hold either a true value or false value. These are not stored as numeric value and cannot be used as such. Values are internally stored as 0(false) and any nonzero value for representing true value.

v. Variant Data Type

This data type allows the user to store any type of data and is the default type in VB. If a variable is declared as variant, then it will have the ability to store any type of values. When any value is assigned to a variant variable, it can automatically adjust the data type to the corresponding value.

2. Explain the four properties of the Grid.

The Grid is very important for accurately positioning the controls.

Grids are the small dots available in the form window.

To control the grid in the form window choose

Tools → Options

Click General tab on the options dialog box.

The four properties of the Grid controls are:

- i) Show Grid
- ii) Grid Width
- iii) Grid Height
- iv) Align Controls to Grid

Show Grid: This option helps the user to turn on or off by changing the show grid setting.

The default setting is on. There is a little reason to turn the grid off.

Grid Width, Grid Height: The width and height text box helps us to set the distance between the grid marks (in twips). The default is 120 twips.

Align Controls to Grid: This check box determine whether controls automatically move to the next grid mark whether they can be placed between grid marks.

3. Write a short note on the following:

- a. TextBox
- b. CheckBox
- c. MessageBox

TextBox: A TextBox control is used to accept input from the user and display output provided by the application. Some general properties of the textbox are: name, text, maxlength, multiline etc. TextBoxes can word wrap automatically.

CheckBox: This control is a little square with an optional checkmark, which acts as a toggle. The CheckBox control lets the user to select or unselect an option. When the checkbox is checked, its value is set to 1 and when it is unchecked the value is set to 0. Every time the user clicks it, it changes the state.

MessageBox: Message boxes display information in a dialog box superimposed on the form. They wait for the user to choose a button before returning to the application.

Message boxes can hold maximum of 1024 characters.

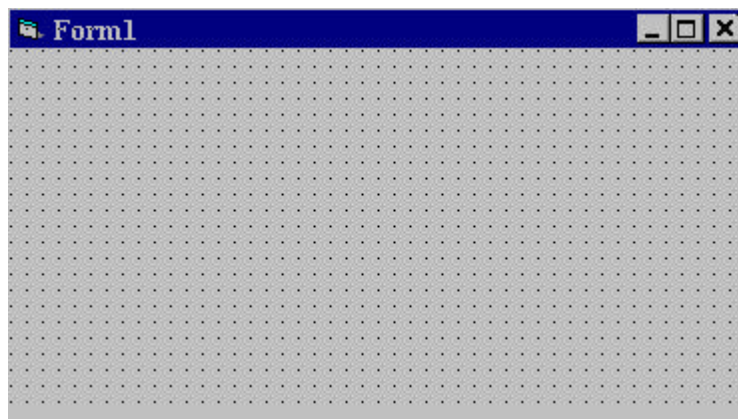
Syntax:

Msgbox Message, TypeofBox, TitleofBox

4. How to customize a FORM? (Nov'14, Apr'14)

The **Form Window** is central to developing Visual Basic applications.

It is where we draw our application.



The Toolbox is the selection menu for controls used in our application.

Altering a Form: To change the position of a form at run time,

- a) Move the cursor to the form Layout Window.
- b) Drag the form to the position where we want to appear at runtime.

The Project Window displays a list of all forms and modules making up our application.

Common Form properties:

- i. Caption: This property sets the title of the form.
- ii. Name: This property is used in the code to identify the object (control).
- iii. Appearance: It is used to determine how the form looks on the screen
- iv. BorderStyle: It determines border style of the form.

- v. Enabled: It determines whether an object can respond to user generated events
- vi. Font: This property retrieves windows font dialog box and used to set font for the information will be displayed on the form.
- vii. Control Box: This property determines whether the form contain control menu box at runtime which is used to minimize, maximize or close the form.

The **scale properties** of the form are used to position the object or text in a form accurately.

It has 5 scale properties. They are: ScaleMode, ScaleWidth, ScaleHeight, ScaleLeft, ScaleTop.

5. Explain the techniques to creating controls of a form.(NOV'12, NOV'14)

There are three ways to creating controls on a form. They are:

- i. Double clicking the control icon in the toolbox.
- ii. Click the required control in toolbox and move the mouse pointer to the form, and draw the control.
- iii. Select the control, hold the mouse button down and drag the mouse to create the object.

Resizing the Existing control:

- i. Use the properties window to adjust the width and height properties (OR)
- ii. Use the sizing handles (i.e) corner of the control

Moving an existing control:

- a. Move the pointer anywhere inside the control click the left mouse button until the control is at location and the release the button.
- b. Select the control, use CTRL + Arrow key to move the control.
- c. Directly adjust the left and top properties in the properties window.

6. Discuss the Message Boxes in VB with examples.

- Message boxes display information in a dialog box superimposed on a form.
- They wait for the user to choose a button before returning to the application
- Message boxes are used for displaying short messages. It can hold maximum of 1024 characters.

- Message boxes not only display data, it can also display an icon or one or more action button is for the user to choose from.
- Syntax:

MsgBox Message, TypeOfBox, TitleOfBox

Message → any combination of characters.

TypeOfBox → It determines the number of buttons and kind of buttons.

Constants:

VbOKOnly	0	Display OK button only.
vbOKCancel	1	Display OK and Cancel buttons.
vbAbortRetryIgnore	2	Display Abort , Retry , and Ignore buttons.
vbYesNoCancel	3	Display Yes , No , and Cancel buttons.
vbYesNo	4	Display Yes and No buttons.
vbRetryCancel	5	Display Retry and Cancel buttons.

TitleOfBox → It is used to add a title to the message box.

Example:

MsgBox "Example of Message Box", VbOkCancel, Test

Output:

Test
Example of Message Box Ok Cancel

7. Write a short note on: Image Controls.

Image controls hold pictures. Image controls can be used to display icons or pictures created with a program such as a Microsoft Paintbrush. They can also hold Windows metafiles or Jpegs or Gifts.

Load a picture into an image control by resetting the value of the Picture property. If choose the Picture property for the image control, this open the dialog box that lets you choose what image file to load.

Properties:

Stretch → It determines the whether the image control adjusts to fit the picture or the picture adjusts to fit the control. Its value is either true or false.

True – The graphic resize to fit the control.

False – (Default) The control resizes to fit the graphic.

Picture → Returns or sets a graphic to be displayed in a control. For the OLE container control, not available at design time and read-only at runtime.

BorderStyle → The borderstyle property for an image control has only two possible values.

0 -> No border

1 -> Fixed Single border

Some of the important events related to image control are: Click, OLEDragDrop, OLEDragOver, MouseUp, MouseDown, MouseMove etc.

8. What are command Buttons? Explain the properties of command buttons.

A command button is also known as push button. It is designed to respond to the user, when he clicked; it invokes a command that has been written into its Click event procedure. Most Visual Basic applications have command buttons that allow the user to simply click them to perform actions. More than 30 properties for a command button.

Some of them are explained below:

1. Name → This property determines the name used in the code.
2. Caption → It determines the text displayed on the command button
3. Visible → This property determines whether the command button is visible or not.
4. Enabled → This property determines whether the button is responsive or not.
5. Font → This property controls which font is used for the caption of the button.
6. Height, Width: → These properties define the height and width of the command button.
7. Left, Top → This property determine the distance between the command button and left and top edge of the form.

9. Write about tools on the edit toolbar. (APR'13)

- ✓ Edit toolbar contains several icons.

✓ To view the edit toolbar in Visual Basic application, go to view → Toolbars → Edit.

✓ It contains the following tools.

- i. List properties/Methods(Ctrl+J)
- ii. List Constants (Ctrl+shift+J)
- iii. QuickInfo(Ctrl+I)
- iv. ParameterInfo(Ctrl+Shift+I)
- v. Complete word(Ctrl+spacebar)
- vi. Indent (Tab)
- vii. Outdent (Shift+Tab)
- viii. Toggle Breakpoint (F9)
- ix. Comment Block

List properties/Methods(Ctrl+J)

Displays a pop-up list box with the properties and methods for the object preceding the period.

List Constants (Ctrl+shift+J)

Displays a pop-up list box with the valid constants.

QuickInfo(Ctrl+I)

Provides the syntax for procedure or method.

ParameterInfo(Ctrl+Shift+I)

It provides the parameter list or argument list of the current procedure (or) function.

Complete word(Ctrl+spacebar)

Completes the keyword or object.

Indent (Tab)

It moves the currently selected text or the cursor to move forward to one TAB position.

Outdent (Shift+Tab)

It moves the currently selected text or the cursor to move backward to one TAB position.

Toggle Breakpoint (F9)

It is used for debugging.

Comment Block:

Used to make the block of statements as non executable.

10. Discuss briefly on scale properties in a form.

The **scale properties** of the form are used to position the object or text in a form accurately.

It has 5 scale properties. They are: ScaleMode, ScaleWidth, ScaleHeight, ScaleLeft, ScaleTop.

ScaleMode:

It allows us to change the units used in form's internal coordinate system. It has seven values. Its default value is 1-twips.

ScaleWidth, ScaleHeight:

These properties are used to set custom scale for the height and width of the form.

ScaleLeft, ScaleTop:

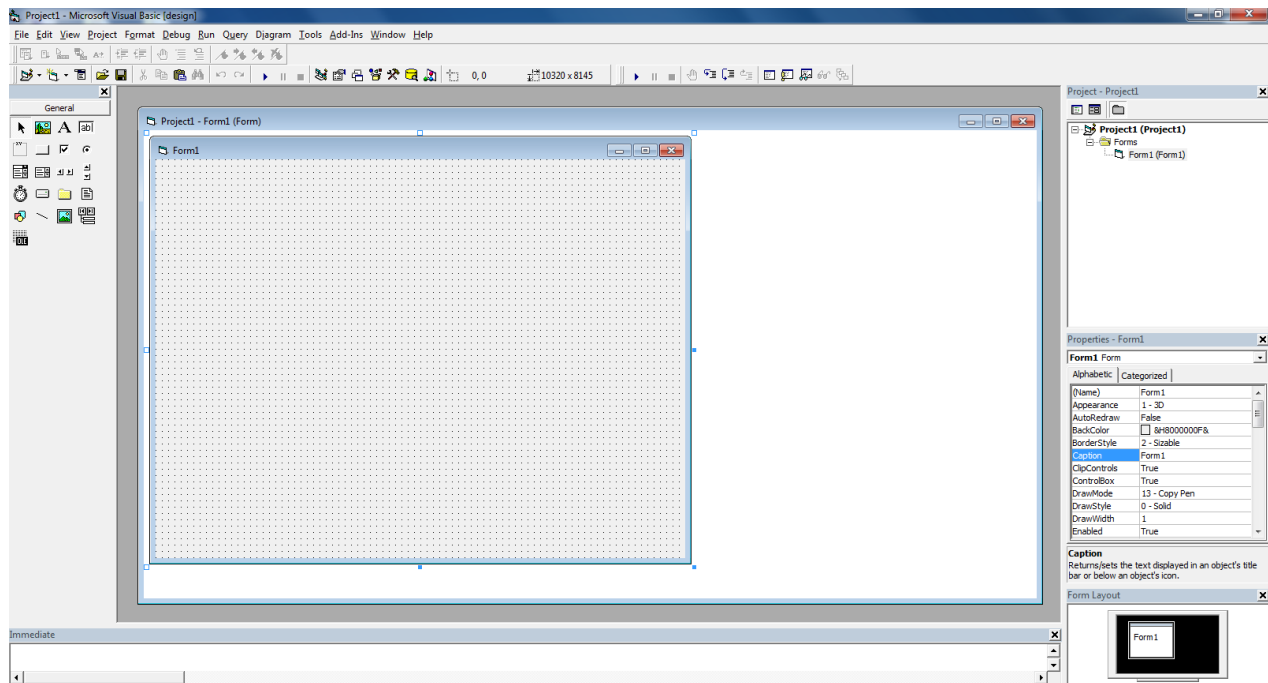
These properties are used to describe the values of left and top corner of the form.

TEN MARKS

1. Explain the components of IDE.
OR
Explain about IDE in detail.

IDE means Integrated Development Environment.

The visual basic IDE is provided when the Visual Basic Application is launched is as shown below.



- ❖ Applications in VB are developed in the Integrated Development Environment.
- ❖ Following are the components of IDE:
 - i. Menu Bar
 - ii. Tools box window
 - iii. Form window
 - iv. Properties window
 - v. Project explorer window
 - vi. Code editor window
 - vii. Form layout window
 - viii. Object browser
 - ix. Context menus
 - x. Standard toolbar

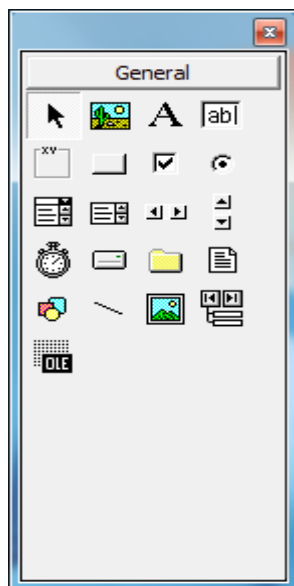
MENU BAR

- ❖ Menu bar is used to perform various tasks
- ❖ It is viewed under the title bar.
- ❖ Each option on the menu bar has its own drop-down list.
- ❖ Access keys and shortcut keys are provided

File	Edit	View	Project	Format	Debug	Run	Query	Diagram	Tools	Add-ins	Help
------	------	------	---------	--------	-------	-----	-------	---------	-------	---------	------

TOOLBOX WINDOW

- ❖ Toolbar contains a set of tools that are used at design time to place controls on a form.
- ❖ Using these controls, an interface between user and the application can be created.
- ❖ It contains more than 20 controls.



FORM WINDOW:

Forms are containers for controls.

User interacts with the controls on a form to obtain the desired result.

User can create and view controls on a form.

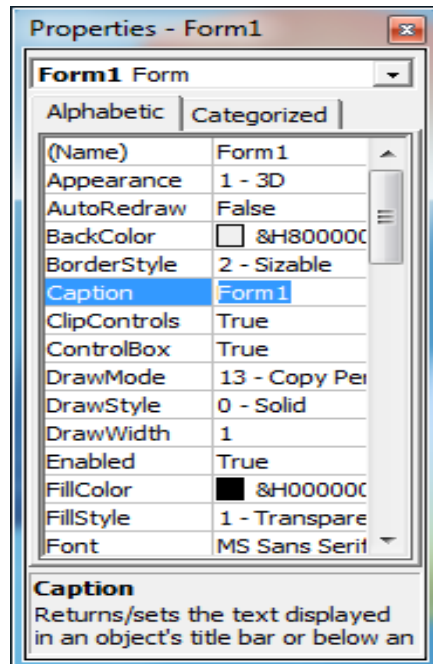
PROPERTIES WINDOW:

Properties window displays the list of design time properties for the selected objects and their current settings.

A property is a characteristic of an object such as size , caption, height, width, etc.

To view the properties window

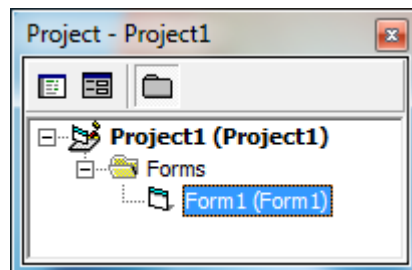
View->Properties or Press F4 shortcut Key.



PROJECT EXPLORER WINDOW:

The project explorer displays a hierarchical list of the projects and all other elements contained in the project.

The elements of the project explorer window are:



View Code->Shows the code window

View Object->Shows the form window

Toggle folders->View or hide the folders in the project explorer.

CODE EDITOR WINDOW:

The code window is used to write, display and edit VB code.

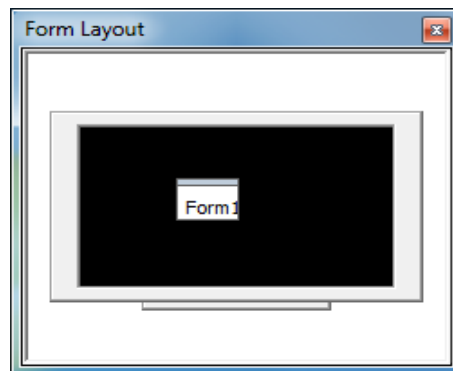
Following are the elements of code window.

- (i) Object List Box ->Displays the name of the selected object.
- (ii) Procedure/Events List Box->Displays all the events recognized by VB for form or the control displayed on the object box.

- (iii) View Selection Button->User can either view a single procedure or all the procedures in a module at one time. Shortcut Key is F7.

FORM LAYOUT WINDOW:

The form layout window decides in which position the form will be displayed at runtime. It allows the user to visually position the forms at design time.



OBJECT BROWSER

It displays the list of objects that can be used in the project. It also displays classes, properties, methods, events and constants in the current project.

CONTEXT MENU:

Context menu contains frequently used commands. To open a context menu, click the right mouse button on the object or press shift+F10.

STANDARD TOOLBAR:

The standard toolbar contains buttons that are shortcut to some commonly used items. Some of them are open project, save project, cut, copy, paste, etc.

2. Explain the various form properties.

The common form properties are:

- Name
- Appearance
- Border style
- Caption
- Control box
- Enabled
- Font
- Height, width
- Icon

Left, top
Mouse pointer, mouse icon
Startup position
Visible
Window state

Name:

- ✓ It is used to specify a name for the form.
- ✓ This property is used only in code.
- ✓ It is used to identify an object.
- ✓ Default value is "Form1".

Appearance:

- ✓ It is used to determine how the form looks on the screen.
 - ✓ It has two values.
- 1 → 3D(Default)
0 → Flat

BorderStyle:

It determines the border style of the form.

It has six values. They are:

0. None
1. Fixed single
2. Sizable(Default)
3. Fixed Dialog
4. Fixed tool window
5. Sizable tool window

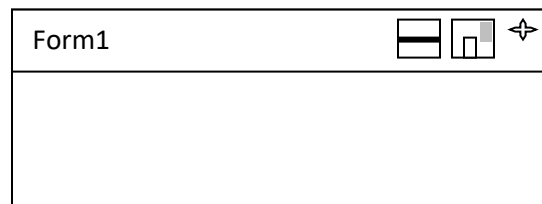
Caption:

Caption property sets the title of the form.

It is used to identify the form when it is minimized.

Control Box:

- ✓ This property determines the whether the form contain control menu box at runtime.
- ✓ Used to minimize, maximize or close the form.



- ✓ It has two values:
True (Default) → Displays control menu box.
False → Removes control menu box.

Enabled:

It determines whether an object can respond to user generated events.

True → Enables the form (default)

False → Disables the form.

Font:

This property retrieves window font dialog box and used to set font for the information will be displayed on the form.

Height, Width:

These properties determine the height and width of the form.

Icon:

It determines icon will be displayed when a form is minimized at run time.

Left, Top:

These properties determine the distance between the left or top of the form and the screen.

MousePointer:

✓ It is used to set the shape of the mouse pointer on the form.

✓ Its default value is 0 (means Arrow)

StartPosition:

✓ It determines the initial position of the form at runtime.

✓ Its default value is 2; form will be displayed at the center of the screen.

Visible:

✓ This property determines whether the form is displayed (visible) or not.

✓ It has two values.

True → Form will be displayed.

False → User cannot see the form.

WindowState:

✓ This property determines how the form will look like at runtime.

✓ It has three values: 0-Normal (Default); 1-Minimized; 2-Maximized.

Scale Properties:

The **scale properties** of the form are used to position the object or text in a form accurately.

It has 5 scale properties. They are: ScaleMode, ScaleWidth, ScaleHeight, ScaleLeft, ScaleTop.

ScaleMode:

It allows us to change the units used in form's internal coordinate system. It has seven values. Its default value is 1-twips.

ScaleWidth, ScaleHeight:

These properties are used to set custom scale for the height and width of the form.

ScaleLeft, ScaleTop:

These properties are used to describe the values of left and top corner of the form.

3. Explain the various data types supported in VB.

Visual basic provides several data types that are used to represent different types of values.

The data types are divided into two categories. They are:

1. Fundamental data types
2. User defined data types.

Fundamental Data Types:

- a) Integer
- b) String
- c) Date
- d) Boolean
- e) Variant

i. Integer Data Type

But these Data type categories are divided into more sub data types.

Byte	Store integer values in the range of 0 - 255
Integer	Store integer values in the range of (-32,768) - (+ 32,767)
Long	Store integer values in the range of (- 2,147,483,468) - (+ 2,147,483,468)
Single	Store floating point value in the range of (-3.4x10 ⁻³⁸) - (+ 3.4x10 ³⁸)
Double	Store large floating value which exceeding the single data type value
Currency	store monetary values. It supports 4 digits to the right of decimal point and 15 digits to the left

ii. String Data type:

Array of characters can be stored by using a string data type. A string variable can be of two types, one is fixed length string variable and other is variable length variable. This data type is useful, when we collect the data from the text box.

iii. Date Data Type:

The date data type is used to store date and time values.

Example:

```
Dim DateOfBirth As Date
```

```
Dim Kickoff Time As Date
```

iv. Boolean Data Type:

Boolean data types hold either a true value or false value. These are not stored as numeric value and cannot be used as such. Values are internally stored as 0(false) and any nonzero value for representing true value.

v. Variant Data Type

This data type allows the user to store any type of data and is the default type in VB. If a variable is declared as variant, then it will have the ability to store any type of values. When any value is assigned to a variant variable, it can automatically adjust the data type to the corresponding value.

User defined data types:

The user can create their own data types. Some of the user defined types are:

- i. Records
- ii. Enumerated data types.

Records:

A record is a mixed type variable. It may contain different kinds of data types.

Syntax:

```
Type data_type
```

```
    Variable
```

```
End type
```

Example:

```
Type student
```

```
    Rollno as Integer
```

```
    Sname as String
```

```
    DOB as data
```

```
End type
```

To declare a variable of this type student is,

Private stud as student (or) public stud as student

The dot (.) operator is used to access the variables of the record,

Stud.rollno =1001

Stud.sname= “abc”

Enumerated data type:

An enumerated data type provides a way to attach names to constraints. The “enum” keyword automatically enumerates a list of words by assigning the m vales like 0,1,2 and so on.

Syntax:

Enum data_type

 List of constants

End enum

Example:

Enum DaysOfWeek

 Sunday =1

 Monday

 Tuesday

 Wednesday

 Thursday

 Friday

 Saturday

End enum

4. Give the usage of the following

a) Message Box

b) Text Box

a) Message Box:

- i) Message boxes display information in a dialog box superimposed on a form.
- ii) They wait for the user to choose a button before returning to the application
- iii) Message boxes are used for displaying short messages. It can hold maximum of 1024 characters.
- iv) Message boxes not only display data, it can also display an icon or one or more action button is for the user to choose from.

Syntax:

MsgBox Message, TypeOfBox, TitleOfBox

Message → any combination of characters.

TypeOfBox → It determines the number of buttons and kind of buttons.

Examples:

VbOKOnly	0	Display OK button only.
vbOKCancel	1	Display OK and Cancel buttons.
vbAbortRetryIgnore	2	Display Abort , Retry , and Ignore buttons.
vbYesNoCancel	3	Display Yes , No , and Cancel buttons.
vbYesNo	4	Display Yes and No buttons.
vbRetryCancel	5	Display Retry and Cancel buttons.

TitleOfBox → It is used to add a title to the message box.

Example:

MsgBox "Example of Message Box", VbOkCancel, Test

Output:

Test		
Example of Message Box		
<table><tr><td>Ok</td><td>Cancel</td></tr></table>	Ok	Cancel
Ok	Cancel	

b) Text Box:

A TextBox control is used to accept input from the user and display output provided by the application. Some general properties of the textbox are: name, text, maxlength, multiline etc. TextBoxes can word wrap automatically.

Properties of TextBox

Enabled: Determines whether the TextBox can respond to user-generated events.

Locked: Determines whether the TextBox can be edited. You can prevent the user from changing the content of the TextBox.

MultiLine: Indicates whether it can accept multiple lines of text.

Alignment: Aligns the text to left justify, right justify or center.

MaxLength: Sets the maximum number of characters that can be entered. If you want to set a limit on the number of characters that the user can enter, you can do it very easily with the MaxLength property.

PasswordChar: This property is useful for a password program. For example, if u enter '*' as a value to this property, whatever character you type in, it will be shown as '*'.

ScrollBars: Says whether the TextBox has a scroll bar.

ToolTipText: Sets the text displayed when the mouse is paused over the control.

Events:

Text Box can recognize 23 events. Some of the important events are: GotFocus, LostFocus, KeyDown, KeyUp, KeyPress.

5. Explain the uses of TextBoxes and Labels with examples.

a). Text Box:

A TextBox control is used to accept input from the user and display output provided by the application. Some general properties of the textbox are: name, text, maxlength, multiline etc. TextBoxes can word wrap automatically.

Properties of TextBox:

Enabled: Determines whether the TextBox can respond to user-generated events.

Locked: Determines whether the TextBox can be edited. You can prevent the user from changing the content of the TextBox.

MultiLine: Indicates whether it can accept multiple lines of text.

Alignment: Aligns the text to left justify, right justify or center.

MaxLength: Sets the maximum number of characters that can be entered. If you want to set a limit on the number of characters that the user can enter, you can do it very easily with the MaxLength property.

PasswordChar: This property is useful for a password program. For example, if u enter '*' as a value to this property, whatever character you type in, it will be shown as '*'.

ScrollBars: Says whether the TextBox has a scroll bar.

ToolTipText: Sets the text displayed when the mouse is paused over the control.

Events:

Text Box can recognize 23 events. Some of the important events are: GotFocus, LostFocus, KeyDown, KeyUp, KeyPress.

b). Labels:

- ✓ Label is used to display information
- ✓ It is also used to display text such as status messages and other program information.

Properties of Labels:

- i. Caption: This property is used to change the text displayed in the label control.
- ii. Alignment: This property is used to align the text in the label. It has three values: 0 – Left aligned, 1-Right aligned, 2- Center aligned.
- iii. Wordwrap: This property is used to resize the text entered in the caption property. If the text exceeds the width of the label control, the text wraps to the next line. Its value is either true/false.
- iv. Autosize: This property causes the control to horizontally expand and adjust to the size of its contents.
- v. Borderstyle: The borderstyle property has two values 0 & 1. 0-No border (default); 1- Single border occurred.

6. Explain Command buttons and Message Boxes.

For command buttons- Refer Question No.8 in 5marks.

For Message Boxes Refer Question no 4(a) in Ten Marks.

UNIT-II

Two Marks

1. Differentiate decision statements and a loop statement (Nov. 11)

Control structures are statements used to change flow of the program. There are two types of control structures available :

- (a) Decision statements : also called conditionals. At this point, the programs can only decide which group of statements to be execute based on condition. Example : If statements, Select case
- (b) Loop statements :at this point the programs can only decide whether to repeat a group of statements or not based on condition. Example : For statements, while and do-until

2. What is meant by Group box controls (Nov. 11)

The **GroupBox** control is good for **grouping related controls** or the **RadioButton** controls in a **RadioButton** group. This control provides a visible **border** and **caption** which can help the user when works with a **complicated** form containing **lot of controls**. In other words, it makes the **interface** much easier for the user to **understand**. **GroupBox** displays a **border** and **caption** which is good for **decoration** and **visual appealing**.

3. How will you display information on a form (Nov. 12)

Print : used to display information on form syntax : `FormName.print expression`

CurrentX : refers to the horizontal position where visual basic will display the information

Syntax: `FormName.CurrentX=value`

CurrentY : refers to the vertical position where it will display the information

Syntax: `FormName.CurrentY=value`

Writeline : used to display information on form's newline Syntax: `FormName.Writeline expression`

Ex: `Form1.CurrentX(150) Form1.CurrentY(150) Form1.print "GOOD MORNING"`

4. What are determinate loop (Nov. 12)

Looping Statements : used to repeat group of statements until it satisfying a specified condition. There are two types of looping statements available (a) determinate (b) in-determinate

Determinate Loop : this repeating an operation for fixed number of times based on condition example :For loops and nested for loops

Syntax : For exp=start to end

<statements>

Next

5. Write a note on : round function (Nov. 12)

This function that rounds up a number to a certain number of decimal places with immediate next number. The format is Round(n,m) which means to round a number n to m decimal places example : Round(7.2567)= 7.26

6. Name any two built-in functions (Nov. 14)

Built-in functions are one, having predefined meaning and fixed syntax. There sever built-in functions available in visual basic :

(A) String handling functions : Len(),Trim()

(B) Numeric functions : Sqr(),Log()

(C) Date and time functions: Weekday(),Minute()

(D) Financial functions : Nper(),IRR()

7. Write the syntax of Select case control structure in VB with an example (Nov. 15)

If we want to design a program, to make decision based on several available options

Syntax : Select case Expression

Case val1

Statements

Case val2

Statements

....

Case else

Statements

End select

8. Write the syntax of format function with an example (Nov. 15)

This function used to print the copy of original phrase in a specified format. Template of this function is :Format(NumericExpression, FormatString)

Example :Me.Print Format(123.456782,"###.###) = 123.457

9. What are two kinds of general procedure in VB (Nov. 13)

In visual programming general procedures plays vital role, all codes in VB are written in the format of procedures. General procedures are self -contained program, for a specific task. There are two types of procedures available in VB are :

(a) Sub Procedure : it is a piece of coding for a specific function, it is accept inputs from the calling procedure but It can't return value back.

[public/private/friend]Sub Procedure-name(Parameters list)

(b) Function Procedure : it is a block of coding of specific task, capable of accepting inputs from calling procedure, and can also return values to main program.

[public/private/friend]function function-name(parameters list) as return-type.

10. Write a recursive function in VB for calculating factorial of a number (Nov. 13)

Function Fact(n as interger)as integer

 If n==0 then

 Fact=0

 Else

 Fact=n*fact(n-1)

 End if

End funcntion

11. Give the general syntax of Do-Until loop with an example (Apr. 13)

Used to repeat block of statement until the specified condition is satisfied.

Syntax : Do

<statements>

 Loop until exp<>value

Example : dim n as integer

 N=5

 Do

 Form1.print "Loop parameter"&n

 N=n+1

 Loop until n<=10

12. List the name of any four numeric built-in functions (Apr. 13)

- (i) Int : it converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number.
Syntax : int(num) **example :** int(2.4)=2, int (-4.6)=-5
- (ii) Sqr : it computes the square root of a number, and the number must be non – negative number. **Syntax :** sqr(number) **example :** sqr(9)=3
- (iii) Abs: it returns the absolute value of a number, it removes the minus sign. Usually it used to calculate the distance. **Syntax :** abs(num) **example :** Abs(-6)=6
- (iv) Log : this function that returns the natural logarithm of a number. Syntax log num
example : Log 10 = 2.302585

13. Write any two types of dialog box (Nov. 10)

- (a) Input box : alternative way of getting inputs from the user. This displays a modal dialog box on the screen. Modality is the principal advantage of input boxes, with which can insist a user supply some information, but we can't set the dimension of the input box

Syntax :

variable = inputbox(prompt,string[,title][,default][,xpos][,ypos][,helpfile,context])

example : str=inputbox("Enter Value")

- (b) Message Box : Message box display information in a dialog box superimposed on the form. They wait for the user to choose a button before returning to the application. Users cannot switch another form as long as visual basic is displaying a message box. Message boxes should be used for short messages or to give transient feedback, it can hold 1024 characters. Syntax : MsgBoxmessageinbox, typeofbox, titleofbox

Eg :MsgBox "Example text",1,64

14. What are style properties of Combo box (Nov. 10)

There are three types of combo boxes, and which one you get depends on the value of the style property:

- (a) If style property=0, can get the combo box with arrow mark. If the user click the arrow mark, can get the drop down list of choices
- (b) If style property=1, the user see the combo box with the list already dropped down
- (c) If style property=2, can see the pull-down list box.

15. Define types of loop in VB (Apr. 12)

Loop is a control structure used to repeat a block of statements based on conditions. There are two types of control structures available in visual basic :

- (a) Determinate loop : this repeating an operation for fixed number of times based on condition example :For loops and nested for loops
- (b) In-determinate loop :continuing until reach a predefined specific goal or until certain initial conditions have finally changed . example : while loop, do-until loop

16. Define with statement (Apr. 12)

This statement used to change/set multiple properties of the single control of visual basic, for example to set height, width and text property of the text box then :

With text1

.height=2000

.width=2000

.text="example for with statement"

End with

17. Explain about goto statement (Apr. 11)

VB retain unconditional jump statement called goto. The routine which using goto statement leads to spaghetti code. This is very hard to read and harder to debug. But in some cases using GOTO actually makes your code cleaner and easy to understand.

Syntax :goto label

.....

Label : code

Example : for i= 0 to 10

 getdata :=inputbox("give input","enterdata ZZZ to end")

 If getdata ="ZZZ" then

 Goto badinput

 Else

 Process data

 End if

Next i

 Badinput :msgbox"error in input"

Five Marks

1. Explain the looping structure with suitable example (Nov. 14)

Loop is a control structure used to repeat a block of statements based on conditions. There are two types of control structures available in visual basic :

- (a) Determinate loop : this repeating an operation for fixed number of times based on condition example :For loops and nested for loops
- (b) In-determinate loop :continuing until reach a predefined specific goal or until certain initial conditions have finally changed . example : while loop, do-until loop

For Loop :

syntax : For exp =initial to end

 <statements>

Next exp

example : for i=0to10

 msgbox “welcome to VB”

next i

Nested For :

syntax : For exp1= initial to end

 <statements>

 For exp2=initial to end

 <statements>

 next exp2

next exp1

example : for i=0 to 10

 print i

 for j=0 to 10

 print j

 next j

next i

Do-while :

syntax : do

 <statements>

Loop While condition

example : dim n as integer n=1

do

 form1.print n

 n=n+1

Loop while n<=10

Do-until:

Syntax : do

 <statements>

Loop until condition

example : do

 form1.print n

 n=n+1

Loop until n>=5

While :

syntax : while condition

<statements>

Wend

example : while n<=5

form1.print n

n=n+1

Wend

2. Write short notes on : Financial function (Nov. 12)

Financial functions are used for handling standard calculations that everyone will occasionally need to do. Various financial functions available are:

- (a) **Pmtfunction** : it returns the payment for an annuity based on periodic, constant payments and a constant interest rate. Syntax : Pmt(RateperPeriod, NumPeriods, whatYouStartWith, WhatYouEndUpWith, WhenDoYouPay) example : Mp=Pmt(0.08/2,30*12,100000,0,1)
- (b) **IPmtfunction** : it returns double value that indicates the interest payment for a fixed-period annuity based on a number of periodic fixed payments the present value(Pv) of a series of payments. Syntax : IPmt(Interesrrate,Fixedpayments,NPer,Pv[,fv[,type]]) example : IPmt(0.08/12,25,360,100000,0,1)
- (c) **FV** : it gives the future value of an annuity based on the periodic payments and a constant interest rate. Syntax : FV(rate,Nper,pmt[,pv[,typr]])
- (d) **IRR** : this function returns a double value indicating the internal rate of return for an array of values that represent cash flow. The *values()* array must contain at least one negative value(payment) and one positive value(receipt) guess(optional) specified an estimate to be returned by IRR. Syntax : IRR[Values()][,guess]
- (e) **NPer** : it returns a double value that indicates the number of periods for an annuity based on periodic fixed payments and fixed interest rate. Syntax : NPer(rate,pmt,pv[,fv[,type]]) example : NPer(.05/12,-1000,100000,0,1)

3. Explain about function procedures with example (Nov. 12)

Functions : it is a self -contained program. User defined functions are created to perform some specific task. It is capable of accepting parameter as inputs from the calling program, and can return the values back to the main program. Main purpose of functions are reusability of code.

Syntax :

```
[private/public/friend] function function-name(exp as datatype<parameter list>) as  
returntype
```

```
<statements>
```

```
Function-name=return type
```

```
End function
```

Example :

```
Public function randomgen(n as integer) as integer
```

```
    Form1.print "Example for user defined function"
```

```
    randomgen=int(n*rnd)
```

```
end function.
```

4. How functions differ from procedures? Explain (NOV. 14)

In programming languages Functions and Procedure are the code segments written separately to take advantage of reusability. If you want write the same code segments many places in the programming file, it is likely that your program would benefit from Functions or Procedures , rather than duplicating code segments in multiple places.

SUBPROCEDURE	FUNCTION
Sub Procedure is procedure that performs some action.	Function Procedure also perform some action but also returns some value to point from which it was called
Syntax for Sub Procedure is as follows: Private Sub <procedure name> ‘ Actions to performed End Sub	Syntax for Function Procedure is as follows: Private Function Commission() ‘ Action to performed Commission = <value that to be return> End Function
Example: Form_Load()	Example: Val(), FormatPercentage()

5. List out the string functions with example (Nov. 15)

Information in vb text boxes is usually kept as strings. So handling the strings in Visual basic is necessary and Visual basic having most powerful built-in string handling functions.

- (i) Len : used to know the length of the string syntax : Len(string) ex : len("visual basic")=12
- (ii) Space : used to insert empty spaces in between the string syntax : space(noofspace)
ex : space(3)
- (iii) Trim : used to remove unwanted space in both the sides of the string syntax:
trim(string) ex : trim(" visual basic ")="visual basic"
- (iv) LTrim, RTrim : used to remove the empty spaces of the left and right hand side of the strings respectively . syntax : Ltrim(string), Rtrim(string)
ex: rtrim("visual ")= "visual"
- (v) Mid : used to retrieve substring or part of string from the phrase based on the index syntax: Mid(string,start[, length]) ex : mid("visual basic",2,4)="sual"
- (vi) Replace : used to replace a character or part of string from the original phrase
syntax :replace(string,find,replacewith[,start][,count][,compare]) ex :
replace("AAAAA","A","Z",1,1)=ZAAAA
- (vii) Stringcomp : used to compare two strings and return integer value as result, if both the strings are equal then it returns 0 if 1st string is less means it returns -1 or it return 1. Syntax : stringcomp(string,string) ex: stringcompr("A","B")=-1

6. Explain the decision making statements with their syntax and example (Nov. 15)

Decision statements also called conditionals. At this point, the programs can only decide which group of statements to be execute based on condition.

If statement

Syntax : if condition then

<statement>

End if

example : if n==0 then

msgbox " welcome"

end if

If-else statement

Syntax: if condition then

<statement>

example : if n==0 then

msgbox " welcome"

```
Else
<statement>
End if
```

Nested if statement

Syntax: if condition then

```
    <statement>
Else if
    <statement>
Else
    <statement>
End if
```

Select case

Syntax: select case expression

```
    Case exp1
<statement>
    Case exp2
<statement>
    ...
    Case else
<statement>
    End select
```

```
else
    msgbox :good bye"
end if
```

example : if n==0 then

```
    me.print "zero"
else if n>0 then
    me.print "positive"
else
    me.print "negative"
end if
```

example : select case grade

```
case is >90
    yourgrade="a"
case is >60
    yourgrade="b"
case else
    yourgrade="c"
end select
```

TEN MARKS

1. Design a form and with their properties and coding to implement at least 5 numeric and date and time functions (Nov. 11)

EXAMPLE FOR BUILT – IN FUNCTIONS

ENTER THE NUMBER

Square root

Round - off

Exponential

Date value

Time value

```
Private sub form1_load()  
Form1.caption="Built-in function"  
Form1.fontname="Times New Roman"  
Form1.backcolor="vbblue"  
Form1.fontstyle="bold"  
End sub  
  
Private sub command1_click()  
Dim n as integer  
n=val(text1.text)  
msgbox "Square root is "&sqr(n)  
End sub  
  
Private sub command2_click()  
Dim n as integer  
n=val(text1.text)  
msgbox "Round value of the number is "&round(n)  
End sub  
  
Private sub command3_click()  
Dim n as integer
```

```

n=val(text1.text)
msgbox "Exponential value is "&exp(n)
End sub

Private sub command4_click()
msgbox "Date value"&DateValue(now)
End sub

Private sub command5_click()
Dim n as integer
n=val(text1.text)
msgbox "Time value"&TimeValue(now)
End sub

```

2. Discuss briefly on Numeric functions with example (Nov. 12)

There are several numeric functions are available in visual basic :

- (a) Int : it converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. Syntax : int(num) example : int(2.4)=2, int (-4.6)=-5
- (b) Sqr : it computes the square root of a number, and the number must be non – negative number. Syntax : sqr(number) example : sqr(9)=3
- (c) Abs: it returns the absolute value of a number, it removes the minus sign. Usually it used to calculate the distance. Syntax : abs(num) example : Abs(-6)=6
- (d) Log : this function that returns the natural logarithm of a number. Syntax log num example : Log 10 = 2.302585
- (e) Exp : this function gives e to the power x, where e is the base for natural logarithms, and x the value in the parentheses. The answer is a single precision if x is an integer or single or else the answer is double. Syntax : exp(num) example : exp(1)=e¹=2.7182818284590
- (f) Round : it rounds up a number to a certain number of decimal places. Syntax : round(n,m) which means to round a number n to m decimal places. Example : round(7.2567,2)=7.26

3. Explain the following with example (a) the format function (b) rich text box (Apr. 12)

(A) Format Function : This function used to print the copy of original phrase in a specified format.

Template of this function is : `Format(NumericExpression, FormatString)`

Example : `Me.print Format(123.45678,"###.##")=123.46`

`Me.printFormat(123456789.99,"#,###")=123,456,789.99`

`Me.prinrFormat(now,longdate)=Wednesday, November, 01,1995`

`Me.prinrFormat(now,mediumtime)=11.08PM`

Name of Format	Description
General Number	Gives the string of digits with no thousands separator
Currency	Uses appropriate thousands separator, after decimal point two digits will display
Fixed	Display at least one digit to the left and 2 digit to the right of the decimal point
Percent	Gives the number in percent form(% after it)
True/False	Display false if the number is 0 or display true
General date	Display date / time if there no fractional point in case of date, no integer point in case of time
Short date	Display date using short date form
Long time	Display time with hour, minute, second
Short time	Displays time with 24 hour clock

(B) Rich text box :it supplied with most versions of visual basic. It is one of the most useful control in toolchest. The rich text box is a custom control, we have need to add it to the tool box. 1. Choose project/components to open the components dialog box

2. choose Microsoft RichTextBox Control6.0

The control lets you display text with multiple fonts and sizes without having to go out and buy third party custom control. More ever it is not limited 32k characters like an ordinary text box.

Properties :

selLength : returns set of chars selected
selStart : return set of starting char of selected text
selText : returns set of string that is selected
selColor : sets color to the selected text
selFontName: lets you change the font
selBold, selItalic, selUnderline, selStrikethru : set the font styles of the selected text respectively.
selFontSize: let you change font size of the currently selected text

4. Distinguish between passing by reference and value (Apr. 11)

Functions are self-contained program used to perform some well-defined task. Main purpose of functions are reusability of the code. The function having capability of accepting parameters from the calling program and can return value to main program. While sending values from main program to a function or a sub procedure there are two ways available :

- (a) Pass-by-value : In this case the copy of an original value of the parameter is send to the sub function, so whatever the changes made on that value only be done inside the function, it will not affect the original values at any cost.

Example :

```
Sub form_load()  
Dim amt as integer  
amt=2  
Print "Inside form load and the value"&amt  
Triple (amt)  
Print "after called the function"&amt  
End sun  
Sub Triple(num as integer)  
Num=3*num  
Print"I am inside the function and value"&amt  
End sub  
Output : Inside form load and the value2
```

I am inside the function and value 6

After called the function 2

- (b) Pass-by-reference: In this case the address of the original value is send to the function, so the changes made in the variable will also reflect the value.

Example :

```
Sub form_load()
```

```
Dim amt as integer
```

```
amt=2
```

```
Print "Inside form load and the value"&amt
```

```
Triple amt
```

```
Print "after called the function"&amt
```

```
End sun
```

```
Sub Triple(num as integer)
```

```
Num=3*num
```

```
Print"I am inside the function and value"&amt
```

```
End sub
```

Output : Inside form load and the value2

I am inside the function and value 6

After called the function 6

5. What are common dialog controls? Explain (Nov. 10)

The common dialog control provides a standard set of dialog boxes for operations such as opening and saving files, setting print options, and selecting colors and fonts. The control also has the ability to display Help by running the Windows Help engine.

The common dialog control



The common dialog control provides an interface between Visual Basic and the procedures in the Microsoft Windows dynamic-link library Commdlg.dll. To create a dialog box using this control, Commdlg.dll must be in your Microsoft Windows \System directory.

You use the common dialog control in your application by adding it to a form and setting its properties. The dialog displayed by the control is determined by the methods of the control. At

run time, a dialog box is displayed or the Help engine is executed when the appropriate method is invoked; at design time, the common dialog control is displayed as an icon on a form. This icon can't be sized.

The common dialog control allows you to display these commonly used dialog boxes:

- Open
- Save As
- Color
- Font
- Print

To use the common dialog control

1. If you haven't already done so, add the common dialog control to the toolbox by selecting **Components** from the **Project** menu. Locate and select the control in the **Controls** tabbed dialog, then click the **OK** button.
2. On the toolbox, click the **CommonDialog** control and draw it on a form.
When you draw a common dialog control on a form, it automatically resizes itself. Like the timer control, the common dialog control is invisible at run time.
3. At run time, use the appropriate method, as listed in the following table, to display the desired dialog.

Method	Dialog displayed
ShowOpen	Open
ShowSave	Save As
ShowColor	Color
ShowFont	Font
ShowPrinter	Print
ShowHelp	Invokes Windows Help

Code changes for Displaying a File Save Dialog Box

The following code demonstrates displaying a **File Save** dialog box, saving the file to the application's folder.

```
' Visual Basic 6.0
```

```
' Uses a CommonDialog control.
```

```
CommonDialog1.InitDir = App.Path
```

```
CommonDialog1.ShowSave
```

VB

```
' Visual Basic
```

```
' Uses a SaveFileDialog component.
```

```
SaveFileDialog1.InitialDirectory = My.Application.Info.DirectoryPath
```

```
SaveFileDialog1.ShowDialog()
```

Code changes for Displaying a Print Dialog Box

The following code demonstrates displaying a **Print** dialog box, printing a file located in the application's folder.

```
' Visual Basic 6.0
```

```
' Uses a CommonDialog control.
```

```
CommonDialog1.FileName = App.Path & "MyFile.txt"
```

```
CommonDialog1.ShowPrinter
```

VB

```
' Visual Basic
```

```
' Uses PrintDocument and PrintDialog components.
```

```
PrintDocument1.DocumentName = My.Application.Info.DirectoryPath _  
& "MyFile.txt"
```

```
PrintDialog1.Document = PrintDocument1
```

```
PrintDialog1.ShowDialog()
```

Code changes for Displaying Help

The following code demonstrates displaying a Help file from your application, opening it to the table of contents.

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.HelpFile = "C:\Windows\Help\calc.hlp"
CommonDialog1.HelpCommand = cdlHelpContents
CommonDialog1.ShowHelp

VB
' Visual Basic
' Uses the Help.ShowHelp method.
Help.ShowHelp(Me, "file://C:\Windows\Help\calc.chm", _
HelpNavigator.TableOfContents)
```

6. Explain about date and time function(Nov. 14)

Application need to be able to access and work with date and time values. Many applications are written for business and scientific purposes, where recording the date and time of the program run is vital to the success of the project.

- (A) Date value : this function returns the date portion of date/time value, with the time portion zeroed out.(the time is 12.00 am if that is zeroed out)
example : dim dtm as date, dtm=DateValue(now) = 8/31/2001 with time 0(12.00am)
- (B) TimeValue: this function returns time portion of date/time value, with the date portion zeroed out.(the date value is 12/30/1899 if zeroed out)
example: dim ddt as date, ddt=DateValue(now) = 9:15:20 PM with date 0(12/30/1899)
- (C) Weekday : it returns a number from 1 to 7 indicating the day of the week for a given date, where 1 is Sunday 7 is Saturday
example : intDOW=Weekday(now) = 6
- (D) WeekdayName : returns a string containing the week day name from Sunday to Saturday example : StriDOW=WeekdayName(now)="Friday"
- (E) Month : this function is used to return a number from 1 to 12 indicating the month portion of a given date
example : intMonth = Month(now) = 8

(F) Day : used to return a number from 1 to 31 indicating the day portion of a given date
example : `intDay=Day(now) = 31`

(G) Year : this function returns a number from 100 to 9999 by indicating the year portion of the date. Example : `intYear =Year(now) = 2001`

(H) Hour : this function returns an integer specifying a whole number 0 to 23 representing the hour of the day.

Example : `intHour=Hour(now)=21(9 PM)`

(I) Minute : this function returns integer specifying a whole number between 0 and 59 representing the minute of the given hour.

Example : `intMin =Minute(now)=15`

(J) Second :returns an integer specifying a whole number between 0 and 59 representing the second value of the given minute.

Example : `intSec=Second(now)=20`

(K) DateSerial : takes three numeric arguments, year, month and day respectively, it returns a date based on those values

Example : `dim ddt as Date, ddt=DateSerial(intYR,intMON,intDay)`

(L) TimeSerial : takes three numeric arguments : hour, minute and second respectively. It returns time based on those values

Example : `dim dtm as Date, dtm=TimeSerial(intHR,intMIN,intSEC).`

UNIT-III

1. Write short notes on list boxes?(a12)

- A list box control displays a list of items from which the user can select one or more.
- List boxes present a list of choices to the user.
- By default, the choices are displayed vertically in a single column, although you can set up multiple columns as well

2. Differentiate check box and option button?(n13)

- In Check box, you can select multiple options. In option button, you can select one option
- Option buttons are circular and check boxes are square

3. What is control Array?(n11)

- A control array is a group of controls that share the same name and type. They also share the same event.
- It has a one element and can grow to as many as your system resources and memory permit, its size also depends on how much memory and windows resources each control requires.

4. Write short notes a data grid?(a11)

- It displays a series of rows and columns representing records and fields from a Record set of Objects
- The data grid control can be quickly configured at design time with little or no code.

5. What are Arrays?(a13)

- An array is a consecutive group of memory locations that all have the same name and the same type
- To refer to a particular location or element in the array, we specify the array name and the array element position number

6. What are combo boxes?(a14)

- A combo boxes combines the feature of textbox & list box.
- This control enables the user to select either by typing the text into combo box or by selecting an item from the list.

7. How to declare arrays?(a13)

- An Array can be declared as follows
Dim [Preserve] var name (Subscript) [Type]
- Dim syntax has the following parts

Part	Description
Preserve	Optional Keyword used to preserve the data in an existing array when you can change the size of the last dimension
Varname	Required name of the variable
Subscript	Required dimensions of an array variable
Type	Data type of a variable

8. Define record?(a12)

- It used at module level to define a user-defined data type containing one or more elements
- It create record with type name as data type

9. Define Grid?(a13)

The grid is so important for accurately positioning controls, mastering it will help you give your applications a finished, professional look. In order to control the grid, choose Tools|Options (ALT+T, O), and then go to the General page on the Options dialog box. The four properties you can control are described next.

10. What is a list?(a14)

- A Set of sequentially indexed elements having the same intrinsic data type.
- Each element of an array has a unique identifying index number. Changes made to one element of an array don't affect the other elements.

11. Why sorting is to be done?(n12)

- In order, To arrange data & to retrieve data from memory easily, the data items should be sorted

12. What is an upper bound and lower bound?(n15)

- The upper bound is the highest values and the lower bound is the lowest values in index values that can be used in functions

- Those functions can be used with static arrays as well as dynamic arrays

13. What is a fixed array?(a11)

- Memory allocation is fixed as the same when the program runs.
- The array cannot be changed during runtime

Eg: Private a(13) as String

Hold only 14 characters

14. What is a preserve keyword used for?(n13)

- Preserve Keyword is used to preserve the values in the array even after the array has been eliminated

15. State any two differences between fixed and dynamic array?

FIXED ARRAY	DYNAMIC ARRAY
Values cannot be changed during runtime	Values can be allocated during runtime.
Memory space is determined while declaration.	Memory is allocated dynamically.

16. What is dynamic array?(a15)

- The array size can be changed during the runtime

Eg: Private a() as String

Dim no() as Integer

No = Inputbox("How many no's")

17. Define Searching?(a16)

- It referred as to search a number or a string in a long list
- There are numerous methods for searching. Binary search is one of them

18. Define Sorting?(a15)

- Sorting is arranging data in order.
- There are many different ways to sort the data.
- There are four different methods for sorting, they are Ripple sort, Bubble sort, Insertion sort and Shell sort

19. Define STR COMP (String Compare)?(n12)

- Sometimes the user while evaluating a string it is to be determined whether the string is less than, equal to (or) greater than, for that the string compare function is used

Syntax: Str Comp (String 1, String2, Compare)

20. Define find method?(n14)

- It searches the text in a Rich Text Box control for a given string

Syntax:

object.Find(string, start, end, options)

21. What is WITH STATEMENT used for?(a13)

- Instead of using multiple periods for accessing, WITH Statement can be used.
- The WITH STATEMENT is a convenient method for quick access.

22. What is Enumerated Constant?(n11)

- It is the enumerated constant that a user can create using a list of variables specified with a constant value.

23. How to add control in a control arrays?(n12)

- ❖ To add a control to control array at design time by simply copying and pasting by ctrl+c, ctrl + v element can be added during the design time.

FIVE MARKS

1. How projects are carried in multiple forms?(n11)

Add Form Command (Project Menu)

It Displays the Add Form dialog box so you can insert a new or existing form into your active project.

Only available at design time.

Toolbar shortcut

Multiple-Document Interface (MDI) Applications

- The multiple-document interface (MDI) allows you to create an applications that maintain multiple forms within a single container form.
- Applications such as Microsoft Excel and Microsoft Word for Windows have multiple-document interfaces.
- A child form is an ordinary form that has its MDIChild property set to True. Your application can include many MDI child forms of similar or different types.
- At run time, child forms are displayed within the *workspace* of the MDI parent form (the area inside the form's borders and below the title and menu bars). When a child form is minimized, its icon appears within the workspace of the MDI form instead of on the taskbar.

Creating an MDI Application

Use the following procedure to create an MDI form and its child forms.

To create an MDI application

1. Create an MDI form

From the **Project** menu, choose **Add MDI Form**.

Note. An Application can have only one MDI form. If a project already has an MDI Form command on the project menu is unavailable.

2. Create the application's child forms.

To create an MDI child form, create a new form (or open an existing one) and set its MDIChild property to true.

An MDI form is similar to an ordinary form with one restriction. You can't place a control directly on a MDI form unless that control has an Align property (such as a picture box control) or has no visible interface (such as a timer control).

2. What are control arrays? Explain their usage in VB with example?(a11)

CONTROL ARRAYS

- A control array is a group of controls that share the same name and type. They also share the same event procedures.
- It has at least one element and can grow to as many elements as your system resources and memory permit, its size also depends on how much memory and windows resources each control requires.
- The maximum index you can use in a control array is 32767. Elements of the same control have their own property settings.
- Common uses for control arrays include menu controls and option button groupings.

Why Use Control Arrays?

- Adding controls with control arrays uses fewer resources than simply adding multiple controls of the same type to a form at design time.
- Control arrays are also useful if you want several controls to share code
- For example, if three option buttons are created as a control array, the same code is executed regardless of which button was clicked.

Creating a Control Array at Design Time

There are three ways to create a control array at design time:

- Assign the same name to more than one control.
- Copy an existing control and then paste it onto the form.
- Set the control's Index property to value that is not Null.

To add a control array element by changing its name

1. Draw the controls you want to be in the control array. (The controls must all be of the same type.) Decide which control will become the first element in the array.
2. Select one of the controls and change its Name setting to the Name setting for the first element in the array.

3. When you type an existing name for a control in the array, Visual Basic displays a dialog box asking you to confirm that you want to create a control array. Choose **Yes** to confirm the action.

To add a control array element by copying an existing control

1. Draw a control in the control array.
2. While the control has the focus, choose **Copy** from the **Edit** menu.
3. From the **Edit** menu, choose **Paste**. Visual Basic displays a dialog box asking to confirm that you want to create a control array. Choose **Yes** to confirm the action.
This control is assigned an index value of 1. The first control you drew has a value of 0.

Adding to a Control Array at Run Time

You can add and remove controls in a control array at a run time using the Load and Unload statements. Use this syntax:

Load *object*(index %)

Unload *object*(index %)

Argument	Description
<i>Object</i>	Name of the control to add to or delete from the control array
Index%	The control's index value in the array

Note: Visual Basic generates an error if you attempt to use the Load statement with an index number already in use in the array

Index Property (Control Array)

Returns or sets the number that uniquely identifies a control in control array. Available only if the control is part of a control array.

object[(number)].**Index**

The **Index** Property syntax has these parts:

Part	Description
<i>Object</i>	Any object expression that evaluates
Number	A numeric expression that evaluates to an integer that identifies an individual control within a control array.

Control Scenario: Adding and Deleting Controls in a Control Array

The control array example demonstrates how controls --- in this case, option buttons --- are added and deleted at run time. The example allows the user to use option buttons that change the background color of a picture box.

Start with a form, and then draw a picture box, a label, two option buttons, and three command buttons.

Events in the Control Array Application

Next, you need to add the event procedures for the option buttons and command buttons. Start by adding the form declaration:

```
Dim MaxId As Integer
```

The Click event procedure is shared by all the option buttons:

```
Private sub optButton_Click(Index As Integer)
picDisplay.BackColor=QBColor(Index+1)
End Sub
```

Name option buttons are added by the Click event procedure for the Add command button. In this example, the code checks that no more than ten option buttons are loaded before the Load statement is executed. Once a control is loaded, its Visible property must be set to True.

```
Private Sub cmdAdd_Click()
If MaxId=0 Then MaxId=1 'Set total option buttons
If MaxId> 8 Then Exit Sub 'Only ten buttons
                        'allowed.
MaxId = MaxId + 1      'Increment button count.
Load optButton(MaxId) 'Create new button.
optButton(0).SetFocus 'Reset button selection
'Set new button under previous button.
optButton(MaxId).Top=optButton(MaxId -1)._
Top +400
optButton(MaxId).Visible=True 'Display new Button.
optButton(MaxId).Caption="Option"&MaxId + 1
End Sub
```

Option buttons are removed by the Click Event procedure for the delete command button

```
Private Sub cmddelete_Click()
```

```
    IF MaxId<= 1 'Keep first two buttons.
```

```
    UnLoadoptButton (MaxId) 'Delete last Buttons.
```

```
    MaxId = MaxId - 1 'Decrement button count.
```

```
    optButton(0).SetFocus 'Reset button Selection.
```

```
End Sub
```

The Close button Click Event ends the application:

```
Private Sub cmdClose_Click()
```

```
End Sub
```

3. Discuss the DoEvent Function and SubMain with examples?(a14)

DO EVENTS

DoEvents Function

- Yields execution so that the operating system can process other events.

Syntax:

DoEvents()

DoEvents Function Example

This example uses the **DoEvents** function to cause execution to yield to the operating system once every 1000 iterations of the loop. **DoEvents** returns the number of open Visual Basic forms, but only when the host application is Visual Basic.

‘Create a variable to hold number of Visual Basic forms loaded

‘And visible

```
Dim I, OpenForms
```

```
For I=1 to 150000 'Start loop
```

```
    If I Mod 1000=0 Then 'If loop has repeated 1000 times.
```

```
        OpenForms=DoEvents 'Yield to operating System.
```

```
    End If
```

```
Next I 'Increment loop counter.
```

Although Timer Events are the best tool for background processing, particularly for very long tasks, the DoEvents function provides a convenient way to allow a task to be canceled.

For example, the following code shows a “Process” button that changes to a “Cancel” button when it is clicked. Clicking it again interrupts the task it is performing.

‘The original caption for this button is
“Process”.

```
Private Sub Command1_Click()
```

```
    ‘Static variables are shared by all instances  
    ‘of a procedure
```

```
    Static blnProcessing As Boolean
```

```
    Dim lngCt As Long
```

```
    Dim intYieldCt As Integer
```

```
    Dim dblDummy As Double
```

```
    ‘When the button is clicked, test whether it’s  
    ‘already processing.
```

```
    If blnProcessing Then
```

```
        ‘If Processing is in progress, cancel it.
```

```
        blnProcessing = False
```

```
    Else
```

```
        Command1.Caption = “Cancel”
```

```
        blnProcessing = True
```

```
        lngCt = 0
```

```
        ‘Perform a million floating-Piont
```

```
        ‘multiplications. After every
```

```
        ‘thousand, check for cancellation.
```

```
        Do While blnProcessingAnd (lngCt< 1000000)
```

```
            For intYieldCt=1 to 1000
```

```
                lngCt = lngCt+1
```

```
                dblDummy = lngct * 3.14159
```

```
            Next intYieldCt
```

‘The DoEvents statement allows other
‘events to occur, including pressing this
‘button a second time.

Doevents

Loop

blnProcessing = False

Command1.Caption = “Process”

MsgBox lngCt & “ multiplications were performed”

End If

End Sub

DoEvents switches control to the operating-environment kernel.

Control returns to your application as soon as all other applications in the environment have had a chance to respond to pending events

This doesn’t cause the current application to give up the focus, but it does enable background events to be processed.

The results of this yielding may not always be what you expect.

For example, the following Click-event code wait until ten seconds after the button was clicked and then displays a message. If the button is clicked while it is already waiting, the clicks will be finished in reverse order.

Private Sub Command2_Click()

Static intClick As integer

Dim intClickNumber As Integer

Dim dblEndTime As Double

‘Each time the button is clicked,

‘give it a unique number.

intClick = intClick + 1

intClickNumber = intClick

‘wait for ten seconds

dblEndTime = Timer + 10#

Do While dblEndTime > Timer

‘Do nothing but allow other

‘applications to process

‘their events

DoEvents

Loop

MsgBox”Click” &intClickNumber& “is Finished”

End Sub

USE OF SUBMAIN()

Starting Without a Startup Form

Sometimes you might want your application to start without any form initially loaded

For Example, you might want to execute code that loads a data file and then displays one of several different forms depending on what in the data file. You can do this by creating a Sub procedure called Main in a Standard module, as in the following example:

Sub Main()

Dim intStatus As Integer

‘ Call a function procedure to check user status.

intStatus = GetUserStatus

‘ Show a startup form based on status.

If intStatus = 1 Then

frmMain.Show

Else

frmPassword.Show

End If

This procedure must be a Sub procedure, and it cannot be in a form module

To **set the Sub Main procedure as in the startup object**, from the **Project menu**, choose Project Properties, select the **General** tab, and select **Sub Main** from the StartupObject box.

4. What is the use of combo box? Explain(a12)

Combo Box Control

The **Combo Box** control combines the features of a **TextBox control** and a **ListBox control**. users can enter information in the text box portion or select an item from the list box portion of the control.

Syntax

Combo Box

This example uses the **AddItem** method to add 100 items to a list box.

To try this example, paste the code into the Declarations section of a form with a **Listbox** control named list1, and then press F5 and click form.

```
Private Sub Form_Click()  
    Dim Entry, I, Msg 'Declare variables.  
    Msg = "choose Ok to add 100 items to your list box."  
    MsgBoxMsg 'Display Message.'  
    For I = 1 To 100 'Count from 1 to 100'  
        Entry = "Entry "&I 'Create entry.  
        List1.AddItem Entry 'Add the Entry.  
    Next I  
    MsgBox = "Choose Ok to remove every other entry."  
    MsgBoxMsg 'Display message.  
    For I = 1 To 50 'Determine how to List1.RemoveItem I 'remove every other  
    Next I 'Item.  
    Msg = "Choose Ok to remove all items from the list box."  
    MsgBoxMsg 'Display message.  
    List1.Clear 'Clear list box.  
End Sub
```

DataCombo Control

The **DataCombo** control is a data-bound combo box that is automatically populated from a field in an attached data source, and optionally updates a field in a related table of another data source.

Syntax

DataCombo

Remarks

The **DataCombo**control is code-compatible with the **DBCombo**control. However the **DataCombo**control is optimized to work with ActiveX Data Objects (ADO)

DBCombo Control

The **DBCombo**control is a data bound combo box with a drop-down list box which is automatically populated from a field in an attached **Data** control, and optionally updates a field in a related table of another **Data** control. The text box portion of **DBCombo**can be used to edit the selected field.

Syntax

DBCombo

Remarks

The **DBCombo**control differs from the standard **ComboBox**control. While the **ComboBox**control list is filled using the **AddItem**method, the **DBCombo**control is automatically filled with data from a field in the **Recordset** Object of a data control to which it is attached.

The Standard **ComboBox**control must be populated manually by using the **AddItem**method. In addition, the **DBCombo**control has the ability to update a field within a related **Recordset**object which ,may reside in a different **Data** control.

ImageCombo control

The **ImageCombo**control is a picture enabled version of the standard Windows combo box. Each item in the list portion of the control can have a picture assigned to it.

Syntax

ImageCombo

Remarks

With a**ImageCombo**control, you can display a list of items that includes pictures. Each item can have its own picture, or you can use the same picture for multiple list item.

Add Method Example

This Example shows how to add a **ComboItem**using the **Add** Method, and use the reference returned to change the properties of the new object.

Dim ci As Integer

```
Set ci = ImageCombo1.ComboItems.  
Add(1, "signal1", "Signal", "RedLight",  
"GreenLight", 0)  
ci.ToolTipText = "Traffic Light"  
ci.Indentation = 2
```

5. Explain the four properties of flex grid?(n13)

Four properties of flex grid

1. Cols, Rows

This properties determine the number of rows & columns in the grid.

They must be integer and the syntax is

GridName.cols = Num of cols%

GridName.Rows = Num of Rows%

Eg ;-frmDisplay.cols = 10

2. Col, Row

This property is return the row & column for the currently selected cell inside grid.

These properties are only available at run time.

Row = 0

Column = 0

Gives the location cell (0,0)

3. Col Position, Row Position

It is used to move through the whole rows & columns occasionally use.

Syntax

GridName col position(number)[=value]

GridName Row position(number)[=value]

4. colwidth,Row height

It specify the width of a specified column or height of a specific row

Syntax

GridName col width (colNumber%) = width%

GridName Row height (RowNumber%) = height%

6. Explain Insertion sort?(n12)

Insertion Sort

Check the new entry to the first entry.

Check backward if it fails move the whole entry down by one. So you have to move all the entry.

Moving a “hole” along as it move to the list

Eg: Private Sub Insertion Sort (A\$C)

Dim no of entries%, I%, J%, Temp\$

No of entries% = UBound(A\$)

For I% = 2 to no of entries%

Temp\$ = A\$(I%)

For J% = I% -1 To step - 1

If(Temp \$ > = A\$(J%)) Then exit for

A\$(J%+1) = A\$(J%)

Next J%

A\$(J%+1) = Temp\$

Next I%

End Sub

If most of the time it moves objet ineffectively.

7. Explain Shell Sort?(a11)

It is the combination of insertion and ripple sort.

Merge the smaller list & Break up the list into three new list.

The insertion sort on this three smaller list and merge again. Now the resulting list is very close to being sorted.

A Final Sort finishes the process

Eg: Sub Shell Sort (A\$C)

Dim no of entries%, Increment%, J%, Temp\$

No of entries% = UBound(A\$)

Increment% = no of entries%

Do Until Increment% 1

```

For I% = Increment% + 1 To No of entries
Temp$ = A$(I%)
For J% = I% - Increment% To 1 Step - Increment%
If(Temp > = A$(J%)) Then exit for
A$(J%+Increment%) = A$(J%)
Next J%
A$(J%+Increment%) = Temp$
Next I%
End Sub

```

8. Explain Ripple sort?(n12)

Ripple Sort

Sort with the First Entry.

Look at the remaining entry 1 by 1 whenever we find a smaller entry swap with first entry always hold the candidate for the smallest entry.

Look at the list by sorting with the 2nd entry and look at the removing entry 3,4 and so on.

Continuous step 3 until all items are worked through.

Disadvantage

The most of the time the comparison in the various loops of wasted.

Eg: Private Sub Ripple Sort (A\$C)

```
Dim no of entries%, No of time%, I%, J%, Temp$
```

```
No of entries% = UBound(A$)
```

```
No of Time% = No of entries% - 1
```

```
For I% = LBound (A$) to no of times%
```

```
For J% = I% + 1 To No of entries%
```

```
If(A$(J%) < A$(I%)) Then Swap A$(J%) A$(I%)
```

```
Next J
```

```
Next I
```

```
End Sub
```

```
Sub swap( x as string y as string)
```

```
Dim Temp as string
```

```
Temp = x
```

```
X = y
```

```
Y = Temp
```

```
End Sub
```

Steps for Testing Sort:

Generation of random test of string.

To create a list of four letter words Form_Click()

Eg: Dim B \$ (1000)

```
Dim I as Integer, RndInt 1 as Integer
```

```
RndInt 2 as Integer
```

```
Randomize
```

```
For I = 1 to 100
```

```
RndInt 1 = Int (26 * Rnd (1) : RndInt 2 = Int (26 * Rnd (1))
```

```
B$ = chr$ (RndInt 1 + 65) + Chr$ (RndInt 2 + 65)
```

```
Next I
```

To Test Ripple sort

```
Ripple Sort B$(1)
```

```
Dim Temp as String
```

```
For I 1 to 100
```

```
Temp = $(I)$
```

```
Next I
```

```
Text1.Text = Temp
```

9. Explain Binary Search?(faq)

Binary Search

1. Divide the list in half
2. Determine whether the item is count or not.
3. Search at a first half of the list otherwise look at the 2nd half.
4. Go Back to step 1 as long as this are left to look at (Repeated Steps)

Eg: Suppose we have a 6000 items after completing step 4 go back step 1 with a list 0 – n items.

Syntax:

```
Private Sub Binary Search(X$(), Target $)
Dim Low as Integer, high as Integer, middle as Integer
Target position = 2
Low = LBound(X$)
High = UBound(X$)
Do
Middle = (Low + high)/2
Select case (X$(middle))
Case is > Target$
High = middle – 1
Case is < Target$
Low = middle + 1
End Select
Loop until(Target Position <> 0)
Print Target Position
End Sub
```

10. Explain Bubble Sort?(Faq)

The idea of bubble sort is the easiest of all: you constantly compare an entry with the one below it. This way, the smallest one “bubbles” to the top.

```
Private Sub BubbleSort 9A$ (())
‘assume slist starts with 1
N = UBound (A$)
For I = 2 To N
For J = N To I Step -1
If A$ 9J-1) A$ (J) Then
‘need to SWAP A$ (J-1), A$ (J)
Temp$ = A$ (J-1)
```

```

        A$ (J-1) = A$(J)
        A$ (J) = Temp$
    End If
Next J
Next I
End Sub

```

The problem is that bubble sort is almost always the slowest sort of all. Since it has few, if any, redeeming virtues, it should be placed, at the very least, with an insertion or ripple sort (which are just as easy to program) for small lists, and with one of the faster sorts, such as Shell sort, for longer lists. (The only place to use it is if the list to be sorted is already essentially sorted!)

11. Distinguish between fixed and dynamic array?(n14)

There are two kinds of arrays in Visual Basic: fixed arrays, where the memory allocation stays the same for the whole time the program runs, and dynamic arrays, where you can change size on the fly while the program is running. The advantage of a fixed array is that since memory is set aside at the beginning of the program, you run a much smaller risk of running out of memory while the program is running. The advantage of dynamic arrays is the flexibility they give. You can change their sizes in response to what the program encounters. Both fixed and dynamic arrays visible to the whole application, to a specific form or module, or only within a function or procedure

To set up a fixed one-dimensional array visible throughout the whole form, place a statement such as

```
Private Errand$(13)
```

Or

```
Private Errand (13) As String
```

In the declaration section of the form This line of code sets up a one-dimensional will be array for string that has 14 “pigeon holes” named Errand (0) through Errand (13) that will be visible to every function or procedure on that form. The items would be stored in the Errands (0) through Errand (13) pigeon holes.

To set up a dynamic one-dimensional array that will be visible to all functions or procedures in the form, place a statement such as

```
Private Errand () As String
```

In the Declaration section of the form Notice the empty parentheses; this is what distinguishes a dynamic array from a fixed array. (One reason that using empty parentheses for a dynamic array is such a good notation is that it makes clear that the whole point of a dynamic array is to let the situation determine the size.

12. Passing by reference & pass by value?(a13)

Pass by Reference

When an argument is passed by reference any changes to the parameter inside the procedure will change the value of the original argument.

Example:

```
Sub Triple(No as Integer)
    Num = 3 * Num
    Print Num
End Sub

Form_Load( )
    Dim Amt as Integer
    Me : Auto redraw = true
    Amt = 2
    Print Amt
    Triple (Amt)
    Print Amt
```

Pass by values

When an argument is passed the original value will be retained even after the program terminates.

In VB, the pass by value will be stored in a temporary id for all variables that are passed by value.

Example:

```
Sub Triple(No as Integer)
```

```
Num = 3 * Num
```

```
Print Num
```

```
End Sub
```

```
Form_Load( )
```

```
Dim Amt as Integer
```

```
Me : Auto redraw = true
```

```
Amt = 2
```

```
Print Amt
```

```
Triple (Amt)
```

```
Print Amt
```

TEN MARKS

1. Explain any ten general properties of the flex grid control?(a16)

General properties of flex grid

1. Cols, Rows

This properties determine the number of rows & columns in the grid.

They must be integer and the syntax is

GridName.cols = Num of cols%

GridName.Rows = Num of Rows%

Eg ;-frmDisplay.cols = 10

2. Col, Row

This property is return the row & column for the currently selected cell inside grid.

These properties are only available at run time.

Row = 0

Column = 0

Gives the location cell (0,0)

3. Col Position, Row Position

It is used to move through the whole rows & columns occasionally use.

Syntax

GridName col position(number)[=value]

GridName Row position(number)[=value]

4.Col width, Row height

It specify the width of a specified column or height of a specific row.

Syntax

GridName col width (colNumber%) = width%

GridName Row height (RowNumber%) = height%

5.Text, Text Matrix

i) Text

Sets or returns the text inside the current cell.

ii) Text matrix:-

Set or retrieve the text in an arbitrary cell without need to change the row & col properties.

Syntax:

Text matrix (row index, col index) = [String]

6. **Cell width, Cell Top, and Cell left, Cell height.**

This properties gives the size the location of a cell relative to it the container.

7. **Alignment**

Set the data alignment in a column.

Left – 0

Right – 1

Center – 2

8. **Fixed Cols, Fixed Rows & Fixed alignment**

By default zero rows and zero columns as a fixed and displayed in gray.

Fixed Alignment said to only the fixed row and fixed columns.

Syntax

GridName.fixedcols = Number of fixed cols%

GridName.fixedRows = Number of fixed Rows%

GridName.fixedAlignment = Setting Number%

9. **Gridlines, scroll bars**

This two properties control whether gridlines & scroll bar appear. (horizontal& vertical).

10. **Left col, Top Row**

It control which are left most column, highest row display from a grid.

2. What are list boxes and combo boxes? Differentiate them. Illustrate them with coding to add and remove items in list and combo boxes?

List Box

- List boxes when you have a fixed list of choice.
- VB automatically add vertical scroll bar whenever is too small for all items it contains

Combo Box

- It combines the feature of text box & list box.

- This control enables the user to select either by typing the text into combo box or by selecting an item from the list
- There are 2 types of combo box and which one get depends on the value of the style property
 1. If the value style property is said to the default value of zero
It will see the drop down, list of choice given in the box.
 2. If the value of the style property of a combo box is one.
The user see the combo box with the list already drop down
 - 0 – drop down
 - 1 – Single combo
 - 2 – drop down list

Generally,

Properties

- I. Add item
- II. Remove item
- III. List
- IV. Clear
- V. List Amount

Unsupported Event

- I. Double Click
- II. Drop Down
- III. Change Event

3. Explain the concept of Error Trapping?(a15)

ERROR TRAPPING

- Determines how errors are handled in the Visual Basic development environment and sets the default state of error trapping for all subsequent instances of Visual Basic.

- To Set the error trapping option for only the current session of Visual Basic can use the toggle command on the code window's shortcut menu
- **Break on All Errors** – Any error causes the projects to enter the break mode, whether or not an error handler is active and whether or not the code is in a class module.
- **Break in Class Module** – Any unhandled error produced in a class module causes the project to enter the break mode at the line of code in the class module which produced the error.
- **Break on Unhandled Errors** – If an Error handler is active, the error is trapped without entering break mode. If there is no active error handler, the error causes the project to enter break mode.

On Error Statement

Enables an error handling routine and specifies the location of the routine within a procedure;
can also be used to disable an error-handling routine

Syntax

On Error GoTo line

On Error Resume Text

On Error GoTo 0

The **On Error** statement syntax can have any of the following forms:

Statement	Description
On Error GoTo line	Enables the error handling routine that starts at line specified in the required line argument. The line argument is any line label or line number. If a runtime error occurs, control branches to line, making the error handler active. The specified line must be in the same procedure as the On Error statement; otherwise, a compile-time error occurs.

On Error Resume Next	Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred where execution continues. Use this form rather than On Error GoTo when accessing objects.
On Error GoTo 0	Disables any enabled error handler in the current procedure.

Remarks

If you don't use an **On Error** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

4. Explain anyone sorting method in VB with example.(n12,a15)

Ripple Sort

- Sort with the First Entry.
- Look at the remaining entry 1 by 1 whenever we find a smaller entry swap with first entry always hold the candidate for the smallest entry.
- Look at the list by sorting with the 2nd entry and look at the removing entry 3,4 and so on.
- Continuous step 3 until all items are worked through.

Disadvantage

The most of the time the comparison in the various loops of wasted.

Eg: Private Sub Ripple Sort (A\$C)

Dim no of entries%, No of time%, I%, J%, Temp\$

No of entries% = UBound(A\$)

No of Time% = No of entries% - 1

For I% = LBound (A\$) to no of times%

For J% = I% + 1 To No of entries%

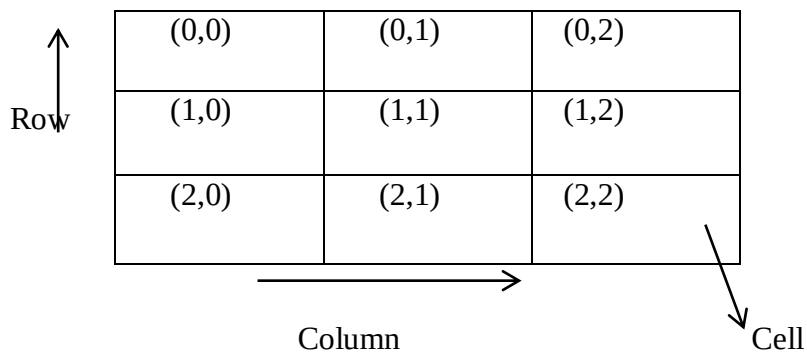
If(A\$(J%) < A\$(J%)) Then Swap A\$(J%) A\$(I%)

```
Next J
Next I
End Sub
```

```
Sub swap( x as string y as string)
Dim Temp as string
Temp = x
X = y
Y = Temp
End Sub
```

5. Explain Flex Grid Control?(n11)

Flex Grid Control



In ms flex grid control in vb is used to create applications that present information in rows and column. Display information in cell.

A cell is a location in ms flex grid at which row and column intersect cells can hold cells, bitmaps or icons and some cells holding text and other having graphics.

The user can select a cell at run time by clicking it or by using the arrow keys.

But cannot edit or alter the cell contents.

It used to display and operates on a tabular data.

It allows complete flexibility to short, merge, format tables containing string and picture.

When bound to a data control MS Flex grid read only data.

General properties of flex grid

1. Cols, Rows

This properties determine the number of rows & columns in the grid.

They must be integer and the syntax is

GridName.cols = Num of cols%

GridName.Rows = Num of Rows%

Eg ;-frmDisplay.cols = 10

2. Col, Row

This property is return the row & column for the currently selected cell inside grid.

These properties are only available at run time.

Row = 0

Column = 0

Gives the location cell (0,0)

3. Col Position, Row Position

It is used to move through the whole rows & columns occasionally use.

Syntax

GridName col position(number)[=value]

GridName Row position(number)[=value]

4. Col width, Row height

It specify the width of a specified column or height of a specific row.

Syntax

GridName col width (colNumber%) = width%

GridName Row height (RowNumber%) = height%

5. Text, Text Matrix

i) Text

Sets or returns the text inside the current cell.

ii) Text matrix:-

Set or retrieve the text in an orbitary cell without need to change the row & col proprerties.

Syntax:

Text matrix (row index, col index) = [String]

6. Cell width, Cell Top, and Cell left, Cell height.

This properties gives the size the location of a cell relative to it the container.

7. Alignment

Set the data alignment in a column.

Left – 0

Right – 1

Center – 2

8. Fixed Cols, Fixed Rows & Fixed alignment

By default zero rows and zero columns as a fixed and displayed in gray.

Fixed Alignment said to only the fixed row and fixed columns.

Syntax

GridName.fixedcols = Number of fixed cols%

GridName.fixedRows = Number of fixed Rows%

GridName.fixedAlignment = Setting Number%

9. Gridlines, scroll bars

This two properties control whether gridlines & scroll bar appear. (horizontal& vertical).

10. Left col, Top Row

It control which are left most column, highest row display from a grid.

6. What are the different types of arrays available? Explain.(a14)

ARRAYS:

An array is a set of sequentially indexed elements having the same intrinsic data type. Each element of an array has a unique identifying index number. Changes made to one element of an array don't affect the other elements.

Array Creation

- **Array Function**

The array function returns a variant containing an array.

Syntax

Array(arglist)

Where

The required arglist argument is a comma-delimited list of values that are assigned to the elements of the array contained within the Variant. If no arguments are specified, an array of zero length is created.

Example:

```
Dim myWeek, MyDay
```

```
MyWeek=Array("Mon","Tue","Wed","Thu","Fri","sat","Sun")
```

‘Return values assume lower bound set to 1 (using Option Base statement).

```
MyDay=MyWeek(2) ‘MyDay contains “Tue”.
```

```
MyDay=MyWeek(4) ‘MyDay contains “Thu”.
```

Declare and Initialize an Array

- **Dim Statement**

The Dim Statement declares variables and allocates storage space.

Syntax:

```
Dim varname[(subscripts)][As types].....
```

Where

1. Varname is the name of the variable.
2. Subscripts is the dimensions of an array variable; up to 60 multiple dimensions may be declared.
3. Type is the data type of the variable.

Example 1:

```
Dim Number As Integer
```

```
Dim a(9) As Integer
```

Where a is an array variable it can store 9 integer values in the array.

Finds the Limits of an Array

- **LBound()**

The Lbound function returns a long containing the smallest available subscript for the indicated dimension of an array.

Syntax:

LBound(arrayname[, dimension])

Where

1. Arrayname is the name of the array variable.
2. Dimension is a whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If the dimension is omitted, 1 is assumed.

Example:

This example uses LBound function to determine the smallest available subscript for the indicated dimension of an array.

Dim Lower

Dim MyArray(1 To 10, 5 To 15, 10 To 20) 'Declare array variables.

Dim AnyArray (10)

Lopwer=Lbound(MyArray, 1) 'Returns 1.

Lower=Lbound(MyArray, 3) 'Returns 10. 3 stands for the third dimension.

- **UBound**

The UBound function returns a long containing the largest available subscript for the indicated dimension of an array.

Syntax:

UBound(arrayname[, dimension])

Where

1. Arrayname is the name of the array variable.
2. Dimension is a whole number indicating which dimension's upper bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If dimension is omitted, 1 is assumed.

Example:

This example uses the UBound function to determine the largest available subscript for the indicated dimension of an array.

Dim Upper

Dim MyArray(1 To 10, 5 To 15, 10 To 20) 'Declare array variables.

Dim AnyArray(10)

Upper=UBound(myArray, 1) 'Returns 10.

Upper=UBound(MyArray, 3) 'Returns 20.

Upper=Ubound(AnyArray) 'Returns 10.

Reinitialize an array

➤ **Erase**

The Erase statement reinitializes the elements of fixed-size arrays and releases dynamic-array storage space.

Syntax:

Erase arraylist

Erase behaves differently depending on whether an array is fixed-size (ordinary) or dynamic. Erase recovers no memory for fixed-size arrays. Erase sets the elements of a fixed array as follows.

Types of Array	Effect of Erase on Fixed Array Elements
Fixed numeric array	Sets each element to zero.
Fixed string array (variable length)	Sets each element to a zero-length string(“”)
Fixed string array(fixed length)	Sets each element to zero
Fixed variant array	Sets each element to Empty
Array of user-defined types	Set each element as if it were a separate variable
Array of objects	Sets each elements to the special value Nothing

Example:

The following example declares an array using the function and finds out the sum of the values in the array.

General Declaration

Dim a(10) as integer

Private sub cmdget_Click()

Dim i as integer

For i = 0 to 9

a(i) = val(inputbox(“Enter a no.”))

```

        next i
End Sub

Private sub cmdadd_Click()

    Dim i as integer
    Dim i as integer
    For i= 0 to 9
        s=s+a(i)
    next i
    msgbox "The sum of the array elements:&str(s)
End Sub

```

Fixed and Dynamic Arrays:

There are two kinds of arrays in Visual Basic: fixed arrays, where the memory allocation stays the same for the whole time the program runs, and dynamic arrays, where you can change size on the fly while the program is running. The advantage of a fixed array is that since memory is set aside at the beginning of the program, you run a much smaller risk of running out of memory while the program is running. The advantage of dynamic arrays is the flexibility they give. You can change their sizes in response to what the program encounters. Both fixed and dynamic arrays visible to the whole application, to a specific form or module, or only within a function or procedure

To set up a fixed one-dimensional array visible throughout the whole form, place a statement such as

```

Private Errand$(13)
Or
Private Errand (13) As String

```

In the declaration section of the form This line of code sets up a one-dimensional array for string that has 14 “pigeon holes” named Errand (0) through Errand (13) that will be visible to

every function or procedure on that form. The items would be stored in the Errands (0) through Errand (13) pigeon holes.

To set up a dynamic one-dimensional array that will be visible to all functions or procedures in the form, place a statement such as

```
Private Errand () As String
```

In the Declaration section of the form Notice the empty parentheses; this is what distinguishes a dynamic array from a fixed array. (One reason that using empty parentheses for a dynamic array is such a good notation is that it makes clear that the whole point of a dynamic array is to let the situation determine the size.

Static Arrays:

Static variables, it is occasionally useful to have an array whose contents remain the same no matter how often the function or procedure is used. By analogy with static variables, this kind of array is usually called a static array. To set up a static array inside a function or procedure, you use the Static keyword:

```
Private Function FunctionName () As String
```

```
    Static Errand (13) As String
```

```
End Sub
```

As with static variables, the information you store in a static array remains intact.

Multidimensional Array:

Arrays with more than one dimension; they're usually called multidimensional arrays. Just as list of data lead to a single subscript (one-dimensional arrays), tables of data lead to double subscripts (two-dimensional).

Two dimensional Declaration

```
Dim a(10, 10) As Integer
```

Dim b(10, 10) As Integer

Dim c(10, 10) As Integer

Here there three two dimensional array we can store 20 integer values in each array.

For example, suppose you want to store a multiplication table in memory—as a table. You could do this with the following code:

```
Static MultTable (1 To 12, 1 To 12) As Integer
```

```
Dim I As Integer, J As Integer
```

```
For I=1 To 12
```

```
    For J= 1 To 12
```

```
        MultTable (I,J) = I*J
```

```
    Next J
```

```
Next I
```

To compute the number of items in a multidimensional array, multiply the number of entries. The dimension statement for the two-dimensional array that I used here sets aside 144 elements.

The convention is to refer to the first entry as giving the number of rows and the second as giving the number of columns. Following this convention, you would describe the fragment for multiplication table above as filling an entire row, column by column, before moving to the next row.

Visual Basic allows you up to 60 dimensions with the Dim statement and 8 with the ReDim statement. A statement like

```
Dim LargeArray %( 2, 2, 2, 2, 2, 2, 2, 2)
```

Index ranges in multidimensional arrays; you obviously need versions of LBound and UBound. The commands

```
LBound (NameOfArray, I)
```

```
UBound (NameOfArray, I)
```

UNIT-IV

TWO MARKS

1. What is MDI? Give the expansion for MDI.

The Multiple Document Interface (MDI) was designed to simplify the exchange of information among documents and all under the same roof. With an MDI application, we can maintain multiple open windows, but not multiple copies of the application.

Data exchange is easier when we can view and compare many documents simultaneously. An MDI application must have at least two forms, the Parent Form and one or more Child Forms. Each of these Forms has certain properties. There can be many child forms contained within the Parent form. The Parent Form may not contain any controls. The parent Form can usually have its own menu.

2. List any four debugging tools.

We can use the Debug menu to gain access to the tools needed for debugging, and most of the tools needed for testing and debugging can be found on the debug toolbar. Choose View--->Toolbars--->Debug.

Usually the debugging tools are used when the program is in break mode. There are more than 10 debugging tools available in Visual Basic. The first three tools are the most common way of switching from project design mode to break mode.

Debugging Tools	Keyboard Shortcut	Function of the Tool
Run/Restart	F5/SHIFT+F5	Start the program as a new or restarts it
Break	CTRL+BREAKs	Giving interrupts to the program
End	-	Ends the program
Single step(step over)	F8	Moves through the program one statement at a time

3. List any 4 methods of common dialog box control.

A dialog box is nothing more than a form with controls on it. We can add a new form to any project (with Project | Add Form) and put any controls you want on the form. The

toolbox includes several file-related controls such as the File List Box control and the Directory List Box control that we can place on the form.

The Common Dialog Box control, however, makes sure that our dialog boxes that mimic those found in numerous other applications perform in the same way as those in other applications. We want to present the same set of common dialog boxes that users are used to seeing in other applications, or they will not adapt to our application as quickly as they might otherwise. In addition, using the Common Dialog Box control and setting a few properties is much simpler and gives us much more time than we would have if we created these dialog boxes from scratch. Four methods of common dialog box control as follows,

- Color dialog box control
- Font dialog box control
- Print dialog box control
- Help dialog box control

4. Define the term Debugging.

Visual Basic provides debugging tools to help the programmer track down such errors. Program errors often do not prevent it from running, but may cause unexpected results. The programmer's interface to the debugging tools is via three debug windows. Such as,

- Immediate window
- Local window
- Watch window

These windows can be accessed from the view menu or using the debug toolbar. Debugging is carried out when the application is in break mode. Break mode is entered by setting breakpoints, pressing Ctrl+Break keys.

5. Write short note on Immediate Window.

Most programming languages have a way to test program statements, procedures, and functions. Visual Basic uses an immediate window with the results of a simple calculation. We can use this window to test statements or to perform quick calculations when we are in break mode.

If we type `Print(?) 5 + 5` in the immediate window and press ENTER key, the Visual Basic quickly responds with the answer 10 on the same immediate window.

6. What is the use of shape control?

The shape control has 20 properties. Usually, we can change them dynamically with code while the application is running.

Setting of property	Shape effect	Border style	Fill style
0	Rectangle (default)	No border	Solid
1	Square	Solid(default)	Transparent(default)
2	Oval	Dashed line	Horizontal line
3	Circle	Dotted line	Vertical line
4	Rounded rectangle	Dash-dot line	Upward diagonal
5	Rounded square	Dash-dot-dot line	Downward diagonal
6	-	-	Cross
7	-	-	Diagonal cross

7. What is meant by PIXEL?

We know how colors are assigned, and we can change the scale of our screen as we like to fit. The syntax for this method is,

`PSet(Col, Row)[, Color Code]`

The PSet method doesn't support named arguments. Since, the color code is optional. We have to replace the parameters with the values we want. The values of the first entry are

column, and the second entry is row. After the VB process this statement, the pixel defined by user will be printed.

For example, in the ordinary scale, using the default size for a form, the line is PSet(3722, 2212).

8. How objects are defined in VB?

The objects that are usable in Visual Basic are collected into object(or type) libraries, for example, Visual Basic's object library, the Visual Basic for Applications object library, Excel's object library and so on.

An object library contains the information that Visual Basic needs to build instances of its objects, as well as information on the methods and properties of the object in the library. The object browser is the tool into the object libraries used in the project. To choose the object Browser,

- Select view/object browser
- Press F2
- Use the toolbar shortcut

9. How to add controls in a control array?

Place some same type of controls (command buttons) on the form. Make their name properties same and then a warning dialog box will come asking whether we want to create a control array, click yes. Or after placing a control on the form, copy that and paste on the form.

Set the index property of each control or we may not change as it is automatically set.

Now it is done. We are ready to use the control array in our code.

10. Write a note on line method.

The line control has 15 properties. Usually, we change them dynamically with code while the application is running. The most important properties for the line control at design time are the BorderWidth property and the BorderStyle property.

BorderWidth determines the thickness of the line. It is measured in pixels and can range from 0 to 8192. BorderStyle property of the Line control has six possible settings, but as before only the last five are really useful.

The most important properties at run time for the line control are the X1, X2, Y1 and Y2 properties. The X1 property sets horizontal left end of the line, X2 property sets horizontal right end of the line, Y1 property sets vertical top end of the line, Y2 property sets vertical bottom end of the line.

11. Write about single stepping tool.

Visual Basic let us execute one statement in our program at a time is called single stepping. We have to press repeatedly F8 button or select the single step tool on the Debug toolbar. It is one line of code at a time.

The first time we press F8 to start single stepping through our code, vb highlights the first executable statement of the program in the code window and places a yellow arrow in the leftmost column.

Debugging Tools	Keyboard Shortcut	Function of the Tool
Single step (step into)	F8	Moves through the program one statement at a time

FIVE MARKS

1. Explain the importance of Testing.

With TestComplete we can easily test Visual Basic applications. We can perform any kind of testing – unit, functional, regression and so on. The easiest way to create a test for the application is to [record](#) a keyword test or a script routine. When playing back the test, TestComplete repeats all the actions we performed during recording. If needed, we can also extend the recorded test manually using a number of specific features provided by TestComplete.

Creating and Recording Tests for VB Applications

With TestComplete, we can [record](#) and [play back](#) user actions in Visual Basic applications, or we can create tests manually from scratch. It is usually easier to record the test first and then modify and enhance the recorded test.

When we record a test, we interact with the tested Visual Basic application as an end-user would: navigate through the application's screens, fill out forms and so on. TestComplete captures all actions we performed in the application and adds them to the test.

A test consists of a sequence of operations that define various interactions with objects in the tested application. For example, in the sample test below we can see that item selection from a combo box is represented by the ClickItem operation, text input into text boxes - by the SetText operation, and so on.

Item	Operation	Value	Description
Orders			
OrderFrm			
ProductComboBox	ClickItem	"FamilyAlbum"	Selects the 'FamilyAlbum' item of the 'P...
CustomerName_Edit	Click	10, 11, ...	Clicks at point (10, 11) of the 'Custom...
CustomerName_Edit	SetText	"John Smith"	Enters the text 'John Smith' in the 'Cus...
OK_Button	ClickButton		Clicks the 'OK_Button' button.

A sample keyword test in Visual Basic

```
function Test1()
```

```
{
```

```
  var control;
```

```
  var control2;
```

```
  control = Aliases.Orders.OrderFrm1;
```

```
control.ProductComboBox.ClickItem("FamilyAlbum");
control2=control.CustomerName_Edit;
control2.Click(10,11);
control2.SetText("JohnSmith");
control.OK_Button.ClickButton();
}
```

The recorded tests can be modified and enhanced in a number of ways to create more flexible and efficient tests. For example, we can:

- Add new operations, reorder operations and modify their parameters.
- Delete or disable unneeded operations (for example, superfluous recorded operations).
- Insert [checkpoints](#) for verifying objects and values in the tested application.
- Create [data-driven tests](#) that run multiple test iterations using different sets of data.

Refer to the following topics to learn more about creating and enhancing tests:

Testing Types

There are two major test formats in TestComplete:

- [Keyword tests](#) - visually configured tests with grid-based editing interface. Best suited for novice users and those without programming knowledge.
- [Scripts](#) - code written in one of the supported scripting languages. May be better suited for advanced users.

We can select the test type when creating a test and cannot change it later. However, we can mix keyword tests and scripts within the same test project and call them from each other.

TestComplete also includes additional test types, such as [low-level procedures](#), [unit tests](#), [distributed tests](#) and so on. We can use them to address specific testing needs. However, most automation is typically done using keyword tests and scripts.

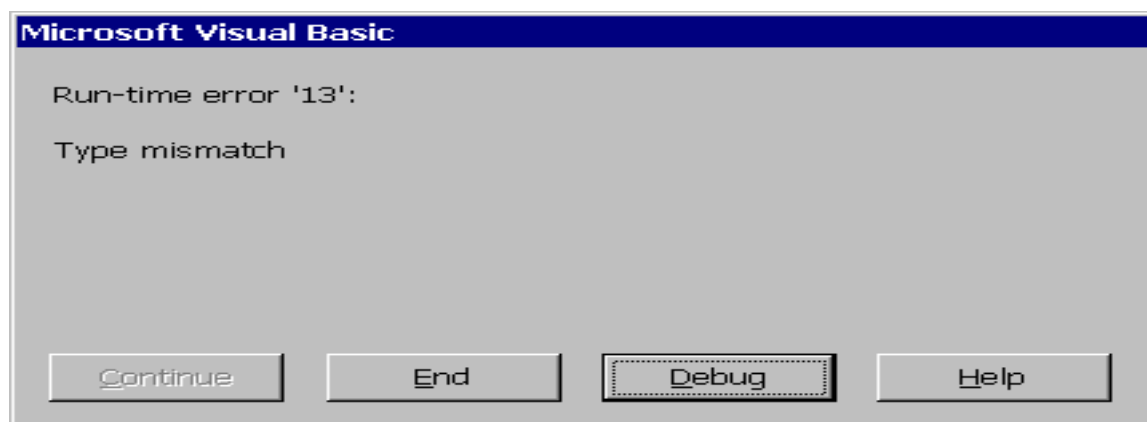
2. Explain the concept of Debugging.

Debugging

Visual Basic provides many tools to help analyze how the code operates. These debugging tools are particularly useful in locating the source of errors, or bugs. We can also use them to help us to learn how code written by other people works. When we write the code, we will never be able to create code that works first time round perfectly. Or if we do, if we change another section of code, this may well affect code we have already written which then does not work. There are generally three types of errors:

Run Time Errors

These occur while the application is running, when Visual Basic finds a statement that is impossible to carry out, such as multiplying the contents of TextBox1 by 10 when textbox1 contains 'Hello'. If this error occurs when running Visual Basic, a dialog is displayed like the one below. we are given the error number along with its message. We have the option to End the current program, Debug the program which pauses the code, or to get help on that error.



Visual Basic Error Dialog Box

If however an error occurs once the application has been made into an exe, the error number and message are given. But once the OK button has been pressed, the program closes.

Language Errors

These result from incorrectly structured code. For example, the Do statement without a corresponding Loop statement, or an opening quote for a string, but no closing one. These sorts of errors are picked out, either the moment we leave the line in the case of the missing quotes, or just before the code executes in the case of the Do...Loop example.

Logic Errors

These errors occur when the code we have written does not run the way we expected it. No error message is given as the code is using the correct syntax, and is not performing invalid operations, however it is not giving the expected result. This is by far the hardest error to track down. Only by going through the code very methodically and analyzing the results will be able to find the cause (or by very good luck, but don't rely on it!)

3. Explain how to create an object in VB with example.

Classes are the building blocks of an object-oriented application. A class is a programming construct that represents an object in the system. Designing an application requires designing the objects involved, the information contained by the objects, and what the objects can do. For more information on classes and object-oriented programming, see [Object-Oriented Programming in Visual Basic](#).

Class Modules

Visual Basic 6.0 provides support for defining classes through the class module. The definition of a class is stored in a special file type, the .cls file. One class is defined in each class module. Some of the definition of the class is embedded in the .cls file and only editable through the project system.

Global Classes

When we create a new class in Visual Basic 6.0, the allowable values of the Instancing property include GlobalSingleUse and GlobalMultiUse. These indicate that other components can invoke the properties and methods of the new class as if they were shared members. An instance of the class is implicitly created the first time one of its members is called.

Data Classes

Visual Basic 6.0 also supports Data Source and Complex Data Consumer classes.

Default Members

In Visual Basic 6.0, we can specify that any particular method or data member is the default member of a class.

Object Lifetime

Object lifetime is determined by the creation and termination of the object instance. The program determines the creation time of the objects it declares, but termination involves a more complex mechanism.

In Visual Basic 6.0, every object instance maintains a reference count. When the last reference to an instance is released and the count goes to zero, the object is terminated immediately.

Upgrade Options

The Upgrade Wizard creates a straightforward upgrade of class modules with minimal changes to the code. In addition to the changes listed above, Visual Basic programmers have several new constructs to choose from that is particularly relevant for object-oriented programming. A few are listed here.

- **Constructors** Constructors replace the use of **Class_Initialize** of Visual Basic 6.0.
- **Default indexed properties** Default indexed properties are used extensively throughout the .NET Framework for accessing collection items.
- **ReadOnly** In Visual Basic 6.0, a property is implicitly read-only if there is no **Get** procedure. Visual Basic 2008 requires the **ReadOnly** keyword to explicitly declare this behavior.
- **Late-binding** Since Visual Basic 6.0 provides late-binding by default, the most likely way to upgrade the code without errors is to set **Option Strict** to **Off**. Changing **Option Strict** to **on** enforces type-safety at compile time, and can expose subtle errors during compilation rather than at run time.
- **Method overloading** Method overloading reduces the number of named members of a class. This makes the class more readable and easier to program against.
- **File name** Visual Basic 6.0 requires the class name and file name to match. In Visual Basic 2008, the programmer controls both and can define several classes in one source file. How many classes belong in one file is typically covered by a coding standard.
- **ByRef** The default for parameters in Visual Basic 6.0 is **ByRef**, so the safest upgrade is **ByRef**.
- **Return keyword** In Visual Basic 6.0, the function name is used as a variable to return the value of the function. Visual Basic 2008 provides the explicit **Return** keyword for returning function values.

- **Operator overloading** You can now define the meaning of addition (+), subtraction (-), and other operators for the classes you define.
- **New operators** The += operator can shorten and simplify a line of code.
- **Inline definition of new values** Syntax in Visual Basic 2008 allows declaring and setting the value of a new variable all in one line of code. This means that you can enforce data constraints from the time the object is instantiated.

Rewriting the code is not without risks. Any time the code is changed it needs to be tested again.

4. Discuss briefly on common dialog boxes.

The Common Dialog Box control is perhaps the most powerful control available because of its capability to become one of several common dialog boxes we have surely seen in Windows applications. The "List Boxes and Data Lists," we learned that the Combo Box control is multifaceted because the control acts like one of three kinds of combo boxes, depending on the Styleproperty we set. The common dialog box goes far beyond the combo box's capability to take on different looks and purposes.

When we place a common dialog box on a form, we won't be able to resize it because the control, like the Timer control, will not appear on the form at runtime, at least not right away. The runtime code will have to display the common dialog box, and the look of the common dialog box displayed depends on what our code needs at the time.

If we are going to work with a data file, we could display the File Open dialog box.

The *quick viewer* is a Windows application that pops up when we right-click a filename. The quick viewer application displays the file in its native format without requiring us to own the original format's parent application (such as Lotus 1-2-3).

We've seen the File Open dialog box in many Windows applications. The dialog box works just as it does in Microsoft Word and other applications; the user can select a pathname or a filename, change the view by clicking one of the View buttons, open the file as read-only so no changes are

made to the file, and even select another drive or computer to choose from by clicking the drop-down list box.

The Common Dialog Box control creates the same File Open dialog box that we see in other applications, in every respect. For example, we can right-click any folder or file displayed inside the dialog box, and a pop-up menu appears with which we can rename the object, quick view the object (if the system contains a quick viewer application for the selected file's type), open the object, copy, cut, or perform a number of other tasks.

The Common Dialog Box control also presents a similar file-related dialog box, that produces a File Save dialog box. The user will also be familiar with the File Save dialog box. Our application can control the type of files displayed (by setting an appropriate filename extension type value) and can respond to the user's selection.

Why Common Dialog Boxes?

We do not have to use the Common Dialog Box control. A dialog box is nothing more than a form with controls on it. We can add a new form to any project (with Project | Add Form) and put any controls you want on the form. The toolbox includes several file-related controls such as the File List Box control and the Directory List Box control that we can place on the form.

The Common Dialog Box control, however, makes sure that our dialog boxes that mimic those found in numerous other applications perform in the same way as those in other applications. We want to present the same set of common dialog boxes that users are used to seeing in other applications, or they will not adapt to our application as quickly as they might otherwise. In addition, using the Common Dialog Box control and setting a few properties is much simpler and gives us much more time than we would have if we created these dialog boxes from scratch.

Figure 12.3 shows another common dialog box that the Common Dialog Box control can generate. Our applications can display the Color dialog box any time we want the user to select a color for an application such as a drawing application. Windows supports several million colors, and the Color dialog box shows only a few at a time. However, the user can locate the exact color desired from the Color dialog box or define his or her own custom colors by clicking the dialog box's Define Custom Colors button.

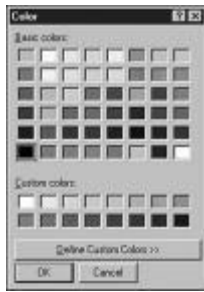


Figure 12.3 The Color dialog box is used by the user to pick a color.

The Font dialog box, shown in Figure 12.4, is another common dialog box that the Common Dialog Box control can display for us. When the application works with text, we should give which user the chance to select a font name or style. Our application cannot always know in advance which fonts the user will have on his system. The Font dialog box will, however, give the user a chance to select font information from his own computer. Once selected, the application can use the user's selection values to generate the text in the selected font.



Figure 12.4 The Font dialog box shows font information from the user's system.

The Print dialog box, shown in Figure 12.5, displays a common printer dialog box that we can display before printing any information to the printer. The dialog box returns information the user selects about the pending print job. As with the other controls, the Print dialog box only returns information to your application; it's up to your application to look at all relevant return information and respond accordingly. Therefore, if the user elects to print six copies, your code that follows the display of the Print dialog box must print six copies of the printed output.



Figure 12.5 The Print dialog box lets the user select a printer for output.

TIP

Want to add fax capabilities to your application? No problem; just display a Print dialog box. As long as the user has installed a Windows-aware fax driver, such as WinFax Pro or Microsoft Fax, the user can select that fax driver from the Print dialog box's Name list box. Your output, as long as you set your application's printer to the user's selected value, goes to the fax machine instead of to the printer.

CAUTION

The Print dialog box that appears on your system might be very different from the one shown in Figure 12.5. All the Common Dialog Box control's dialog boxes display available information based on their own computer settings. Therefore, your application will use these dialog boxes to let the user select from the properties of his own computer, depending on how those properties differ from yours.

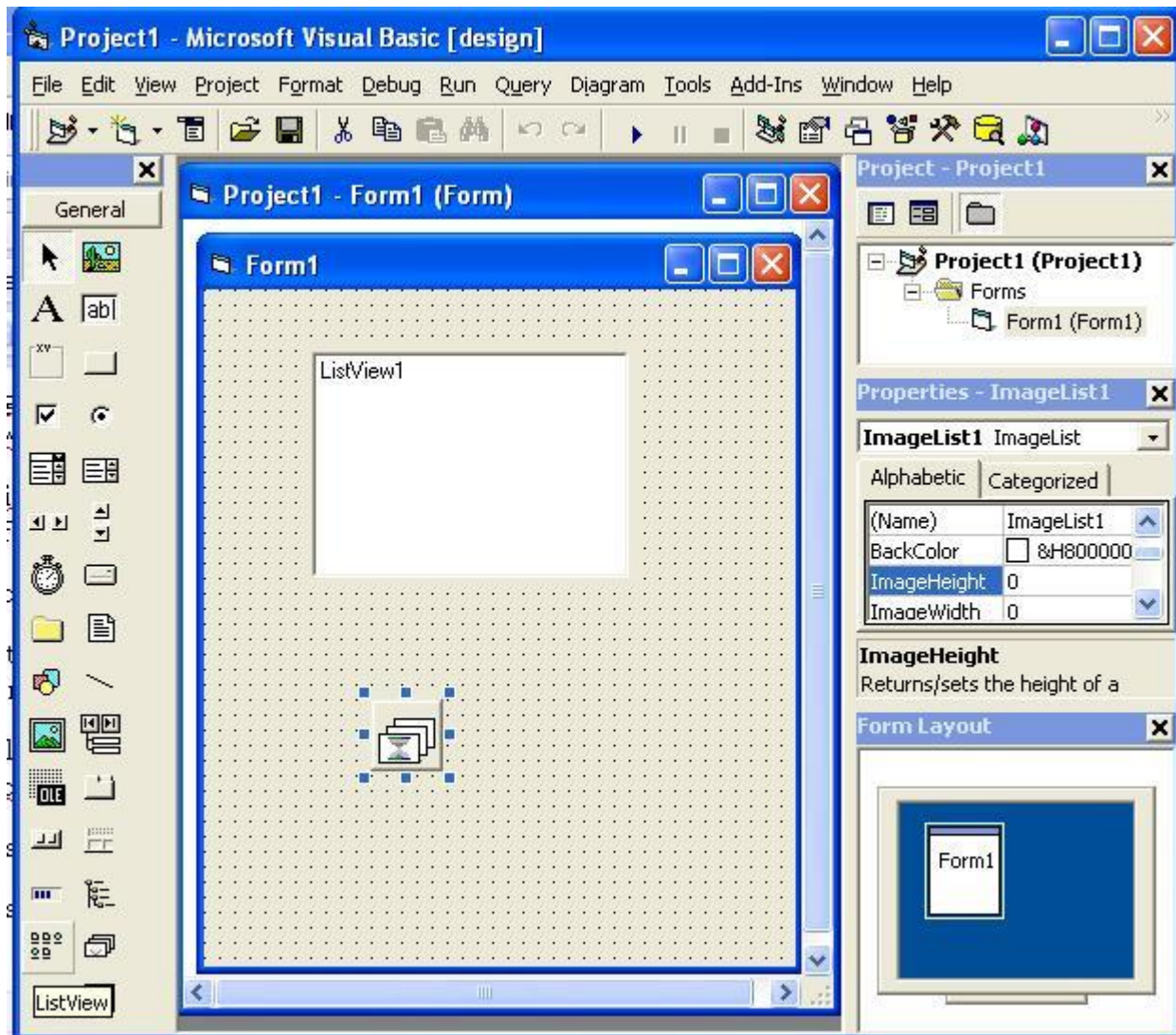
The final dialog box that the Common Dialog Box control can display is a Help window, such as the one in Figure 12.6. The Help window is linked to a help file that you generate. Generating help files is not a trivial task, as you'll see in Hour 23, "Distributing Your Applications." Therefore, this lesson doesn't show you how to connect a help file to your application. However, the Common Dialog Box control's Help dialog box can produce the front-end dialog box your user interacts with when you learn more about providing help in Hour 23.



5. Explain list view common control in detail.

An **ImageList** Control contains a collection of images that can be used by other Windows Common Control, Such as ListView,TreeView,TabStrip and ToolBar controls.It does not appear on the form at run time. It serves as a container for icon that are accessed by other control such as ListView,TreeView,TabStrip and ToolBar controls.

A ListView control displays data as ListItem objects. each ListItem object can have an optional icon associated with the Label of the object.



1. Click on "Start". Then select "All Programs, Microsoft Visual Studio 6.0 and **Microsoft Visual Basic 6.0.**".
2. Select "**Standard EXE**" from the list in the New Project dialog box and click on "**Open.**" Click on "Project" on the menu bar, then select "**Components**" from the drop-down menu.

3. Scroll down the list in the box until **Microsoft Windows Common Controls 6.0 (SP4)** is visible. Click on the checkbox to select the component and then click on “OK.” All components show in the Toolbox.
4. Choose the “**ListView1**” control from the list of controls in the toolbox. Place the control on the form in Visual Basic.
5. Choose the “**ImageList1**” control from the list of controls in the toolbox. Place the control on the form in Visual Basic.

6. Explain the line and shape controls.

The **Shape** control in Visual Basic 6.0 has no equivalent in Visual Basic 2008. However, you can use [Graphics](#) methods to achieve the same results, or you can use the Visual Basic Power Packs **OvalShape** or **RectangleShape** controls. These controls are available as add-ins.

Conceptual Differences

In Visual Basic 6.0, the **Shape** control provides an easy way to draw rectangles, circles, and other shapes on a form at design time. The **Shape** control is a *lightweight* control, which means that it does not have a Windows handle, also known as an **HWND**.

In Visual Basic 2008, there is no equivalent for the **Shape** control, and lightweight controls are no longer supported. However, there are ways to draw shapes on a form both at design time and at run time.

Note:

The Visual Basic Power Packs controls include **LineShape**, **OvalShape**, and **RectangleShape** controls that can be used to replace the **Line** and **Shape** controls. In addition to duplicating the behavior of the Visual Basic 6.0 **Line** and **Shape** controls, these controls add new capabilities, including gradient fills, run time selection, and run time events.

You can download the Visual Basic Power Packs from the [Microsoft Visual Basic 2005 Power Packs](#) page on the MSDN Web site.

At design time, you can draw a square or rectangular shape on a form by adding a [Label](#) control and setting the [Text](#) property to an empty string, the [BorderStyle](#) property to FixedSingle, and the [BackColor](#), [Width](#), and [Height](#) to the desired color and dimensions.

At run time, you can draw rectangles, ellipses, and complex shapes in a form's [Paint](#) event handler by creating a new object from the [Graphics](#) class and calling its methods.

In Visual Basic 6.0, you can use a **Shape** control to draw a shape on top of a container control such as a **PictureBox** or **Framecontrol** by adding a **Shape** control to the container.

In Visual Basic 2008, you can achieve the same effect by calling the [Graphics](#) methods in the **Paint** event of the container control.

Code Changes for the Shape Control

The following code examples illustrate the differences in coding techniques between Visual Basic 6.0 and Visual Basic 2008.

Code Changes for Drawing Rectangular Shapes

The following code demonstrates drawing a filled rectangle on a form at run time. In the Visual Basic 6.0 example, the **Shapecontrol** is used; assume that a **Line** control was added at design time. The Visual Basic 2008 example demonstrates two different methods: using a [Label](#) control and using [Graphics](#) methods.

Code Changes for Drawing a Circle

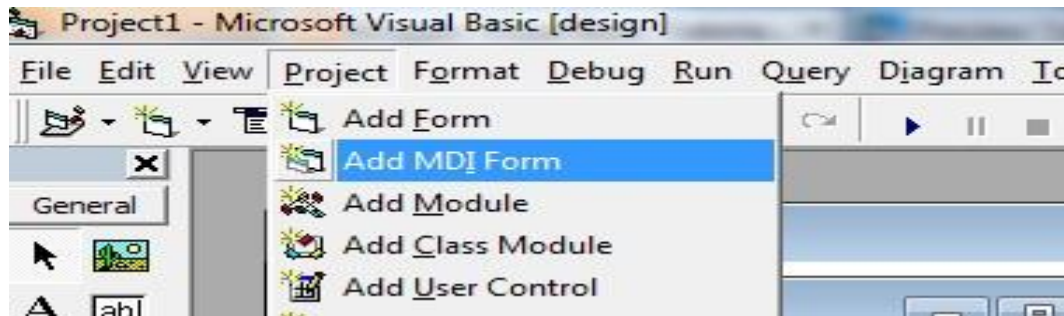
The following code demonstrates drawing a circle on a form at run time. In the Visual Basic 6.0 example, the **Shape** control is used; assume that a **Shape** control was added at design time. The Visual Basic 2008 example uses [Graphics](#) methods.

TEN MARKS

1. Discuss the MDI forms with example.

MDI stands for Multiple Document Interface. You have probably seen many MDI applications. When you want to handle multiple documents, MDI forms are useful in a Windows program.

How to add an MDI form to the current project?



Project -> Add MDI form. Click Project from the menu bar, and click Add MDI form. It's simple! Remember, a project can have only one MDI form.

Restrictions of the MDI form

1. You can have only one MDI form per project.
2. You can't place most controls on an MDI form. The only controls that can be placed on the surface of the MDI form are Menus, Timer, CommonDialog, PictureBox, ToolBar, and StatusBar.

These restrictions are there because MDI forms are the special type of forms, especially used to handle multiple child forms.

How does the MDI form work?

There can be only one MDI parent form in a project with one or more MDI child forms (or simply child forms).

- **MDI child form:** To add a child form, you have to add a regular form, and set the MDIchild property to True. You can have many child forms and can show an MDI child form using the Show method.
- **AutoShowChildren property of an MDI form:** The default value of the AutoShowChildren property is True. When it is True, the MDI child forms are displayed once they are loaded. When the value is False only then you can keep it hidden after loading, otherwise not.

- **Restrictions of the MDI child forms:**

1. You can't display an MDI child form outside its parent.
2. You can't display a menu bar on the MDI child form.

Now coming to the point - how the MDI form works. The parent form contains a menu bar on top of it. From there, the user opens or creates a new document. In this way, the user accomplishes his/her work in one or multiple documents, then saves and closes the document (form). You can create instances of a single form in the code using the Set keyword (Using the object variables).

'Inside the MDIForm module

```
Private Sub mnuFileNew_Click()  
    Dim frm As New Form1  
    frm.Show  
End Sub
```

- **ActiveForm property:** This is the Object type read-only property of the MDI form. You can apply this property to one of the children. For example, you can close the active form using this property from the Close menu command of the menu bar.

'In the MDI form

```
Private Sub mnuFileClose_Click()  
    If Not (ActiveForm Is Nothing) Then Unload ActiveForm  
End Sub
```

Note that '(ActiveForm Is Nothing)' represents that there is no active form. The 'Not' keyword before it negates the value.

2. What are the common controls found in Visual Basic? Explain them.

Visual Basic (VB) has two types of Controls: Intrinsic Controls and Active Controls. These controls represent all the Graphics User Interfaces, as well as Operating Systems resources, available to programmers.

Intrinsic Controls

There are 20 controls which are built in to VB. These controls are called intrinsic controls. Every one of the intrinsic controls are available to every VB program you write. When you create a "New" VB project, all 20 of the intrinsic controls will show up in the Toolbox. This is true f

here are many free controls which can be used freely at design-time. However, commercial controls require that you install them using a password before you can use them within your project at design time. If you simply copy an OCX to your system and register it with REGSVR32, don't be surprised if you get an error message when you try to put the OCX on a form!

Intrinsic Control List

The following list shows the 20 intrinsic controls that come with all version of VB. I have a separate tutorial section to discuss them further:

checkbox	frame	OLE Container
combobox	horizontal scrollbar	optionbutton
commandbutton	vertical scrollbar	picturebox
data control	image control	shape control
dirlist	label	textbox
drivelist	line	timer
frame	listbox	

Sample ActiveX Control List

The next list shows some of the OCX controls.

Common Dialog	CMDLG32.OCX	automatically installed
DataComboBox	MSDATLST.OCX	automatically installed
DataList	MSDATLST.OCX	automatically installed
MSFlexGrid	MSFLXGRD.OCX	automatically installed

Visual Basic Intrinsic Controls

VB comes with 20 built-in controls.

Introduction

All controls are not equally useful. Some you will use on every application you write. Others you will use only when you have a special need for the features the controls offer.

When you start VB, you'll always find the intrinsic controls displayed in the toolbox. The controls are built-in to the VB files and do not exist in an external file (with a .OCX extension) the way the ActiveX controls do. In the toolbox, each of the controls has its own distinctive icon.

The Most Useful Intrinsic Controls

These nine intrinsic controls are pretty much used on every VB application I've written. Start your learning with these and then branch out. Further down on this page I have a brief comment on each of the controls.

Command Button ImageBox

CheckBox PictureBox

TextBox ListBox

Label ComboBox

Option Button

The Rest of the Intrinsic Controls

The other eleven intrinsic controls are also valuable but I find myself using these less often than the others. Also, you'll find that you use fewer of these within an application than you do of the nine that I listed as the most useful controls.

ADO Data Control Vertical Scroll

DirListBox Line

DriveListBox Shape

FileListBox OLEContainer

Frame Timer

Horizontal Scroll

Database Features

Several of the intrinsic controls can display or edit data directly out of a database. With VB, the ADO Data Control is used to access the database information and to distribute it on to the other intrinsic controls which can handle database information. VB uses the terminology "databound" to describe controls which have built-in features for handling database access.

Multi-processing: Timer Control

Timer is the most unusual of the intrinsic controls. By setting the .INTERVAL property this control will automatically create an event on a regular basis. No other control does this! Programmer can use it to create an action at a certain time and then turn the control off to prevent repeats. Using multiple timers, an application can be multi-tasking.

Visual Basic ActiveX Controls

As with the Intrinsic controls, not all of the ActiveX controls are equally useful. Some you will use on many applications but you will use others only when you have a special need for the features the controls offer.

When you start VB, **none** of the ActiveX controls are displayed in the Toolbox. Only the intrinsic controls are displayed, so you must manually insert the ActiveX controls into the Toolbox as you need them. To do so, right-mouse click on the toolbox and go to "Components", select the controls to put on the toolbox and press "OK". All of the ActiveX controls are contained within OCX files.

The Most Useful ActiveX Controls

These are the nine most used ActiveX controls:

Coolbar	Progress Bar
ImageList	Tabbed Dialog
Internet Transfer Control	Tab Strip
ListView control	ToolBar
Multimedia control	

The Specialty ActiveX Controls

Some of the VB controls are very excellent tools for special system resource control, such as Winsock in TCP/IP networking programming.

Communication control	RichTextBox
Data Repeater	SysInfo
MAPI	Winsock
MSChart	

More ActiveX Controls

Animation	MonthView
DateTimePicker	PictureClip
FlatScrollBar	Slider
ImageCombo	UpDown
MaskedEdit	

Database Controls and ODBC

Database controls make database access very easy for programmers to utilize existing databases in their applications.

Accessing

Databases

Remember that there are a variety of database products on the market, and that many vendors have their own proprietary formats in which the data is stored. In the case of Microsoft's Access database program, the method of accessing the database is through software which Microsoft calls its Jet Database Engine. Other vendors have their own access methods.

Many companies, including Microsoft, have provided their databases with the ability to be read by software which uses a standard database manipulation method known as **ODBC (Open Database Connectivity)**. While a programmer can choose to let the VB database-aware controls handle the details of accessing or manipulating databases (recordsets), it is also possible to use code to create and edit databases.

DAO(DataAccessObject) The first method VB offered for using ODBC to create/edit databases was DAO. It allowed the programmer to create a variety of objects in code, objects whose properties and methods could be used by programmers to create/edit the databases. This worked for Microsoft Access databases as well as any other ODBC-compatible database.

Microsoft also offered a second data access interface to ODBC which had some improved features over DAO, but which also lost some of the more general capabilities of DAO. This approach, called RDO (Remote Data Objects), primarily targeted the more complex aspects of stored procedures and result sets. It is generally not used by most programmers.

ADO(ActiveXDataObjects) Microsoft is in the process of replacing the ADO/RDO features with the newer ADO. ADO has a flatter model (meaning fewer objects) and is intended to be simpler

to use. In its HELP files, Microsoft states that "ADO definitely represents the future of data access technology"

For any programmer who simply wants to put a data control on a form and then display database information in other controls, the choice of ADO/DAO/RDO is not critical because the controls handle it for you. The advantage in manipulating databases in code is that you have a far greater degree of control over what actions are performed against the databases.

VBDataControl VB provides two controls which make the link to the database file and which create the recordset that is exposed to the rest of the controls in your application. The two are identical in concept but differ in the flexibility they offer to the programmer. I'll discuss the common features first and then cover some of the differences.

The concept of a data control is pretty simple. Programmers set the `.DataBaseName` property to tell the data control which database file to read. Then set a `.RecordSource` property to tell it which table within the file to make available as a recordset to other controls. You can also create recordsets which are combinations of one or more tables by setting the `.RecordSource` property to an SQL statement.

For any controls which support databases, you set the `.DataSource` property to the data control and the `.DataField` property to the specific field within the table/recordset that you want bound to the control.

At that point the bound controls will display the information from the chosen field of the recordset. If you make changes to the data then the database can be changed to reflect the changes by either using the data control to move to a new record or by executing the `.Refresh` method of the data control.

You can create entire database applications with essentially **no coding**, just the setting of control properties.

You can set the `.RecordSource` to a command which tells the data control to create a recordset that is a selected subset of the fields in one or more tables. And there is a lot more that you can do with SQL to reduce the amount of coding that you have to write yourself. Every VB programmer should be familiar with the use of SQL to create/manipulate recordsets. SQL can even be used to make mass changes to a recordset, including search and replace actions.

The two data controls :

- **Data control (intrinsic version)**

This is the original, intrinsic version of the control. It supports the JET database engine and can satisfy most beginners' needs.

- **ADO Data control**

This is the latest version of the data control, implemented as an ActiveX control. In line with keeping your distribution files to a minimum, use the intrinsic control whenever possible.

DataBoundControls Once you have the recordset available from the data control, you'll find VB to have a rich variety of controls which can access the recordset. There are 7 intrinsic controls and 16 ActiveX controls which you have available to use. Each has its place and over time you'll find yourself using most of them.

ComboBox

Here are the three versions that are available with VB, and some comments on how to decide which one to use.

- **ComboBox**

This is the original, intrinsic, version of the control. Use it whenever possible to keep down the size of your application.

- **DataComboBox**

This is Microsoft's most recent rendition of a combo control.

- **DBCombo**

Left over from VB5, you can forego using this one in favor of the newer DataCombo control.

List

Here are the three versions that are available with VB, and some comments on how to decide which one to use.

- **ListBox**

This is the original, intrinsic, version of the control.

- **DataList**

This is Microsoft's most recent rendition of a list control.

- **DBList**

Left over from VB5, you can forego using this one in favor of the newer DataList control.

Grid

There are actually four versions of a grid control that come with VB, but only three of them can be databound. Here are the three versions and some comments on how to decide which one to use or all versions of VB.

ActiveX Controls

Microsoft uses this terminology to refer to any control which exists as a separate file whose extension is OCX. To use an ActiveX control in your VB program, the control must be registered in the Windows Registry.

If you have a control which has not been registered by other software, you can register it yourself using the free program provided by Microsoft. The program, REGSVR32, comes with all version of Windows and is usually found in the Windows folder. To use it, simply type in:

REGSVR32 control.ocx

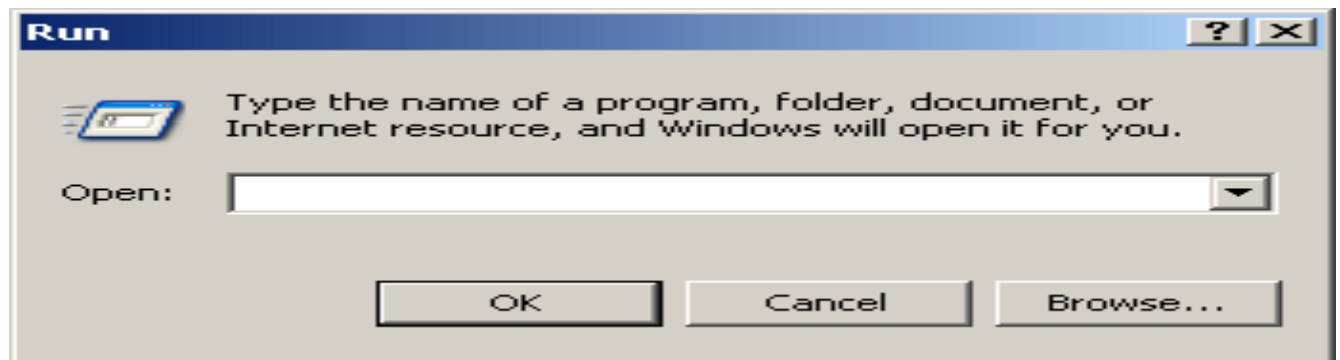
where the "control.ocx" is the filename of the control that you want to register.

IDE and Controls

Here's a key point to remember. Just because you register a control does not mean that you can use the control in one of your projects. Registration only assures you that the control can be used by an application at run-time. Whether or not a control can be used at design-time (within the IDE) depends on the control.

3. Explain the various Dialog Boxes used in VB with example coding.

A dialog box is a form defined with particular properties. Like a form, a dialog box is referred to as a container. Like a form, a dialog box is mostly used to host child controls, insuring the role of dialog between the user and the machine. Here is an example of a dialog box:



A dialog box has the following characteristics:

- The only system button it is equipped with is Close . As the only system button, this button allows the user to dismiss the dialog and ignore whatever the user would have done on the dialog box
- It cannot be minimized, maximized, or restored. A dialog box does not have any other system button but Close
- It is usually modal, in which case the user is not allowed to continue any other operation on the same application until the dialog box is dismissed
- It provides a way for the user to close or dismiss it

Introducing Dialog Boxes

1. Start Microsoft Visual Basic and create a new Windows Application named **SolasPropertyRental1**
2. From the Common Controls section of the Toolbox, click ListView and click the form
3. While the list view is still selected, in the Properties window, change the following characteristics
(Name): **LvwProperties**
View: **Details**
4. Still in the Properties window, click Columns and click its ellipsis button
5. In the ColumnHeader Collection Editor, click Add
6. In the right list, click Text and type **Property #**
7. Click Add.

In the right list, click Text and type **Property Type**

8. Click Add.

In the right list, click Text and type **Bedrooms**

9. Click Add.

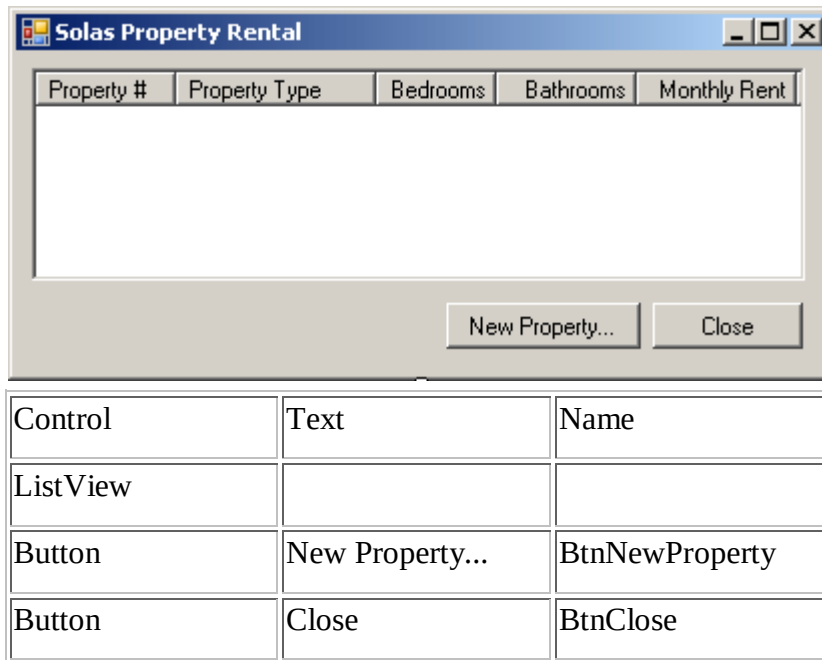
In the right list, click Text and type **Bathrooms**

10. Click Add.

In the right list, click Text and type **Monthly Rent**

11. Click OK

12. Complete the design of the form as follows:



The screenshot shows a Windows application window titled "Solas Property Rental". It features a table with five columns: "Property #", "Property Type", "Bedrooms", "Bathrooms", and "Monthly Rent". Below the table are two buttons: "New Property..." and "Close".

Control	Text	Name
ListView		
Button	New Property...	BtnNewProperty
Button	Close	BtnClose

13. To add another form to the project, on the main menu, click Project -> Add Windows Form...

14. In the middle list, make sure Windows Form is selected. Set the Name to **PropertyEditor** and click Add

Dialog Box Creation

To create a dialog box, you start with a form, which you can get by creating a Windows Application or deriving a class from Form. Here is an example:

Imports System.Drawing

Imports System.Windows.Forms

Module Exercise

Public Class Starter

Inherits Form

```

Dim components As System.ComponentModel.Container
Public Sub New()
    InitializeComponent()
End Sub
Public Sub InitializeComponent()
    Text = "Domain Configuration"
    Width = 320
    Height = 150
    Location = New Point(140, 100)
    StartPosition = FormStartPosition.CenterScreen
End Sub
End Class
Function Main() As Integer
    Dim frmStart As Starter = New Starter
    Application.Run(frmStart)
    Return 0
End Function
End Module

```

Characteristics of Dialog Boxes

The Border Style of a Dialog Box

There are a few actions you should perform on a form to transform it into a dialog box; but normally, these are only suggestions, not rules. Based on the Microsoft Windows design and standards, to create a dialog box, you should set a form's **FormBorderStyle** property to **FixedDialog**. Setting this property changes the borders of the form to the standard borders of a dialog box (the border of a dialog box is thinner than that of a regular form). You can set this characteristic in the Properties window or programmatically. Here is an example:

```

Public Sub InitializeComponent()
    Text = "Domain Configuration"

```

```
Width = 320
Height = 150
Location = New Point(140, 100)
StartPosition = FormStartPosition.CenterScreen
```

```
FormBorderStyle = Windows.Forms.FormBorderStyle.FixedDialog
```

```
End Sub
```

The Minimize and Maximize Boxes

Besides taking care of the border, you should also set both the **MinimizeBox** and the **MaximizeBox** properties to **False**. This causes the window to display only the system Close button. You can set these characteristics in the Properties window or programmatically. Here is an example:

```
Public Sub InitializeComponent()
    Text = "Domain Configuration"
    Width = 320
    Height = 150
    Location = New Point(140, 100)
    StartPosition = FormStartPosition.CenterScreen

    FormBorderStyle = Windows.Forms.FormBorderStyle.FixedDialog
    MinimizeBox = False
    MaximizeBox = False
End Sub
```

Configuring a Dialog Box

1. To transform the PropertyEditor form into a dialog box, in the Properties window, change its characteristics as follows:

FormBorderStyle: FixedDialog

StartPosition: CenterParent

MinimizeBox: False

MaximizeBox: False

ShowInTaskbar: False

2. Design the dialog box as follows:

Control	Text	Name
Label	Property Type:	
<u>TextBox</u>		TxtPropertyType
Button	Cancel	BtnCancel
Label	Bedrooms:	
TextBox		TxtBedrooms
Button	OK	BtnOK
Label	Bathrooms:	
TextBox		TxtBatrooms
Label	Monthly Rent:	
TextBox		TxtMonthlyRent

An Application With Various Forms or Dialog boxes

When you create a Windows Forms Application, the starting form is made available to you. If one form is not enough for your application, you can add as many as necessary. To add (or to create) a (new) form, you have various options:

- On the main menu, you can click Project -> Add New Item...
- On the main menu, you can click File -> Add New Item...

- In Solution Explorer, you can right-click the name of the project, position the mouse on Add, and click Add New Item...

In the Add New Item dialog box and in the middle section, click Window Form (.NET), provide a name in the Name edit box then click Open.

If your application is using various forms and you want to display a particular one at design time:

- In the Forms Designer, you can click the tab that corresponds to the desired form and that has [Design]
- On the main menu, you can click Window and click the name of the form in the list under Close All Documents
- In Solution Explorer, expand the Header Files node if necessary and double-click the name of the desired form that has the .h extension

If you visually add two (or more) forms to your application, you may need to link them, allow one to call the other. To do this, in the top section of the file, type **#include** followed by the name of the header file in which the form was defined. In the section where you want to access the form, declare a handle to the class of the form and use the **new** operator to allocate memory for it. To display the other form, you can call its **Show()** method.

4. How to work with graphics in VB? Explain with examples.

Working with graphics is easy in Visual Basic 6. VB6 gives you the flexibility and power to make graphical applications in easy steps. It is rich in graphics related features. The built-in methods, properties and events allow you to use these features most effectively.

In this tutorial, you will learn about various graphic methods, properties and techniques. The graphic methods and properties let you perform some graphical operations. More specifically, they allow you to draw points, lines, circles, rectangles, ellipses and other shapes. You will also learn how to display text and images. Finally, this tutorial introduces you to the Paint event.

This tutorial is for the beginner learners. So the concepts are presented in the simplest way possible. Code examples have been used wherever they became necessary.

Graphic methods

The graphic methods allow you to draw on the form and the PictureBox control. In Visual Basic 6, graphic methods are only supported by the form object and the PictureBox control. However, the later versions of Visual Basic allow you to use them with other objects and controls.

First of all, the graphic methods in this section are discussed only to give you a basic idea about them. Later in this tutorial, I will show you how to use them in your code to perform certain operations like printing text, drawing shapes etc.

The-common-graphic-methods-are-explained-below:

- **Print:** Print is the simplest graphic method in Visual Basic 6. This method has been used throughout the earlier versions of the language. It prints some text on the form or on the PictureBox control. It displays texts.
- **Cls:** The Cls method is another simple graphic method that is used to clear the surface of the form or the PictureBox control. If some texts are present, you can use the Cls method to remove the texts. It clears any drawing created by the graphic methods.
- **Point:** The Point method returns the color value from an image for a pixel at a particular point. This method is generally used to retrieve color values from bitmaps.
- **Refresh:** The refresh method redraws a control or object. In other words, it refreshes the control. Generally, controls are refreshed automatically most of the times. But in some cases, you need to refresh a control's appearance manually by explicitly invoking the Refresh method.
- **PSet:** The PSet method sets the color of a single pixel on the form. This method is used to draw points.
- **Line:** The Line method draws a line. Using the Line method, you can also draw other geometric shapes such as rectangle, triangle etc.
- **Circle:** The Circle method draws a circle. Using the Circle method, you can also draw other geometric shapes such as ellipses, arcs etc.
- **PaintPicture:** The PaintPicture method displays an image on the form at run-time.

- TextHeight: The TextHeight method returns the height of a string on the form at run-time.
- TextWidth: The TextWidth method returns the width of a string on the form at run-time.

The LoadPicture function

The LoadPicture function loads a picture to the form or to the PictureBox control. It sets the picture to the control in order to display it. The function takes the file path as an argument. The LoadPicture function allows you to set pictures at run-time.

Example:

```
Picture1 = LoadPicture("C:\MyPic.JPG")
```

Here, Picture1 is the PictureBox control. When this code is executed the picture is loaded and set to the PictureBox control. If Visual Basic cannot find the picture in the specified location, it throws a run-time error '53'.

The RGB function

The RGB function returns an integer, a color code which is used to set colors in Visual Basic code. The RGB color code is a combination of red, green and blue colors. Consider the following example to understand RGB function in VB6.

Example:

```
Form1.BackColor = RGB (120, 87, 55)
```

The RGB color is set as the background color of the form object. The first, second and the third arguments represent red, green and blue colors respectively. The color value is an integer. These values are in a range of 0 to 255. So you can use any value between 0 and 255 to obtain a color.

Graphic properties

The graphic properties are useful while working with the graphic methods. Some of the form's properties and some of the PictureBox's properties are the graphics properties. The common graphic properties are discussed in this section. You'll learn more about them using code examples later in this tutorial.

Consider-the-following-graphic-properties.

- DrawMode: The DrawMode property sets the mode of drawing for the appearance of output from the graphic methods. In the DrawMode property, you can choose from a variety of values.

- **DrawStyle:** The DrawStyle property sets the line style of any drawing from any graphic methods. It allows you to draw shapes of different line styles such as solid, dotted, dashed shapes etc.
- **DrawWidth:** The DrawWidth property sets the line width of any drawing from any graphic methods. While drawing shapes, you can control the thickness of the lines using this property.
- **FillColor:** The FillColor property is used to fill any shapes with a color. You may use the symbolic color constants to fill your shapes. You may also use the color codes as well as the RGB function.
- **FillStyle:** The FillStyle property lets you fill shapes in a particular filling style.
- **ForeColor:** The ForeColor property is used to set or return the foreground color.
- **AutoRedraw:** Set the AutoRedraw property to True to get a persistent graphics when you're calling the graphic methods from any event, but not from the Paint event.
- **ClipControls:** Set the ClipControls property to True to make the graphic methods repaint an object.
- **Picture:** The Picture property is used to set a picture. Pictures can be set both at design time and run-time.

Run-time graphic properties

CurrentX and CurrentY are the run-time properties which are used to set and return the position of a shape or point at run-time.

- **CurrentX:** The CurrentX property sets or returns the horizontal coordinate or X-coordinate of the current graphic position at run-time.
- **CurrentY:** The CurrentY property sets or returns the vertical coordinate or Y-coordinate of the current graphic position at run-time.

Printing text

In this section, I will show you how to use the Print method to display text on the form or the PictureBox control in various styles or colors.

Printing-on-the-form

The following code prints some text on the form.

Example:-

```
Private Sub cmdPrint_Click()
```

```
Form1.Print "Hello world"  
Form1.Print "Welcome to Visual Basic 6"  
Form1.Print "Visual Basic is awesome!"
```

End Sub

The above code can be written in the following way too.

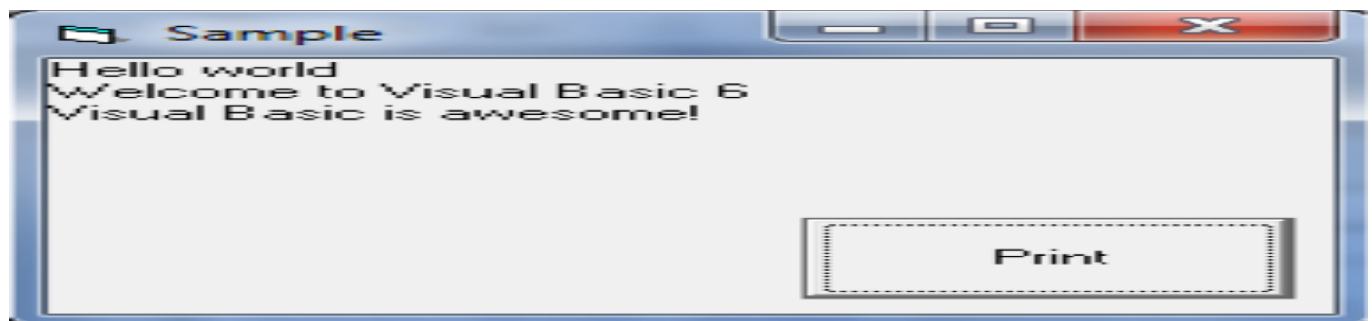
Example:-

```
Private Sub cmdPrint_Click()  
    Print "Hello world"  
    Print "Welcome to Visual Basic 6"  
    Print "Visual Basic is awesome!"
```

End Sub

In the above code, the Print method is called without the object name. Here 'Form1' is the object name. When you're writing code inside the form module, you may omit the form's name while invoking its methods.

Output of code:-



Specifying-printing-positions

You can change the printing position from (0, 0) to other. Examine the following code.

Example:-

```
Private Sub cmdPrint_Click()  
    CurrentX = 500  
    CurrentY = 1000  
    Print "This is a new text"
```

End Sub

Output of code:-



Printing-a-customized

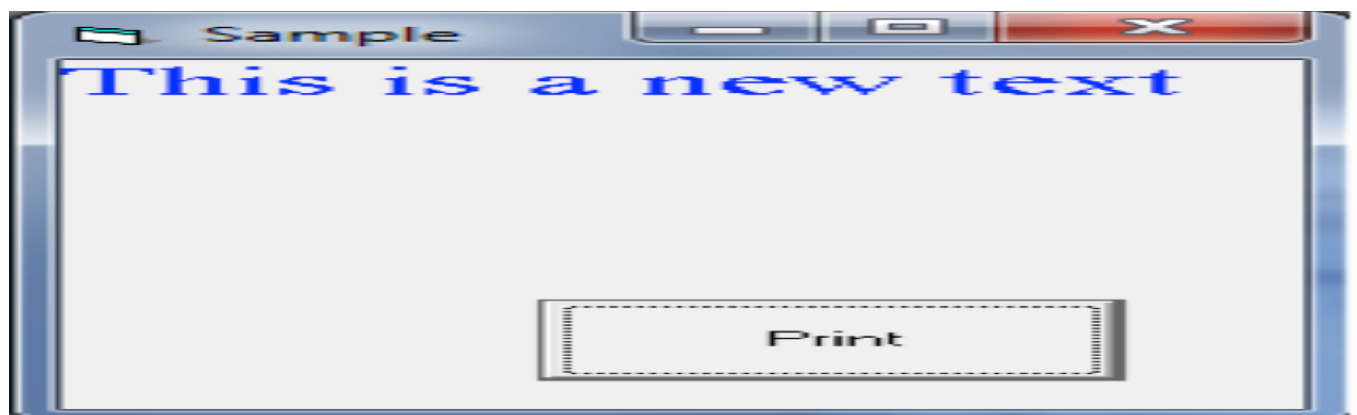
text

You can display text using different styles, sizes and colors. Consider the following code example.

Example:-

```
Private Sub cmdPrint_Click()  
    Form1.FontSize = 18  
    'Form1 object name is omitted  
    ForeColor = vbBlue  
    Font = "MS Serif"  
    Print "This is a new text"  
End Sub
```

Output of code:-



Drawing points

This section shows you how to draw points using the PSet method and how to use the Step keyword with the PSet method.

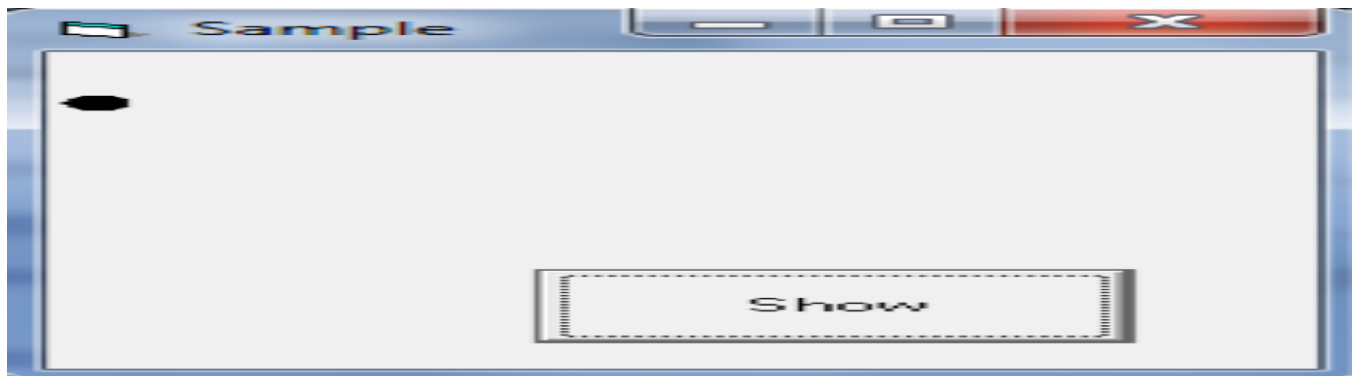
Drawing points using the PSet method

The Pset method allows you to draw a point. You need to specify the coordinate i.e. the drawing position. You can also pass a color constant that is an optional argument in the PSet method.

Example:-

```
Private Sub cmdShow_Click()  
    DrawWidth = 10  
    PSet (100, 500)  
End Sub
```

Output of code:-



Relative-positioning-with-the-Stepkeyword

The Step keyword allows you to draw in a position relative to the current position. See the example.

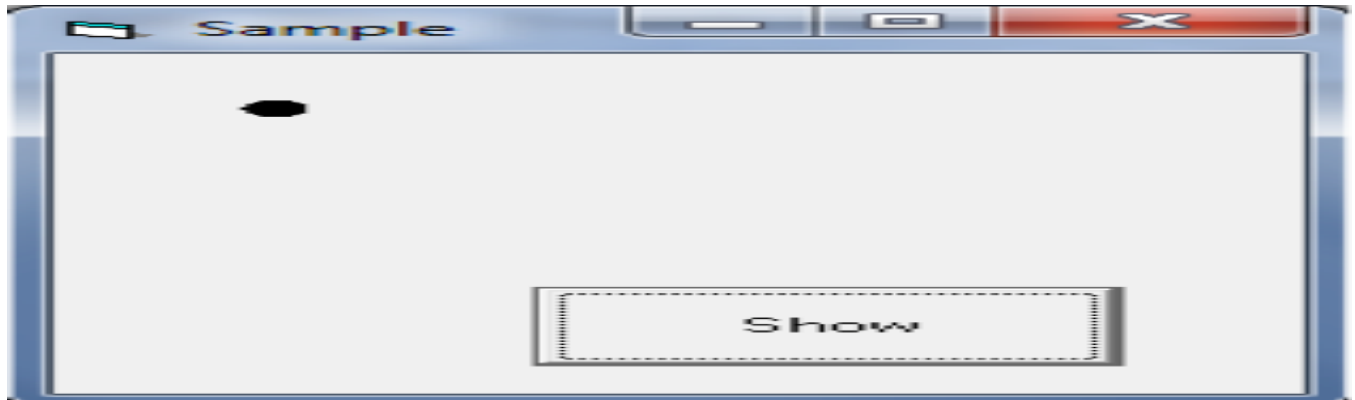
Example:-

```
Private Sub cmdShow_Click()  
    DrawWidth = 10  
    CurrentX = 500  
    CurrentY = 500  
    PSet Step(0, 0)  
End Sub
```

The above code draws a point in the (0, 0) position relative to the current position that is (500, 500).

That means, the point is drawn in the (500, 500) position. But this is (0, 0) position relative to the current position.

Output of code:-



Drawing lines

The Line method lets you draw lines in Visual Basic 6. You need to specify the starting point and the finishing point of the line in the argument. You may also specify the color of the line. This is optional, though.

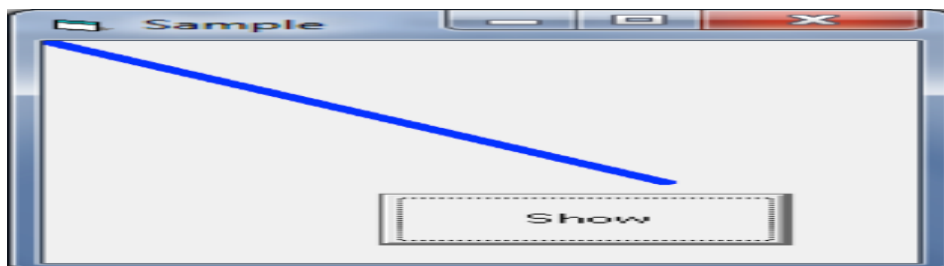
A-simple-line

The following code example shows how to draw a simple line using the Line method in Visual Basic 6.

Example:-

```
Private Sub cmdShow_Click()  
    DrawWidth = 5  
    'A hyphen is required between the points  
    Line (0, 0)-(2000, 2000), vbBlue  
End Sub
```

Output of code:-



A-line-with-drawing-styles

Form's DrawStyle property lets you draw lines using a particular style. The constant values of the DrawStyle property are 0 (vbSolid), 1 (vbDash), 2 (vbDot), 3 (vbDashDot), 4 (vbDashDotDot), 5 (vbTransparent) and 6 (vbInsideSolid). The default value is 0, vbSolid. You may use the numeric constant or the symbolic constant such as vbSolid, vbDash etc to change drawing styles in your code.

NOTE: The DrawStyle property does not work if the value of DrawWidth is other than 1.

Example:-

```
DrawWidth = 1
```

```
DrawStyle = 1
```

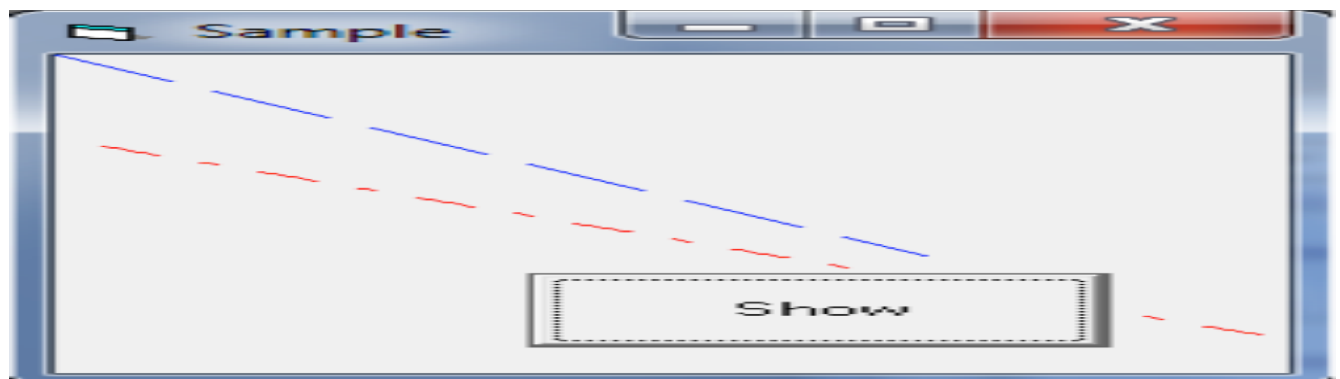
```
'A hyphen is required between the points
```

```
Line (0, 0)-(2000, 2000), vbBlue
```

```
DrawStyle = vbDashDot
```

```
Line (100, 900)-(2800, 2800), vbRed
```

Output of code:-



Drawing circles

You can draw a circle using the Circle method in Visual Basic 6. You may also use the Circle method to draw different geometric shapes such as ellipses, arcs etc. You need to specify the circle's center and radius values to draw a circle using the Circle method.

A-simple-circle

The following code draws a simple circle using the Circle method in Visual Basic 6.

Example:-

```
Private Sub cmdShow_Click()
```

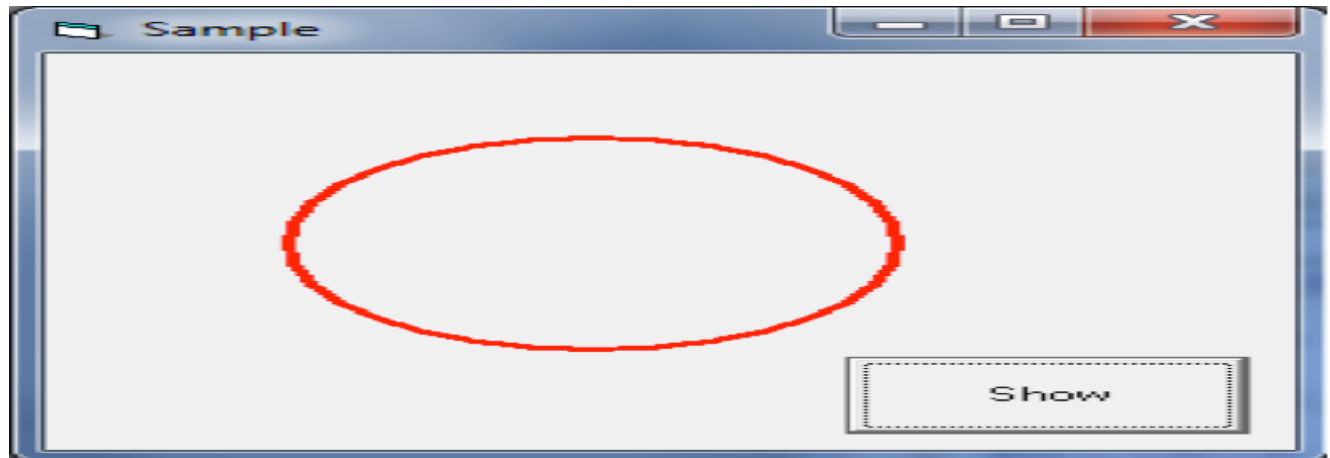
```
DrawWidth = 3
```

```
Circle (1800, 1800), 1000, vbRed
```

```
End Sub
```

In the above code, (1800, 1800) is the circle's center, and the radius value is 1000. The color constant 'vbRed' is an optional argument.

Output of code:-



A-circle-filled-with-color

The following code example shows how to fill a circle with color in Visual Basic 6.

Example:

```
Private Sub cmdShow_Click()
```

```
    FillStyle = vbSolid
```

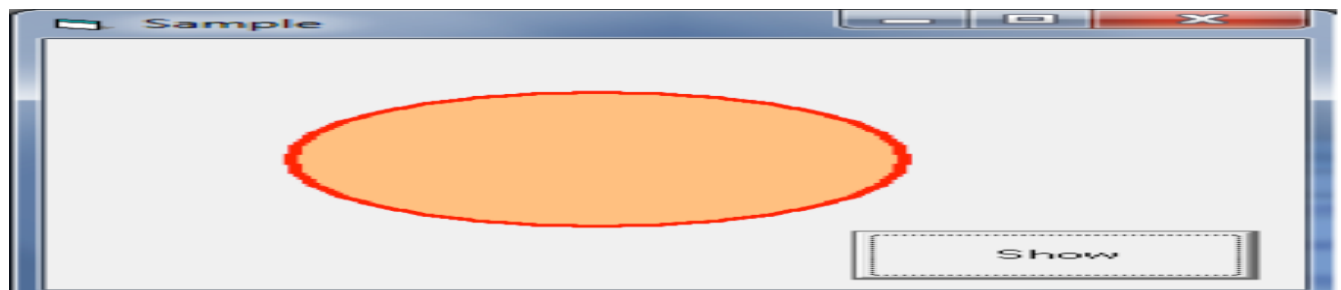
```
    FillColor = &H80C0FF
```

```
    DrawWidth = 3
```

```
    Circle (1800, 1800), 1000, vbRed
```

```
End Sub
```

Output of code:-



Rectangle

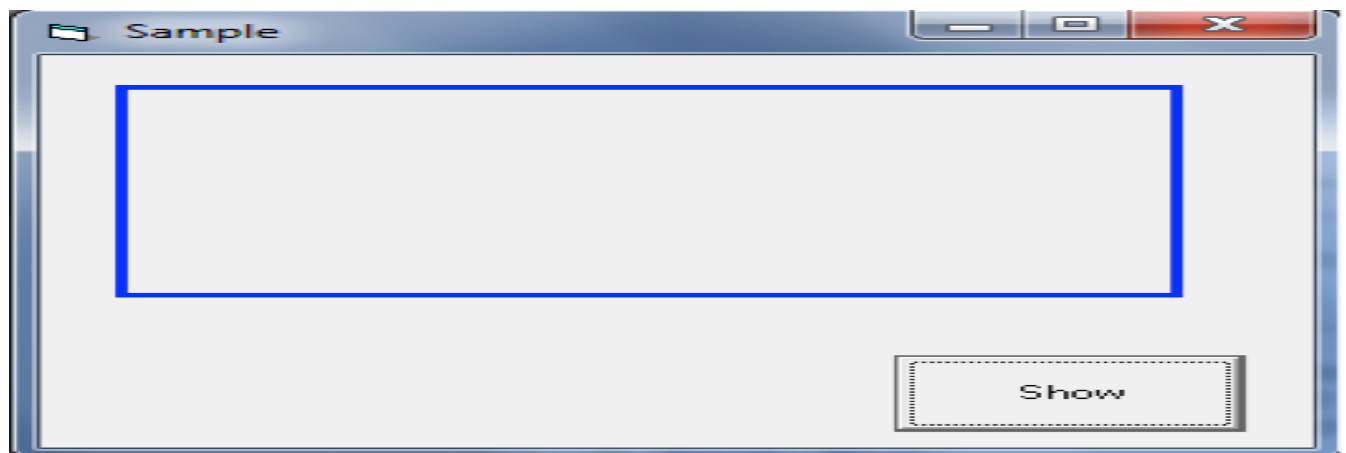
The Line method can be used to draw different geometric shapes such as rectangle, triangle etc. The following example shows you how to draw a rectangle using the Line method in Visual Basic6.

Example:-

```
Private Sub cmdShow_Click()  
    DrawWidth = 3  
    Line (300, 300)-Step(4000, 2000), vbBlue, B  
End Sub
```

The B argument in the Line method lets you draw a rectangle.

Output of code:-



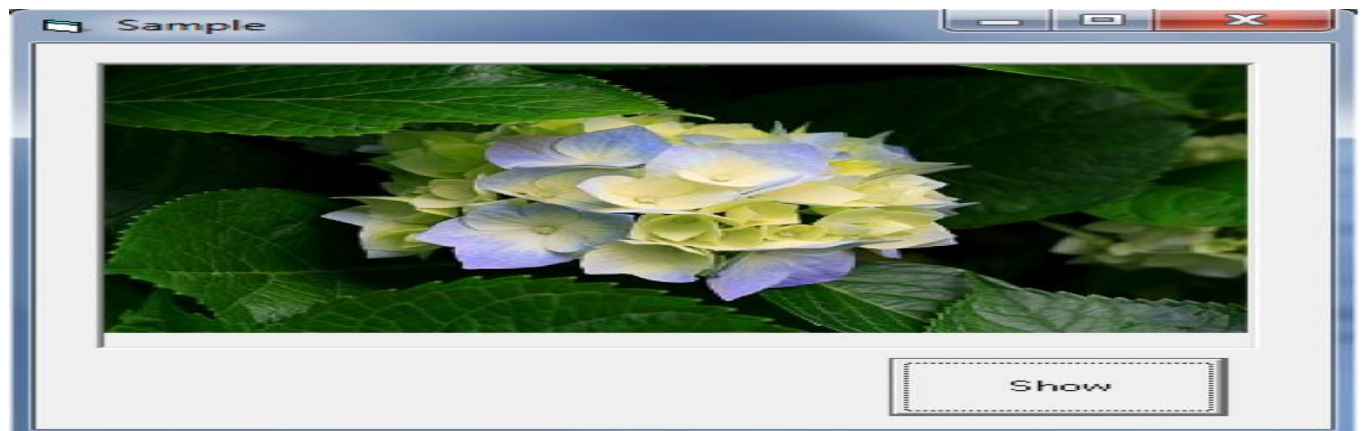
Displaying an image

The LoadPicture function sets a picture to the PictureBox control or the form object. It requires the file path as an argument. The following example shows you how to use the LoadPicture function.

Example:-

```
Private Sub cmdShow_Click()  
    Picture1 = LoadPicture("D:\pic.JPG")  
End Sub
```

Output of code:-



The Paint event

The Paint event fires automatically when the form is refreshed. For instance, the Paint event fires when you uncover areas in a form or when you resize the form. If the AutoRedraw property is set to True, this event will not be invoked. And while resizing, if you shrink the form, this event does not fire.

You may use the necessary graphic methods inside the Paint event procedure so that whenever the form is refreshed, the graphic methods are automatically called.

Example:-

```
Private Sub Form_Paint()  
    DrawWidth = 5  
    Circle (Rnd * 3000, Rnd * 7000), Rnd * 800, vbYellow  
End Sub
```

New circles are drawn automatically when you resize the form. New circles are drawn in random positions and with random sizes.

5. Write Short Notes on the following.

(A) Menu Editor Window.

You can use the Menu Editor to create new menus and menu bars, add new commands to existing menus, replace existing menu commands with your own commands, and change and delete existing menus and menu bars.

To display the Menu Editor

- From the **Tools** menu, choose **Menu Editor**.

—or—

Click the **Menu Editor** button on the toolbar.

While most menu control properties can be set using the Menu Editor, all menu properties are available in the Properties window. The two most important properties for menu controls are:

- Name — This is the name you use to reference the menu control from code.
- Caption — This is the text that appears on the control.

Other properties in the Menu Editor, including Index, Checked, and NegotiatePosition, are described later in this chapter.

Using the List Box in the Menu Editor

The menu control list box (the lower portion of the Menu Editor) lists all the menu controls for the current form. When you type a menu item in the Caption text box, that item also appears in the menu control list box. Selecting an existing menu control from the list box allows you to edit the properties for that control.

For example, Figure 6.7 shows the menu controls for a File menu in a typical application. The position of the menu control in the menu control list box determines whether the control is a menu title, menu item, submenu title, or submenu item:

- A menu control that appears flush left in the list box is displayed on the menu bar as a menu title.
- A menu control that is indented once in the list box is displayed on the menu when the user clicks the preceding menu title.
- An indented menu control followed by menu controls that are further indented becomes a submenu title. Menu controls indented below the submenu title become items of that submenu.
- A menu control with a hyphen (-) as its Caption property setting appears as a separator bar. A *separator bar* divides menu items into logical groups.

Note A menu control cannot be a separator bar if it is a menu title, has submenu items, is checked or disabled, or has a shortcut key.

To create menu controls in the Menu Editor

1. Select the form.
2. From the **Tools** menu, choose **Menu Editor**.

—or—

Click the **Menu Editor** button on the toolbar.

3. In the **Caption** text box, type the text for the first menu title that you want to appear on the menu bar. Also, place an ampersand (&) before the letter you want to be the access key for that menu item. This letter will automatically be underlined in the menu.

The menu title text is displayed in the menu control list box.

4. In the **Name** text box, type the name that you will use to refer to the menu control in code. See "Menu Title and Naming Guidelines" later in this chapter.
5. Click the left arrow or right arrow buttons to change the indentation level of the control.
6. Set other properties for the control, if you choose. You can do this in the Menu Editor or later, in the Properties window.
7. Choose **Next** to create another menu control.

—or—

Click **Insert** to add a menu control between existing controls.

You can also click the up arrow and down arrow buttons to move the control among the existing menu controls.

8. Choose **OK** to close the Menu Editor when you have created all the menu controls for that form.

The menu titles you create are displayed on the form. At design time, click a menu title to drop down its corresponding menu items.

Separating Menu Items

A separator bar is displayed as a horizontal line between items on a menu. On a menu with many items, you can use a separator bar to divide items into logical groups. For example, the File menu in Visual Basic uses separator bars to divide its menu items into three groups.

To create a separator bar in the Menu Editor

1. If you are adding a separator bar to an existing menu, choose **Insert** to insert a menu control between the menu items you want to separate.
2. If necessary, click the right arrow button to indent the new menu item to the same level as the menu items it will separate.
3. Type a hyphen (-) in the **Caption** text box.
4. Set the **Name** property.
5. Choose **OK** to close the Menu Editor.

Note Although separator bars are created as menu controls, they do not respond to the Click event, and users cannot choose them.

Assigning Access Keys and Shortcut Keys

You can improve keyboard access to menu commands by defining access keys and shortcut keys.

Access Keys

Access keys allow the user to open a menu by pressing the ALT key and typing a designated letter. Once a menu is open, the user can choose a control by pressing the letter (the access key) assigned to it. For example, ALT+E might open the Edit menu, and P might select the Paste menu item. An access-key assignment appears as an underlined letter in the menu control's caption.

To assign an access key to a menu control in the Menu Editor

1. Select the menu item to which you want to assign an access key.
2. In the **Caption** box, type an ampersand (&) immediately in front of the letter you want to be the access key.

For example, if the Edit menu shown in Figure 6.9 is open, the following Caption property settings respond to the corresponding keys.

Menu control caption	Caption property	Access keys
Cut	Cu&t	t
Copy	C&opy	o
Paste	&Paste	p
Delete	De&lete	l
Select All	Select &All	a
Time/Date	Time/&Date	d

To assign a shortcut key to a menu item

1. Open the **Menu Editor**.
2. Select the menu item.
3. Select a function key or key combination in the **Shortcut** combo box.

To remove a shortcut key assignment, choose "(none)" from the top of the list.

(B)Testing programs in Break mode.

At design time, you can change the design or code of an application, but you cannot see how your changes affect the way the application runs. At run time, you can watch how the application behaves, but you cannot directly change the code.

Break mode halts the operation of an application and gives you a snapshot of its condition at any moment. Variable and property settings are preserved, so you can analyze the current state of the application and enter changes that affect how the application runs. When an application is in break mode, you can do the following,

- Modify code in the application.
- Observe the condition of the application's interface.
- Determine which active procedures have been called.
- Watch the values of variables, properties, and statements.
- Change the values of variables and properties.
- View or control which statement the application will run next.
- Run Visual Basic statements immediately.
- Manually control the operation of the application.

Note You can set breakpoints and watch expressions at design time, but other debugging tools work only in break mode.

Entering Break Mode at a Problem Statement

When debugging, you may want the application to halt at the place in the code where you think the problem might have started. This is one reason Visual Basic provides breakpoints and Stop statements. A *breakpoint* defines a statement or set of conditions at which Visual Basic automatically stops execution and puts the application in break mode without running the statement containing the breakpoint. See "Using Stop Statements" later in this chapter for a comparison of Stop statements and breakpoints.

You can enter break mode manually if you do any of the following while the application is running:

- Press CTRL+BREAK.
- Choose Break from the Run menu.
- Click the Break button on the toolbar.

It's possible to break execution when the application is idle (when it is between processing of events). When this happens, execution does not stop at a specific line, but Visual Basic switches to break mode anyway.

You can also enter break mode automatically when any of the following occurs:

- A statement generates an untrapped run-time error.
- A statement generates a run-time error and the Break on All Errors error trapping option has been selected.
- A break expression defined in the Add Watch dialog box changes or becomes true, depending on how you defined it.
- Execution reaches a line with a breakpoint.
- Execution reaches a Stop statement.

Fixing a Run-Time Error and Continuing:-

Some run-time errors result from simple oversights when entering code; these errors are easily fixed. Frequent errors include misspelled names and mismatched properties or methods with objects — for example, trying to use the Clear method on a text box, or the Text property with a file list box.

Often you can enter a correction and continue program execution with the same line that halted the application, even though you've changed some of the code. Simply choose Continue from the Run menu or click the Continue button on the toolbar. As you continue running the application, you can verify that the problem is fixed.

If you select Break on All Errors from the Default Error Trapping State option group on the General tab on the Options dialog box (available from the Tools menu), Visual Basic disables error handlers in code, so that when a statement generates a run-time error, Visual Basic enters break mode. If Break on All Errors is not selected, and if an error handler exists, it will intercept code and take corrective action.

UNIT-V

TWO MARKS

1. Write the different kind of Drag-and-Drop operations (NOV'11)

- (i) DragMode Property
- (ii) DragIcon Property
- (iii) DragDrop Event
- (iv) DragOver Event
- (v) Drag Method

2. Write the advantages of OLE. (NOV'11)

Object Linking and Embedding (OLE) is a technology, developed by Microsoft that allow embedding and linking to documents and other objects. It is also used to develop and use custom user interface elements.

3. What are mouse event procedures? (APR'12, NOV'15)

- (i)MouseDown - When user clicks or press anyone of mouse buttons
- (ii) MouseUp - When user releases anyone of mouse buttons
- (iii) MouseMove - When user moves mouse pointer over the control or form.

4. What are file system controls? (APR'12)

File system controls in Visual Basic allow users to select a new drive, see the hierarchical directory structure of a disk, or see the names of files in a given directory. Various File system Controls are,

- (i) File List box,
- (ii) Directory List box, and
- (iii) Drive List box

5. Write a note on : The File Copy function. (NOV'12)

The file copy function copies a file from the source path to another path. It does not allow wildcards. The syntax of File Copy function is,

FileCopy Source, Destination

6. What is OLE Automation? (NOV'12, APR'14, APR'15)

OLE automation is an inter-process communication mechanism created by Microsoft. It is based on a subset of Component Object Model (COM), that was intended for use by scripting languages.

7. Write a note on: A directory list box. (APR'13)

A directory list box displays the directory structure of current drive. The current directory shows up as an open file folder. Subdirectories are shown as closed folders. Other directories are shown as non-shaded open folders.

8. What are Visual Basic's File handling Commands? (NOV'13, APR'16)

(i) MKDIR (ii) KILL (iii) NAME (iv) CHDIR (v) RMDIR

9. What is the usage of DLL servers? (NOV'13)

Dynamic Link Library (DLL) is a collection of small programs. They do not load into memory with main program. When it is called by main program at that time only it will be loaded into memory.

10. Define FILE. (NOV'14)

File is a collection of related informations, which are called as records. Visual basic handles two different types of files,

- (i) Text files and
- (ii) Binary files.

11. What is COM?

Component Object Model (COM) is a binary-interface standard for software components. It is used to enable inter-process communication and dynamic object creation.

12. Write any two advantages of DLL. (APR'15)

- 1. Saves memory
- 2. Saves Disk Space

13. Explain the two types of an application interface. (NOV'15)

- (i) Client Application or Container - Receives the Information,
- (ii) Server Application or Source - Sends the Information.

14. Write any four events/methods/properties used for drag and drop operation for controls. (APR'16)

DragDrop, DragOver, DragIcon, DragMode

15. What are the various file handling commands in VB?

FileCopy, FileDateTime, GetAttr, SetAttr

FIVE MARKS

1. Briefly Explain the OLE automation interface. (NOV'11, NOV'13)

- An object is a COM object implementing the IDispatch interface. Automation objects are referred as ActiveX objects.
- An application that manipulates an ActiveX object is referred to as an ActiveX client. This interface exposes four methods. Invoke method allows calling methods of a class by name, with a random number of parameters.
- Most existing COM components are automation compliant which allow both late binding and compile time early binding. This is achieved by implementing dual interfaces, which are derived from IDispatch interface.
- Late binding is slower and more reliable. For late binding, the client only needs to know the name of the desired object and the names and parameters of methods it actually uses.
- For early binding. The client needs to know the complete definition and identifier for every interface requested.
- Automation servers may be implemented as single-use or multi-use.
- In single-use, each client connects to an individual instance of the server with an individual instance of the class factory.
- In multi-use, many clients connect to the server, sharing that server's class factory across all clients.

2. Discuss briefly on any TWO VB file handling commands with suitable examples.

(APR'12)

CHDIR:

The ChDir command is used to change the current directory. The user can specify an absolute path or relative path.

Syntax: ChDir *pathname*

Example: ChDir D:\VB\Projects 'Current directory will be changed
'to D:\VB\Projects

KILL:

This command deletes the given file from specified path or current directory, It is also removes set of files.

Syntax: **Kill *pathname***

Example: **Kill test.txt** ‘Removes test.txt from current directory.

ExmapleL **Kill *.txt** ‘Removes all the txt extension files.

3. Write short notes on: The MouseEvent Procedures. (NOV’12) (NOV’13) (APR’16)

- All controls in visual basic are mouse sensitive. Everytime the user uses the mouse, the mouse event is fired.
- A click event is fired, if the user clicks a mouse. When the user double clicks a mouse, double click event will be fired.
- When the user press a button,MouseDown event will be fired. When the user releases a button, MouseUp event will be fired. Whenever the user moves a moues, MouseMove event will be executed.
- Most of the controls in VB support mouse event procedures.
- There are three mouse event procedures in visual basic.
 - ✓ MouseDown - It occurs when the user clicks or press any of the mouse button.
 - ✓ MouseUp - It occurs when the user releases anyone of the mouse button.
 - ✓ MouseMove - It occurs when the user moves the mouse pointer on the control or to the blank area of the form.
- All the mouse event procedures are of the same syntax and same parameters.

Syntax:

Sub Objectname_MouseEvent(*Button as Integer, Shift as Integer, X as single, Y as single*)

Statements

End Sub

4. Write short notes on: OLE Drag and Drop (APR’13)

- Mouse can be used in drag-and-drop, which initiate some action by dragging from one element on the screen to another.

- Visual basic provides excellent support for drag-and-drop. Traditional drag-and-drop is limited to dragging between controls in the same application.
- OLE drag-and-drop allows the user to drag data between applications as well as within an application.
- It works only with applications that support it.

The DataObject Class:

- The data object class is the heart of OLE drag-and-drop.
- When drag-and-drop operation started, the data that is being dragged is packaged in a DataObject object.
- When the item is dropped, this object is available to the target which examines whether the data format is supported by it. If so, the data is accepted by the target.
- Only a subset of VB controls support OLE drag-and-drop. The following controls act as a source as well as target. Textbox, PictureBox, Image, DirListbox, Listbox and FileListbox.
- Checkbox and Commandbutton are limited to serve as target.
- DataObject class use manual drag-and-drop. It has four methods.
 - Clear() - Removes all data and other information from the DataObject.
 - GetData() - Returns the data from the DataObject object as a variant.
 - SetData() - Inserts data into a DataObject object using a specified format.
 - GetFormat() - Returns a true/false value indicating whether data is available in the specified format.

5. Explain the functions of DLL servers. (NOV'14)

- Programming languages support two kinds of programming- Reusable code and Non reusable code. Reusability can be achieved with functions and procedures.
- Visual basic supports DLL servers to make the functions and procedures available to all the users.

- To construct a DLL, open a new project and in the new project dialog box, select ActiveX DLL.
- Then VB automatically adds a class module to the project explorer window. Properties window of the class module contains six properties. Important among those properties are as follows:

i. **Private**

This is the default value. It indicates that other applications are not allowed to access the information about the class, and cannot create instances of it. Private objects are available within the current project only.

ii. **PublicNotCreatable**

Other applications can use objects of this class. They cannot use the CreateObject function or New operator to create objects. This is helpful when a utility class is to be exposed and not to be modified by others.

iii. **MultiUse**

This property allows other applications to create objects from the class.

iv. **GlobalMultiuse**

It is same as MultiUse property, except that the properties and methods of the class can be invoked if they were global functions. It's not necessary to create an instance of the class explicitly.

6. Briefly explain the sequential file handling modes. (APR'15)

Sequential file provides the following modes.

- Input - Opens a file in read mode.
- Output - Opens a file in write mode.
- Append - If the given file already exists, any writes will be appended to the end of file.
- Create - Creates a new file and removes old file, if it already exists.
- CreateNew- Creates new file and returns error, if it already exists.
- Open - Opens an existing file and returns error, if it does not exist.

- **OpenorCreate-** If the file already exists, it will be opened otherwise a new file will be created.
- **Truncate -** Opens an existing file and deletes all existing content.

7. Explain the general form of the File Open Command. (NOV'15)

- The first step in working with files in Visual Basic is to open the file. This can be achieved by using FileStream class.
- The FileStream constructor accepts the file name to be opened as the first parameter, followed by a number of other parameters defining the mode in which the file to be opened.
- The general form of file open command is as follows.

Syntax:

Open <Filename> For <Mode> As <FileIdentifier>

Filename - Filename will be a valid variable name which denotes the physical name where the operating system maintains.

Mode - It denotes which operations are allowed on the file. (Input , Output, Append, etc.)

FileIdentifier - This is the number between 1 and 255 preceded by the “#” sign that is used to identify the file while it is open.

8. Explain any four file handling functions. (APR'16)

File handling functions in VB is:

- (i) FileCopy
- (ii) FileDateTime
- (iii) GetAttr
- (iv) SetAttr

(i) FileCopy

The fileCopy function copies a file from source path to destination path.

Syntax: **FileCopy source, destination**

Example: **FileCopy "e:\welcome.txt", "d:\app\test.txt"**

Note: This function does not allow wildcard characters

1. FileDateTime

It returns the date and time of a file that was created or last modified.

Syntax: **FileDateTime(File_path)**

Example: **Print FileDateTime("D:\calculate.java")**

2. GetAttr

It returns an integer value representing the type of file

Syntax: **GetAttr(File_path)**

Example: **Print GetAttr("D:\calculate.java")**

Various attributes are:

- a) Normal – VbNormal -0
- b) ReadOnly-VbReadOnly -1
- c) Hidden-VbHidden -2
- d) System – VbSystem- 4
- e) Volume – VbVolume- 8
- f) Directory – VbDirectory -16
- g) Archive – VbArchive- 32

GetAttr returns 3 when the file has both hidden and readonly attributes

i.e, VbHidden + VbReadOnly => 2 + 1 = 3

3. SetAttr

The SetAttr function sets attribute information for files.

Syntax: **SetAttr(PathName, Attributes)**

Example: **SetAttr "C:\Test.txt", vbHidden + vbReadOnly**

TEN MARKS

1. Explain the methods available in the File operations with necessary example.

(NOV'11)

[Ref. Question No. 9 in FIVE Marks]

2. Explain about OLE drag and drop. (APR'12, NOV'14)

- Mouse can be used in drag-and-drop, which initiate some action by dragging from one element on the screen to another.
- Visual basic provides excellent support for drag-and-drop. Traditional drag-and-drop is limited to dragging between controls in the same application.
- OLE drag-and-drop allows the user to drag data between applications as well as within an application.
- It works only with applications that support it.

The DataObject Class:

- The data object class is the heart of OLE drag-and-drop.
- When drag-and-drop operation started, the data that is being dragged is packaged in a DataObject object.
- When the item is dropped, this object is available to the target which examines whether the data format is supported by it. If so, the data is accepted by the target.
- Only a subset of VB controls support OLE drag-and-drop. The following controls act as a source as well as target. Textbox, PictureBox, Image, DirListbox, Listbox and FileListbox.
- Checkbox and Commandbutton are limited to serve as target.
- DataObject class use manual drag-and-drop. It has four methods.
 - i. Clear() - Removes all data and other information from the DataObject.

Syntax: Clear()

- ii. GetData() - Returns the data from the DataObject object as

a variant.

Syntax: GetData(Format)

iii. SetData()

- Inserts data into a DataObject object using a specified format.

Syntax: SetData Data, Format

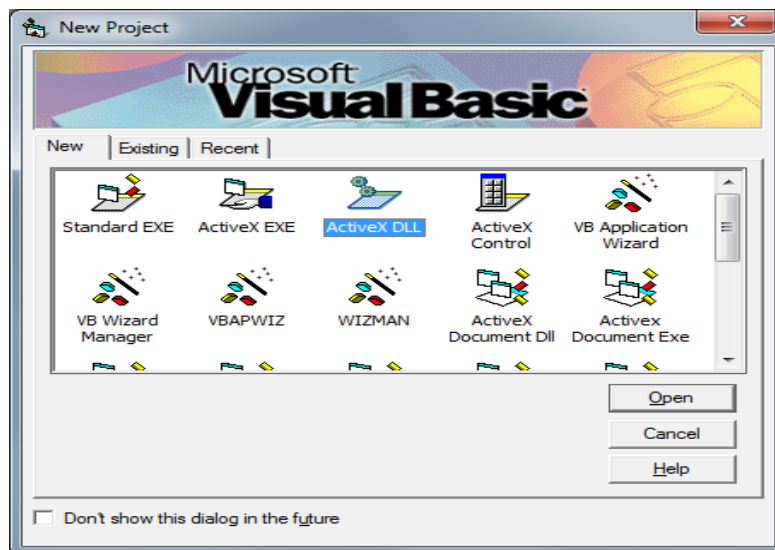
iv. GetFormat()

- Returns a true/false value indicating whether data is available in the specified format.

Syntax: GetFormat()

3. Describe in detail, Building COM/OLE DLL servers. (NOV'12)

- Programming languages support two kinds of programming- Reusable code and Non reusable code. Reusability can be achieved with functions and procedures.
- Visual basic supports DLL servers to make the functions and procedures available to all the users.
- To construct a DLL, open a new project and in the new project dialog box, select ActiveX DLL.



- Then VB automatically adds a class module to the project explorer window. Properties window of the class module contains six properties. Important among those properties are as follows:

i. Private

This is the default value. It indicates that other applications are not allowed to access the information about the class, and cannot

create instances of it. Private objects are available within the current project only.

ii. **PublicNotCreatable**

Other applications can use objects of this class. They cannot use the CreateObject function or New operator to create objects. This is helpful when a utility class is to be exposed and not to be modified by others.

iii. **MultiUse**

This property allows other applications to create objects from the class.

iv. **GlobalMultiuse**

It is same as MultiUse property, except that the properties and methods of the class can be invoked if they were global functions.

It's not necessary to create an instance of the class explicitly.

The following steps for creating and using DLL for adding two numbers are as follows:

- (1) Once the class module is created in the ActiveX dll project, set the instancing the property of the class into 6-GlobalMultiUse.
- (2) Change the name of the project into any valid name, for example "Addition"
- (3) Add the following code into the class:

```
Public Sub Add(X As Integer, Y As Integer)
    Dim Z As Integer
    Z = X + Y
    MsgBox "Added Result : " & Z
End Sub
```

- (4) Finally select "Make Addition.Dll" option from File Menu.
- (5) Now, Open a new project and select "Standard.Exe" . Goto **Project→References** select "Addition" and click "OK" button.
- (6) Write a click event on the form as shown below:

```
Private Sub Form_Click()
    Add 10, 20
End Sub
```

- (7) Now run the project. When the user clicks on the form add procedure will be called and the result "30" will be displayed on the message box.

4. Explain about OLE automation (APR'13)

- An object is a COM object implementing the IDispatch interface. Automation objects are referred as ActiveX objects.
- An application that manipulates an ActiveX object is referred to as an ActiveX client. This interface exposes four methods. Invoke method allows calling methods of a class by name, with a random number of parameters.
- Most existing COM components are automation compliant which allow both late binding and compile time early binding. This is achieved by implementing dual interfaces, which are derived from IDispatch interface.
- Late binding is slower and more reliable. For late binding, the client only needs to know the name of the desired object and the names and parameters of methods it actually uses.
- For early binding. The client needs to know the complete definition and identifier for every interface requested.
- Automation servers may be implemented as single-use or multi-use.
- In single-use, each client connects to an individual instance of the server with an individual instance of the class factory.
- In multi-use, many clients connect to the server, sharing that server's class factory across all clients.

5. Explain the File System Controls in VB with examples. (NOV'13, APR'16).

- File system controls in VB allow users to select a new drive, see the hierarchical directory structure, or see the name of the files in a given directory.
- **File List box**
 - A file list box displays the files in the current directory. It set the position, size, color and font characteristics through the properties window or through code as needed.

Properties:

Pattern - Pattern determines the files to be displayed in the file list box. It can be set in the design time as well as runtime and allows wildcard characters.

Path - Path property sets or returns the current path for the file list box.

Events:

Click, DblClick, Gotfocus, Lostfocus, Dragover, MouseUp, MouseDown, OLEDragOver, etc.

Example:

```
For I = 0 To File1.ListCount - 1
    Filenames$(I) = File1.List(I)
Next
```

- **Directory List Box**

- A directory list box displays the directory structure of the current drive.

Properties:

ListCount - This property returns the number of subdirectories in the current directory.

Path - It returns the current path.

Events:

Click, DragDrop, DragOver, Gotfocus, LostFocus, OLEDragdrop, etc.

Example:

```
Private Sub Dir1_Change()
    File1.Path = Dir1.Path
End Sub
```

- **Drive List Box**

- Drive list boxes are pull down list boxes which displays the current drive. When the user clicks on the arrow, VB pulls down a list of all the valid drives.

Properties:

Drive - Important property which sets or returns the current drive.

Events:

Change, DragDrop, DragOver, GotFocus, Lostfocus, etc.

Example:

```
Private Sub Drive1_Change()
```

Dir1.Path = Drive1.Drive

ChDrive Drive1.Drive

End Sub

6. How to monitor the activity of a mouse? Explain. (APR'14)

All controls in visual Basic are mouse sensitive. Every time if, the user uses the mouse a mouse event is fired. Also a click event is fired if the user clicks the mouse. When the user double clicks the mouse, a Dblclick event is fired. When the user presses a mouse button MouseDown event is fired. When the user releases the mouse, a MouseUp event will be fired. Every time the mouse moves, a MouseMove event occurs.

Most of the controls in VB support the mouse event procedure. Some controls such as combo box have no support for the mouse events.

The various mouse event procedures are:

- i) MouseDown – It occurs when the users clicks or press any of the mouse button.
- ii) MouseUp - – It occurs when the users releases any one of the mouse button.
- iii) MouseMove- – It occurs when the users moves the mouse pointer on the control or to the blank area of the form.

All the mouse event procedures are of the same form and same parameters.

```
Sub ObjName_MouseEvent(Button as Integer, Shift as Integer, X as Single, Y as Single)
    Statements
End Sub
```

Examples:

Here,

ObjName → is the name of the Control from which the event is being fired.

Button → is an integer which indicates the left, right or middle button of the mouse

Mouse Button Pressed	Button Parameter Value
Left	1
Right	2
Left & Right	3
Middle	4
Left & Middle	5
Right & Middle	6
All	7

Shift → It is an integer which indicates whether the mouse is clicked along with the shift key, ctrl key or Alt key.

X, Y → X and Y co-ordinates of the current mouse position.

MouseDown:

MouseDown event is executed when the user clicks any mouse button. Example for MouseDown event is as follows:

```
Private Sub Form_MouseDown(Button as Integer, Shift as Integer, X as single, Y as single)
    Circle(X,Y), 75
```

```
End sub
```

The above example simply draws small circles at the mouse's current location using the circle method on the form. The parameters X,Y represents the center of the circle and the second parameter 75 represents the radius of the circle in twips.

MouseUp:

The MouseUp event fired when the users releases any one of the mouse button. The below example describes the functionality of a MouseUp event of the form.

```
Private Sub Form_MouseUp(Button as Integer, Shift as Integer, X as single, Y as single)
    Dim c as Integer
    Randomize
    C=Int (15 * Rnd)
    Fillstyle = 0
    FillColor=QBColor(C)
    Circle(X,Y),300
```

```
End sub
```

```
Private Sub Form_MouseDown(Button as Integer, Shift as Integer, X as single, Y as single)
    Circle(X,Y), 75
```

```
End sub
```

MouseMove:

VB does not generate a MouseMove event for every Pixel. The number of events generated will be limited by the software and hardware combination and its architecture. However whenever the user moves the mouse, a MouseMove event is fired.

Example:

```
Private Sub CtrlNmae_MouseMove(Button as Integer, Shift as Integer, X as single, Y as single)
    PSet(X,Y)
End Sub
```

7. Explain the Following:(APR'15)

a. File system Controls

[Ref. Question No. 5 in TEN Marks]

b. DLL Server.

[Ref. Question No. 5 in FIVE Marks]

**8. Explain any five events/methods/properties used for drag and drop operation for controls.
(NOV'15)**

The following table summarizes the events, methods and properties used for dragging and dropping of controls.

Item	Description
DragMode Property	Allows automatic dragging (value=1) or manual dragging (value=0)
DragIcon Property	Set this to change from the gray rectangle to a custom icon when dragging.
DragDrop Event	Associated with the target of the operation. This is generated when the source is dropped on the target control.
DragOver Event	Associated with any control. The source control passes over during dragging.
Drag Method	Starts or stops dragging when dragmode is set to manual.