CHAPTER 5

Windows and Web Applications Development

CHAPTER OUTLINE

5.1 DESIGNING ASPECTS OF C#.NET WINDOWS APPLICATION FORM

Designing a C# .NET Windows application form involves a few key aspects that you should consider to create an effective user interface. Here are some of the important things to keep in mind :

- **Layout :** Decide on the layout of your form. You can use a standard layout, such as a single-column or two-column layout, or you can create a custom layout that fits your specific needs. Make sure that the layout is intuitive and easy to navigate.
- **Visual Design :** Choose a color scheme and font that are appropriate for your application. Use contrasting colors and appropriate font sizes to make your text easy to read. Use appropriate icons and images to help users understand what actions they can take on the form.
- **Form Controls :** C#.NET provides several form controls, including labels, text boxes, buttons, radio buttons, check boxes, and drop-down lists. Group related controls together and use labels to clearly indicate what each control does. Select the appropriate form controls based on the user's needs and the purpose of the application.
- **Data Validation :** When users enter data into your application, you should validate it to ensure it meets the required format and values. Use appropriate validation controls, such as required field validator, Regular expression validator, and compare validator, to validate the user input. Provide clear error messages to guide users in correcting any errors.
- **Error Handling :** Your application should handle errors gracefully and provide meaningful error messages to the user. Use try-catch blocks to catch exceptions and display error messages that are easy to understand.
- **Accessibility :** Your application should be accessible to all users, including those with disabilities, by including features such as keyboard navigation, ALT text for images, screen reader compatibility, and text-to-speech functionality.
- **Navigation :** The application should be easy to navigate, and the user should be able to move from one section to another with ease. Use menus, tabs, and buttons to facilitate navigation.
- **User Feedback :** Provide feedback to users when they take an action on the form, such as submitting a form or clicking a button. Use messages, notifications, or progress bars to keep users informed of what is happening.

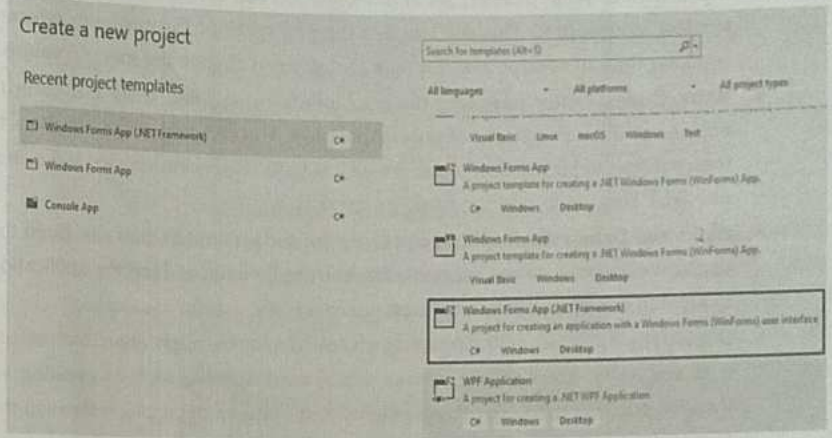
- **Help Documentation :** Provide clear and concise help documentation to the user, so they can easily navigate and use your application. Include FAQs, user guides, and tutorials to help users get started with your application.

By keeping these aspects in mind, you can design a user-friendly and effective C# windows application form.

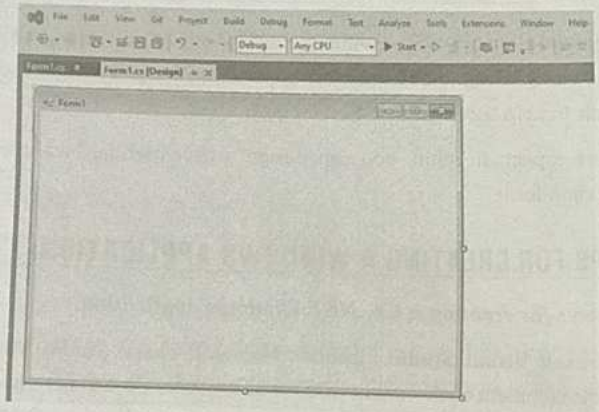
5.2 STEPS FOR CREATING A WINDOWS APPLICATION

Here are the steps for creating a C# .NET Windows application :

1. **Open Microsoft Visual Studio :** Launch Microsoft Visual Studio, which is the integrated development environment (IDE) used to create C# windows applications.
2. **Create a New Project :** Select "create a new project" from the visual studio start page. In the new project dialog box, select "Windows Forms App (.NET Framework)" from the list of project templates, choose a name and location for your project, and click "Create".



3. **Design the user Interface :** Once the project is created, the visual studio designer will open a blank form for you to design the user interface of your application. You can add and configure form controls, such as text boxes, labels, buttons, and menus, to build the user interface.You can switch between this view and code view at any time by right-clicking the design surface or code window and then clicking view code or view designer.



At the top of the form displays the forms title. Form1 is the default name, and you can change the name to your convenience. The title bar also includes the control box, which holds the minimize, maximize, and close buttons.

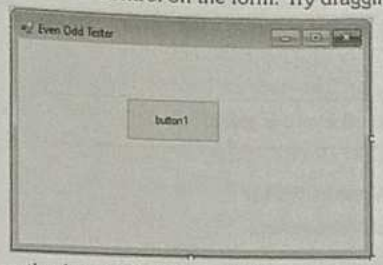
If you want to set any properties of the form, you can use visual studio property window to change it. This window lists the properties of the currently selected windows form or control, and it is here that you can change the existing values.

4. **Write Code :** After designing the user interface, you can write the code to implement the functionality of your application. You can add event handlers to form controls to handle user input, create functions to perform specific tasks, and use .NET libraries to perform complex operations.
5. **Build and Debug the Application :** Once the code is written, you can build the application to generate an executable file. You can then run and test the application to identify and fix any errors or bugs.
6. **Deploy the Application :** After testing, you can deploy the application to distribute it to end-users. You can use various deployment options, such as creating an installer, publishing the application to a network share, or deploying it through the Microsoft Store.

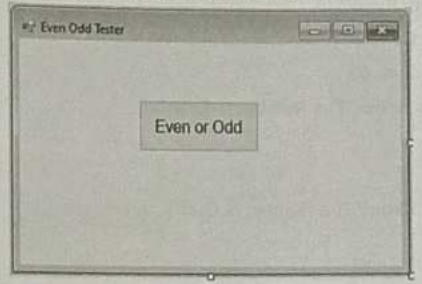
Example of Windows Application : Lets make a simple windows application that will allow the user to enter a number into the TextBox and press a button. Upon pressing the button we will get output whether the number is even or odd in a Messagebox.

First, select the form by clicking on it. In the properties window, you can click and modify various properties of the form. We change the title bar of the form to "Even Odd Tester" by changing the Text property.

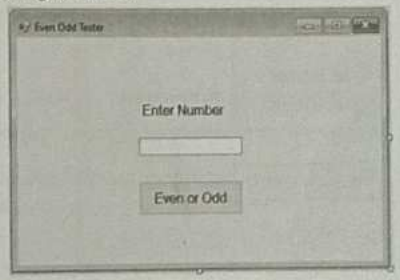
To add controls to the form, select the control you want from the Toolbox on the left. Then click and drag in the form window in the location you want the control to go. You can also click and resize the control on the form. Try dragging a button onto the form :



Click the button on the form and change its Name property to "btnEvenOdd". This is setting the name of the variable used to reference the button object from its default of "button1". Also change the buttons text property to "Even or Odd". You should see the change to the text field take place on the form :



In the same fashion as the button, add a label to the form and name it "lblResult" and set its Text field to "Enter Number". Also add a TextBox to the form, name it "txtNumber" and delete the contents of the Text field. As you add the controls to the form, you will see vertical lines that help you align controls to each other. The result should look something like this :



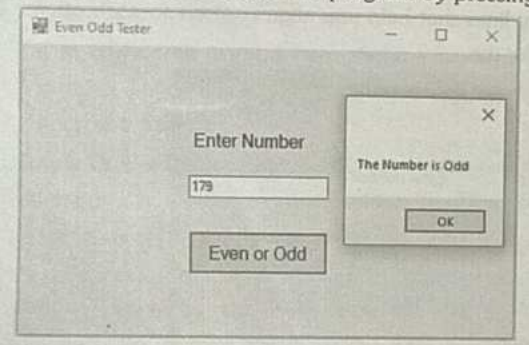
Select the Button and click on it. This defines a method named "btnEvenOdd_Click" that will be invoked when it is clicked. Immediately Visual Studio will bring you to an empty code skeleton that it generates for the method :

```
private void btnEvenOdd_Click(object sender, EventArgs e){
    }
```

The code for the Clicked Event is given below

```
using System;
using System.Windows.Forms;
namespace FirstWindowsApp{
public partial class Form1 : Form{
public Form1(){
Initialize Component();
    }
private void btnEvenOdd_Click(object sender, EventArgs e){
int number = Convert.ToInt32(txtNumber.Text);
if (number % 2 == 0){
Message Box.Show("The Number is Even");
    }
else{
MessageBox.Show("The Number is Odd");
    }
  }
 }
}
```

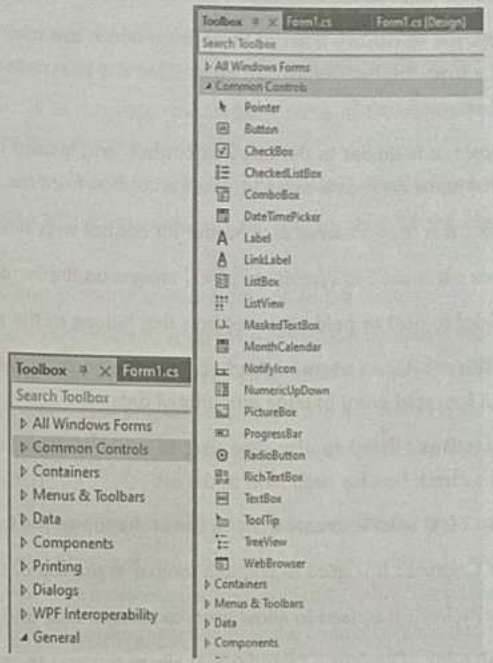
Debug the program or execute the program by pressing F5.



5.3 VARIOUS ELEMENTS OF USER INTERFACE

The user interface is what appears in the applications window when it runs. It consists of various elements with which the user can interact and control the application. The first element of the user interface is the form.

In the visual studio environment you can see the tool box. There are lots of controls grouping there in the tool box according to their functionalities. Just click the > sign before each group then you can see the controls inside the group. You can select basic controls from common controls group. You can place the control in your form by drag and drop the control from your toolbox to form control.



1. **Pointer :** This is the only item in the Toolbox that doesn't draw a control. It is used to resize or move a control after it's been drawn on a form.
2. **Label :** Used to display text on the form that you don't want the user to change, such as a caption under a graphic.
3. **LinkLabel :** A Hyperlink label.

4. **TextBox :** Used to allow the users to enter and edit text.
5. **Button :** It is the most common element of the Windows interface. It represents an action that is carried out when the user clicks the button.It is used to execute a command or trigger an event when clicked.
6. **CheckBox :** Used to create a box that the user can easily choose to indicate if something is true or false, or to display multiple choices when the user can choose more than one.It is used to toggle the selection of an option.
7. **RadioButton :** Used in a group of option buttons to display multiple choices from which the user can choose only one.
8. **ListBox :** Used to display a list of items from which the user can select one or more items from the list.The list can be scrolled if it has more items than can be displayed at one time.
9. **ComboBox :** It is similar to the List Box control, and is used to provide a drop-down list of items and allow users to select an option from the list.
10. **GroupBox :** It is used to serve as a border for control with similar needs.
11. **PictureBox :** It is used to display graphical images on the form.
12. **Panel :** Used to host or hold other controls that belong to the same group.
13. **DataGridView :** Allows users see and edit multiple rows of data simultaneously, also useful for rapid entry of large amounts of data
14. **CheckedListBox :** Used to allow the user to select multiple items in the list by providing a check box for each item in the list.
15. **MenuStrip :** It is used to create a menu bar at the top of the form.
16. **ToolStrip Control :** It is used to create a toolbar at the top of the form.
17. **DateTimePicker :** It is used to allow users to select a date and time.
18. **MonthCalendar :** Displays a calendar that allows the user to change months and select a date.
19. **HScrollBar and VScrollBar :** These controls allow users to scroll through large amounts of data that do not fit within the visible area of a control.
20. **NumericUpDown :** It is used to allow users to select a numeric value by clicking on up and down arrows.

21. **ProgressBar :** It is used to display the progress of an operation.
22. **TrackBar :** It is used to allow users to select a numeric value by sliding a pointer along a track.
23. **ToolTip :** It is used to display a brief message when the mouse pointer is positioned over a control.
24. **Timer :** It is used to perform tasks at regular intervals.

5.4 PROPERTIES OF CONTROLS LIKE LABEL, TEXTBOX, BUTTON, CHECKBOX, RADIOBUTTON, COMBOBOX, LISTBOX, DATAGRID

C# provides a variety of user interface controls that can be used in windows applications. Some common properties that these controls share include :

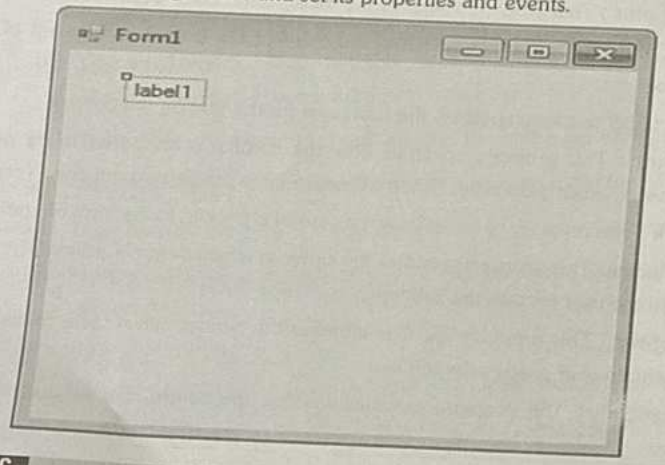
- **Name :** This property specifies the name of the control, which can be used to refer to the control in code.
- **Text :** This property specifies the text that is displayed on the control.
- **Location :** This property specifies the location of the control on the form or container.
- **Size :** This property specifies the size of the control.
- **Visible :** This property specifies whether the control is visible or hidden.
- **Enabled :** This property specifies whether the control is enabled or disabled.
- **BackColor :** This property specifies the background color of the control.
- **ForeColor :** This property specifies the color of the text or foreground of the control.
- **Font :** This property specifies the font used for the text on the control.
- **Anchor :** This property specifies how the control is anchored to its parent container, which determines how it will resize when the parent container is resized.
- **Dock :** This property specifies how the control is docked to the form or container.
- **TabIndex :** This property specifies the order in which controls will receive focus when the user presses the Tab key.
- **TabStop :** This property specifies whether the control can receive focus or not when the user presses the tab key.
- **ReadOnly :** This property specifies whether the control can be edited by the user.

- **TextAlign :** This property specifies the alignment of the text on the control, if applicable.
- **PasswordChar :** This property specifies the character used to mask input in a password field.
- **Checked :** This property specifies whether a check box or radio button is checked.
- **BorderStyle :** This property specifies the style of the border around the control.
- **AutoSize :** This property specifies whether the control automatically adjusts its size to fit its contents.
- **Value :** This property specifies the value of the control, such as the selected item in a drop-down list or the text entered in a text box.

(i) **Label Control :** Label control is used to display text on a form or container without allowing the user to edit or interact with the text. Usually you need to add no event handling code for a standard label. A Label control is used as a display medium for text on forms. Label control does not participate in user input or capture mouse or keyboard events.

Creating a Label

At Design-time : To create a label control at design-time, you simply drag and drop a label control from toolbox to a Form. After you drag and drop a label on a form. The Label looks like below Figure. Once a Label is on the form, you can move it around and resize it using mouse and set its properties and events.



At Run-time : Label class represents a Label control. We simply create an instance of Label class, set its properties and add this it to the Form controls.

In the first step, we create an instance of the Label class. The following code creates a Label control object.

```
// Create a Label object
Label label1 = new Label();
```

In the next step, we set properties of a label control. The following code sets background color, foreground color, Text, Name, and Font properties of a Label.

```
// Set background and foreground
label1.BackColor = Color.Red;
label1.ForeColor = Color.Blue;
label1.Text = "I am a Username";
label1.Name = "UserNameLabel";
label1.Font = new Font("Verdana", 16);
```

In the last step, we need to add a Label control to the Form by calling Form.Controls.Add method. The following code adds a Label control to a Form.

```
Controls.Add(label1);
```

Label Properties :

- **Name :** This property specifies the name of the label control.
- **Text :** This property specifies the text that is displayed in the label control.
- **TextAlign :** This property specifies the alignment of the text within the Label control. The options include Left, Center, and Right.
- **Font :** This property specifies the font used for the text in the Label control.
- **ForeColor :** This property specifies the color of the text in the Label control.
- **BackColor :** This property specifies the background color of the Label control.
- **AutoSize :** This property specifies whether the Label control automatically adjusts its size to fit its contents.
- **BorderStyle :** This property specifies the style of the border around the Label control. The options include None, FixedSingle, Fixed3D, and etched.

- **Enabled :** This property specifies whether the Label control is enabled or disabled.
- **Visible :** This property specifies whether the Label control is visible or hidden.
- **Location :** This property specifies the location of the Label control on the form or container.
- **Size :** This property specifies the size of the Label control.

(ii) **TextBox Control :** TextBox control is used to allow the user to enter and edit text. A TextBox control is used to display, or accept as input, a single line of text. In a text box, a user can type data or paste it into the control from the clipboard. For displaying a text in a TextBox control, you can code like this.

```
textBox1.Text = "Hello";
```

You can also collect the input value from a TextBox control to a variable like this way.

```
string var;
var = textBox1.Text;
```

TextBox Properties :

- **Name :** This property specifies the name of the TextBox control.
- **Text :** This property specifies the text that is displayed in the TextBox control.
- **TextAlign :** This property specifies the alignment of the text within the TextBox control. The options include Left, Center, and Right.
- **TextLength :** This property returns the length of a TextBox contents.
- **Font :** This property specifies the font used for the text in the TextBox control.
- **ForeColor :** This property specifies the color of the text in the TextBox control.
- **BackColor :** This property specifies the background color of the TextBox control.
- **AutoSize :** This property specifies whether the TextBox control automatically adjusts its size to fit its contents.
- **AutoCompleteMode :** This property specifies the type of automatic completion available for the TextBox control. The options include None, Suggest, and Append.
- **BorderStyle :** This property specifies the style of the border around the TextBox control. The options include None, FixedSingle, Fixed3D, and etched.

- **ReadOnly :** This property specifies whether the TextBox control is read-only, meaning that the user cannot edit the text in the control.
- **Multiline :** This property specifies whether the TextBox control allows multiple lines of text or only a single line.
- **ScrollBars :** This property specifies the type of scrollbars that are displayed in the TextBox control. The options include None, Horizontal, Vertical, and Both.
- **PasswordChar :** This property specifies the character that is displayed in place of the actual characters entered in the TextBox control when it is used to enter passwords.This is commonly used for password entry fields.
- **CharacterCasing :** This property specifies the case of the characters entered in the TextBox control. The options include UpperCase, LowerCase, and Normal.
- **MaxLength :** This property specifies the maximum number of characters that can be entered in the TextBox control.
- **WordWrap :** This property specifies whether the text in the TextBox control wraps to the next line when it reaches the edge of the control.
- **ScrollBars :** This property specifies whether the TextBox control displays scroll bars for scrolling through long blocks of text. The options include None, Horizontal, Vertical, and Both.
- **SelectedText :** This property returns the selected text in a TextBox control.
- **HideSelection :** This property specifies whether the selected text in the TextBox control is hidden when the control loses focus.

TextBox Events

TextChanged Event : When user input or setting the Text property to a new value raises the TextChanged event.

```
private void textBox1_TextChanged(object sender, EventArgs e)
    {
        label1.Text= textBox1.Text;
    }
```

Keydown Event :

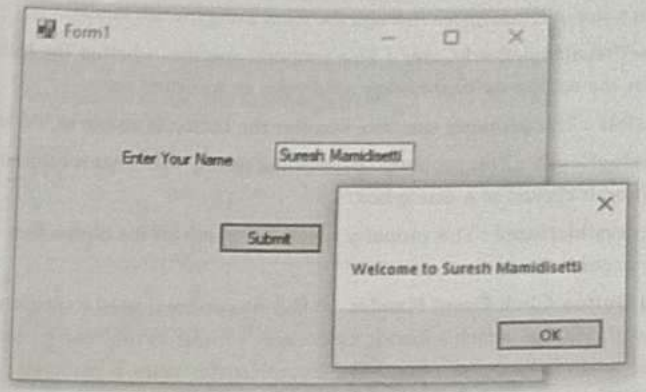
You can capture which key is pressed by the user using KeyDown event

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
    {
if (e.KeyCode == Keys.Enter)
    {
MessageBox.Show("You pressed Enter Key");
    }
if (e.KeyCode == Keys.CapsLock)
    {
MessageBox.Show("You pressed CapsLock Key");
    }
  }
```

TextBox Example

```
using System;
using System.Windows.Forms;
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
    }
private void button1_Click(object sender, EventArgs e)
    {
MessageBox.Show("Welcome to " + textBox1.Text);
    }
  }
}
```


Output :



(iii) Button Control : The Button control is used to perform an action when the user clicks on it.A Button can be clicked by using the mouse, ENTER key, or SPACEBAR if the button has focus.

Button Control Properties :

- **Name :** This property specifies the name of the Button control.
- **Text :** This property specifies the text that is displayed on the Button control.
- **Font :** This property specifies the font used for the text on the Button control.
- **ForeColor :** This property specifies the color of the text on the Button control.
- **BackColor :** This property specifies the background color of the Button control.
- **Enabled :** This property specifies whether the Button control is enabled or disabled.
- **FlatStyle:** This property specifies the appearance or style of the Button control. The options include Flat, Popup, Standard, and System.
- **Image :** This property specifies the image displayed on the Button control.
- **ImageAlign :** This property specifies the alignment of the image on the button.
- **TextImageRelation :** Gets or sets the position of text and image relative to each other.
- **TextAlign :** This property specifies the alignment of the text within the button control. The options include left, center, and right.
- **Size :** This property specifies the size of the button control.

- **TabIndex :** This property specifies the order in which the button control is accessed when the user navigates through the form using the tab key.
- **UseVisualStyleBackColor :** This property specifies whether the button control uses the system-defined background color or a custom color.
- **Visible :** This property specifies whether the button is visible or hidden.
- **DialogResult :** This property specifies the dialog result that is returned when the button is clicked in a dialog box.
- **AccessibleName :** This property specifies a name for the button that can be used by accessibility tools.

Adding Button Click Event Hander : A Button control is used to process the button click event. We can attach a button click event handler at run-time by setting its click event to an EventHandler obect. The EventHandler takes a parameter of an event handler. The click event is attached in the following code snippet.

```
button1.Click += new EventHandler(button1_Click);
```

The signature of button click event handler is listed in the following code snippet.

```
private void button1_Click(object sender, EventArgs e)
{
}
```

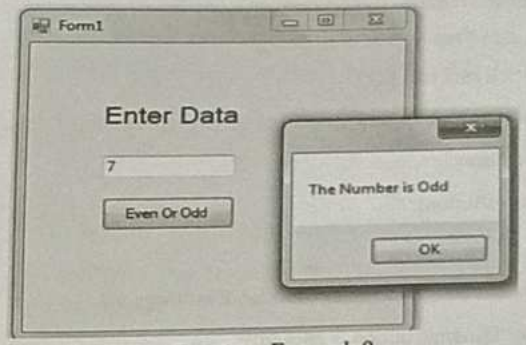
Button Control Example1

```
using System;
using System.Windows.Forms;
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
    }
private void btnEvenOdd_Click(object sender, EventArgs e){
int number = Convert.ToInt32(txtNumber.Text);
if (number % 2 == 0){
```



```
MessageBox.Show("The Number is Even");
    }
else{
MessageBox.Show("The Number is Odd");
    }
  }
 }
}
```

Output :

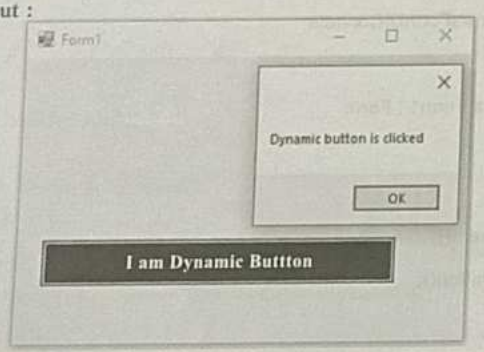


Creating Button at Runtime Example2

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1(){
Initialize Component();
CreateDynamicButton();
    }
```

```
private void CreateDynamicButton(){
        Button dynamicButton = newButton();
// Set Button properties
        dynamicButton.Height = 40;
        dynamicButton.Width = 300;
        dynamicButton.BackColor = System.Drawing.Color.Red;
        dynamicButton.ForeColor = Color.Blue;
        dynamicButton.Location = newPoint(20, 150);
        dynamicButton.Text = "I am Dynamic Button";
        dynamicButton.Name = "DynamicButton";
        dynamicButton.Font = newFont("Georgia", 16);
// Add a Button Click Event handler
        dynamicButton.Click += new EventHandler(DynamicButton_Click);
// Add Button to the Form
Controls.Add(dynamicButton);
    }
private void DynamicButton_Click(object sender, EventArgs e){
MessageBox.Show("Dynamic button is clicked");
    }
  }
}
```

Output :



(iv) **CheckBox Control :** CheckBox control is used to provide a user interface for selecting or deselecting an option. CheckBoxes allow the user to make multiple selections from a number of options. CheckBox will give the user an option, such as true/false or yes/no. You can click a check box to select it and click it again to deselect it.

CheckBox Properties :

- **Name :** This property specifies the name of the CheckBox control.
- **Text :** This property specifies the text that is displayed next to the CheckBox control.
- **TextAlign :** This property specifies the alignment of the text next to the CheckBox control.
- **Font :** This property specifies the font used for the text next to the CheckBox control.
- **ForeColor :** This property specifies the color of the text next to the CheckBox control.
- **BackColor :** This property specifies the background color of the CheckBox control.
- **Checked :** This property specifies whether the CheckBox control is checked or unchecked.
- **CheckAlign :** This property is used to align the check mark in a CheckBox.
- **CheckState :** This property specifies the current state of the CheckBox control. The options include Checked, Unchecked, and Indeterminate.
- **AutoCheck :** This property specifies whether the CheckBox control automatically checks or unchecks when clicked by the user.
- **ThreeState :** This property specifies whether the CheckBox control has three states. If set to true, the checkbox can be in one of three states: checked, unchecked, or indeterminate.
- **Appearance :** This property specifies the appearance of the CheckBox control. The options include Normal, Button, and Flat.
- **Size :** This property specifies the size of the checkbox.
- **Enabled :** This property specifies whether the CheckBox control is enabled or disabled.

- **Visible :** This property specifies whether the CheckBox control is visible or hidden.
- **TabIndex :** This property specifies the tab order of the CheckBox control.
- **AutoSize :** Gets or sets a value that indicates whether the control resizes based on its contents.
- **AutoEllipsis :** This property specifies whether the CheckBox control automatically truncates text that is too long to fit within the control.

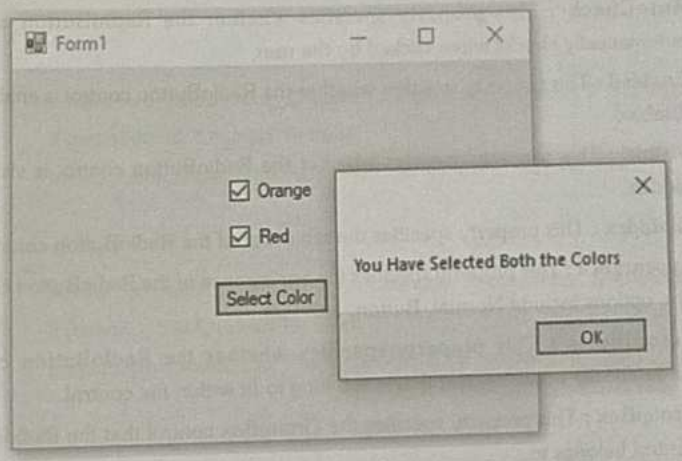
EXAMPLE

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1(){
InitializeComponent();
    }
private void button1_Click(object sender, EventArgs e){
if ((checkBox1.Checked == true) && (checkBox2.Checked == false)){
this.BackColor = Color.Orange;
      }
else if ((checkBox1.Checked == false) && (checkBox2.Checked == true)){
this.BackColor = Color.Red;
      }
else if ((checkBox1.Checked == true) && (checkBox2.Checked == true)){
MessageBox.Show("You Have Selected Both the Colors");
      }
else{
```



```
MessageBox.Show("You Did not Selected Any Color");
      }
    }
  }
}
```

Output :



(v) **RadioButton Control :** RadioButton control is used to allow the user to select one option from a group of options. A radio button or option button enables the user to select a single option from a group of choices when paired with other RadioButton controls. When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked.

Radio Button Properties :

- **Name :** This property specifies the name of the RadioButton control.
- **Text :** This property specifies the text that is displayed next to the RadioButton control.
- **TextAlign :** This property specifies the alignment of the text next to the RadioButton control.
- **Font :** This property specifies the font used for the text next to the RadioButton control.

- **ForeColor :** This property specifies the color of the text next to the RadioButton control.
- **BackColor :** This property specifies the background color of the RadioButton control.
- **Checked :** This property specifies whether the RadioButton control is checked or unchecked.
- **CheckAlign :** This property is used to align the check mark in a RadioButton.
- **AutoCheck :** This property specifies whether the RadioButton control automatically checks when clicked by the user.
- **Enabled :** This property specifies whether the RadioButton control is enabled or disabled.
- **Visible :** This property specifies whether the RadioButton control is visible or hidden.
- **TabIndex :** This property specifies the tab order of the RadioButton control.
- **Appearance :** This property specifies the appearance of the RadioButton control. The options include Normal, Button, and Flat.
- **AutoEllipsis :** This property specifies whether the RadioButton control automatically truncates text that is too long to fit within the control.
- **GroupBox :** This property specifies the GroupBox control that the RadioButton control belongs to.
- **GroupName :** This property specifies the name of the group of RadioButton controls to which the current RadioButton control belongs. This property is used to ensure that only one RadioButton control in a group is checked at any given time.

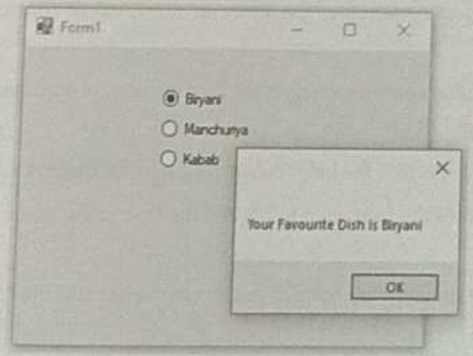
EXAMPLE

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormUIControls
{
public partial class Form1 : Form
 {
```



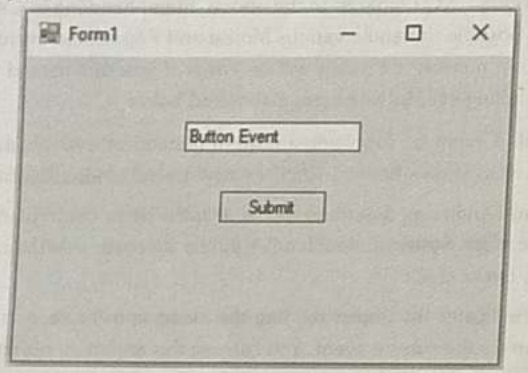
```
  public Form1()
      {
  InitializeComponent();
      }
  private void radioButton1_CheckedChanged(object sender, EventArgs e){
  if (radioButton1.Checked == true){
  MessageBox.Show("Your Favourite Dish is " + radioButton1.Text);
        }
      }
  private void radioButton2_CheckedChanged(object sender, EventArgs e){
  if (radioButton2.Checked == true){
  MessageBox.Show("Your Favourite Dish is " + radioButton2.Text);
        }
      }
  private void radioButton3_CheckedChanged(object sender, EventArgs e){
  if (radioButton3.Checked == true){
  MessageBox.Show("Your Favourite Dish is " + radioButton3.Text);
        }
    }
  }
}
```

Output :



```
public Form1()
    {
InitializeComponent();
        button1.Click += newEventHandler(button1_Click);
    }
private void button1_Click(object sender, EventArgs e)
{
        textBox1.Text = "Button Event";
    }
  }
}
```

Output :



Handling Mouse Events : You can handle various Mouse actions by using the events specified in the Control class. The following Example shows how to handle a simple MouseUp Event

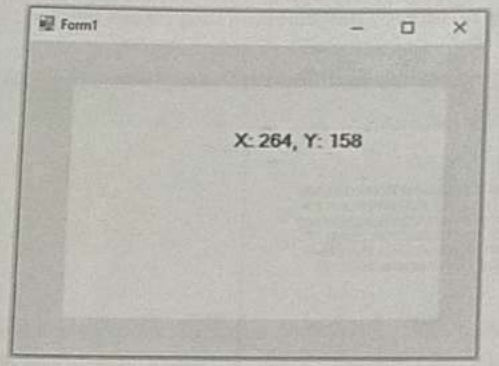
EXAMPLE-2

```
using System;
using System.Windows.Forms;
namespace MousePositionExample
{
public partial class Form1 : Form
  {
```



```
public Form1()
    {
InitializeComponent();
        panel1.MouseUp += new MouseEventHandler(panel1_MouseUp);
    }
private void panel1_MouseUp(object sender, MouseEventArgs e)
    {
      label1.Text = "X: " + e.X.ToString() + ", Y: " + e.Y.ToString();
    }
  }
}
```

Output :



Handling KeyBoard Events : Every modern programming language contains all necessary functions for handling KeyBoard related events. C# also provides us with three events Keypress, KeyUp, and KeyDown, which you can use to handle Keyboard events. Example3, below, shows the usage of the KeyUp Event.

EXAMPLE-3

```
using System;
using System.Windows.Forms;
namespace EventHandlingDemo
{
```

(vi) **ListBox Control :** ListBox control is used to display a list of items that the user can select one or more items from. ListBox control provides a user interface to display a list of items. Users can select one or more items from the list. A ListBox may be used to display multiple columns and these columns may have images and other controls.

ListBox Properties :

- **Name :** This property specifies the name of the ListBox control.
- **Items :** This property is a collection that contains the items displayed in the ListBox control.
- **SelectedIndex :** This property specifies the index of the currently selected item in the ListBox control. If no item is selected, this property is set to -1.
- **SelectedItem :** This property specifies the currently selected item in the ListBox control. If no item is selected, this property is set to null.
- **SelectionMode :** This property specifies the selection mode of the ListBox control. The options include Single (allows the user to select only one item), MultiSimple (allows the user to select multiple items), and MultiExtended (allows the user to select multiple items while holding down the Ctrl or Shift key).
- **DataSource :** This property is used to bind the ListBox control to a data source.
- **DisplayMember :** This property specifies the name of the property in the data source to display as the text of the ListBox items.
- **ValueMember :** This property specifies the name of the property in the data source to use as the value of the ListBox items.
- **Sorted :** This property specifies whether the items in the ListBox control are sorted alphabetically.
- **IntegralHeight :** This property specifies whether the ListBox control automatically resizes to fit the entire height of the items.
- **ItemHeight :** This property specifies the height of each item in the ListBox control.
- **Font :** This property specifies the font used to display the text of the items in the ListBox control.
- **ForeColor :** This property specifies the color of the text of the items in the ListBox control.

- **BackColor :** This property specifies the background color of the ListBox control.
- **BorderStyle :** This property specifies the border style of the ListBox control.
- **Enabled :** This property specifies whether the ListBox control is enabled or disabled.
- **Visible :** This property specifies whether the ListBox control is visible or hidden.
- **TabIndex :** This property specifies the tab order of the ListBox control.
- **MultiColumn :** This property specifies whether the ListBox control displays items in multiple columns.
- **ColumnWidth :** Gets or sets the width of columns in a multicolumn ListBox.
- **ScrollAlwaysVisible :** This property specifies whether the vertical scroll bar of the ListBox control is always visible.
- **HorizontalScrollbar :** Gets or sets a value indicating whether a horizontal scroll bar is displayed in the control.

ListBox Events

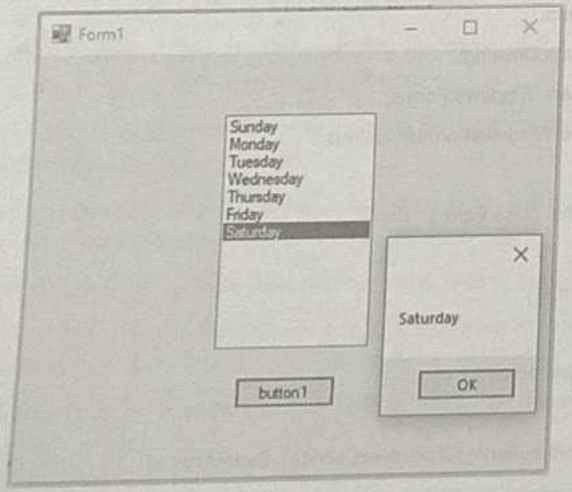
Event	Description
ItemCheck	(CheckedListBox only) Occurs when the check state of one of the listitems changes.
SelectedIndexChanged	Occurs when the index of the selected item changes.

Example Program

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
    }
private void button1_Click(object sender, EventArgs e)
```

```
        {
        foreach (Object obj in listBox1.SelectedItems){
        MessageBox.Show(obj.ToString());
              }
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        listBox1.Items.Add("Sunday");
        listBox1.Items.Add("Monday");
        listBox1.Items.Add("Tuesday");
        listBox1.Items.Add("Wednesday");
        listBox1.Items.Add("Thursday");
        listBox1.Items.Add("Friday");
        listBox1.Items.Add("Saturday");
        listBox1.SelectionMode = SelectionMode.MultiSimple;
        }
    }
}
```

Output :



(vii) **ComboBox Control :** ComboBox control is used to allow the user to select an item from a list of options or to type in a custom value. As the name implies, a combo box is a combination of TextBox and ListBox controls. Unlike the ListBox, it is not possible toselect more than one item in the list of items contained in a ComboBox and it is optionally possible to type new entries in the list in the TextBox part of the ComboBox.

Commonly, the ComboBox control is used to save space on a dialog. When the user clicks the arrow button to the right of the text box, a list box unfolds in which the user can make a selection. As soon as he or shedoes so, the list box disappears and the display returns to normal.

ComboBox Properties :

- **Name :** This property specifies the name of the ComboBox control.
- **Text :** This property specifies the text displayed in the ComboBox control when no item is selected.
- **Items :** This property is a collection that contains the items displayed in the ComboBox control.
- **ItemHeight :** Gets or sets the height of an item in the combo box.
- **SelectedIndex :** This property specifies the index of the currently selected item in the ComboBox control. If no item is selected, this property is set to -1.
- **SelectedItem :** This property specifies the currently selected item in the ComboBox control. If no item is selected, this property is set to null.
- **DroppedDown :** If set true, the dropped down portion of the ComboBox is displayed. By default, this value is false.
- **DropDownHeight :** Gets or sets the height in pixels of the drop-down portion of the ComboBox.
- **DropDownWidth :** Gets or sets the width of the of the drop-down portion of a combo box.
- **DropDownStyle :** This property specifies the drop-down style of the ComboBox control. The options include DropDown, DropDownList, and Simple.
- **DataSource :** This property is used to bind the ComboBox control to a data source.

- **DisplayMember :** This property specifies the name of the property in the data source to display as the text of the ComboBox items.
- **Enabled :** This property specifies whether the ComboBox control is enabled or disabled.
- **Visible :** This property specifies whether the ComboBox control is visible or hidden.
- **TabIndex :** This property specifies the tab order of the ComboBox control.
- **Font :** This property specifies the font used to display the text of the items in the ComboBox control.
- **ForeColor :** This property specifies the color of the text of the items in the ComboBox control.
- **BackColor :** This property specifies the background color of the ComboBox control.
- **AutoCompleteMode :** This property specifies the auto-complete mode of the ComboBox control. The options include None, Suggest, Append, and SuggestAppend.
- **AutoCompleteSource :** This property specifies the source of the auto-complete suggestions for the ComboBox control. The options include None, FileSystem, HistoryList, RecentlyUsedList, AllUrl, and AllSystemSources.
- **Sorted :** This property specifies whether the items in the ComboBox control are sorted alphabetically.
- **MaxDropDownItems :** This property specifies the maximum number of items displayed in the drop-down portion of the ComboBox control.

Combobox Events :

Events	Description
TextChanged	Occurs when the Text property changes
SelectedIndexChanged	Occurs when the selection in the list portion of thecontrol changed.
DropDown	Occurs when the list portion of the control is dropped down.

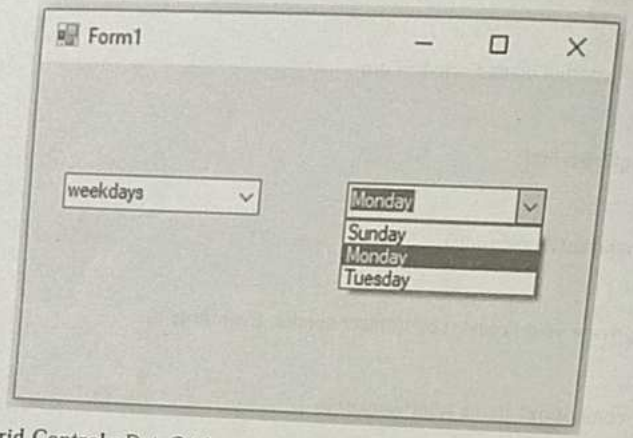
EXAMPLE

```
using System;
using System.Drawing;?
using System.Windows.Forms;
```



```
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
    }
private void Form1_Load(object sender, EventArgs e)
    {
comboBox1.Items.Add("weekdays");
comboBox1.Items.Add("year");
    }
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
comboBox2.Items.Clear();
if (comboBox1.SelectedItem.ToString() == "weekdays"){
comboBox2.Items.Add("Sunday");
comboBox2.Items.Add("Monday");
comboBox2.Items.Add("Tuesday");
      }
else if (comboBox1.SelectedItem.ToString() == "year"){
comboBox2.Items.Add("2012");
comboBox2.Items.Add("2013");
comboBox2.Items.Add("2014");
      }
    }
  }
}
```

Output :



(viii) **DataGrid Control :** DataGrid control is used to display and manipulate tabular data. DataGrid control displays data in a tabular form, rows and columns. A DataGrid control also provides functionality to sort, filter, scroll, and find records.

To display a table in the System.Windows.Forms. DataGrid at run time, use the Set DataBinding method to set the DataSource and DataMember properties to a valid data source. The following data sources are valid:

- DataTable
- DataView
- DataSet
- DataViewManager
- Single dimension array
- Any component that implements the IList interface

DataGrid Properties :

- **Name :** This property specifies the name of the DataGrid control.
- **DataSource :** This property specifies the data source for the DataGrid control. This can be a DataTable, DataView, or any other object that implements the IList interface.
- **DataMember :** This property specifies the name of the table or list within the data source to bind to the DataGrid control.

- **AllowSorting :** This property specifies whether the user can sort the data in the DataGrid control.
- **AlternatingBackColor :** This property specifies the background color of alternate rows in the DataGrid control.
- **AllowUserToAddRows :** This property specifies whether the user can add new rows to the DataGrid control.
- **AllowUserToDeleteRows :** This property specifies whether the user can delete rows from the DataGrid control.
- **AllowUserToResizeColumns :** This property specifies whether the user can resize columns in the DataGrid control.
- **AllowUserToResizeRows :** This property specifies whether the user can resize rows in the DataGrid control.
- **AutoGenerateColumns :** This property specifies whether the DataGrid control automatically generates columns based on the data source.
- **CaptionText :** This property specifies the caption text displayed at the top of the DataGrid control.
- **ColumnHeadersVisible :** This property specifies whether the column headers are visible in the DataGrid control.
- **RowHeadersVisible :** This property specifies whether the row headers are visible in the DataGrid control.
- **GridLineColor :** This property specifies the color of the grid lines in the DataGrid control.
- **GridLineStyle :** This property specifies the style of the grid lines in the DataGrid control.
- **ReadOnly :** This property specifies whether the user can edit the data in the DataGrid control.
- **SelectionMode :** This property specifies the selection mode of the DataGrid control. The options include CellSelect, FullRowSelect, and FullColumnSelect.
- **TabIndex :** This property specifies the tab order of the DataGrid control.
- **Font :** This property specifies the font used to display the text in the DataGrid control.

- **ForeColor :** This property specifies the color of the text in the DataGrid control.
- **BackColor :** This property specifies the background color of the DataGrid control.
- **AllowEditing :** This property specifies whether the user can edit the data in the DataGrid control.
- **AllowDeleting :** This property specifies whether the user can delete rows in the DataGrid control.
- **AllowAdding :** This property specifies whether the user can add new rows to the DataGrid control.

5.5 ENABLE, DISABLE, HIDE, AND SHOW THE CONTROLS IN WINDOWS APPLICATIONS

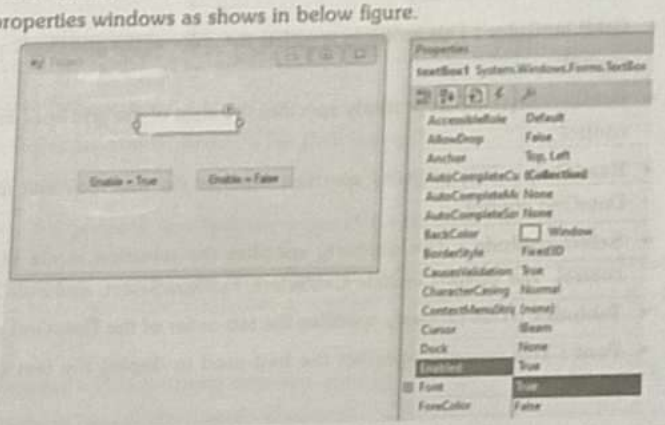
In C# Windows applications, you can enable, disable, hide, and show controls using their respective properties.

To enable or disable a control, you can use the **Enabled** property. When Enabled is set to true, the control is enabled and can be interacted with. When Enabled is set to false, the control is disabled and cannot be interacted with. Enabled Property of the control can be manipulated at design time or at run time.

```
button1.Enabled = false;  // disables the button
textBox1.Enabled = true;  // enables the text box
```

Enable/DisableTextbox at Designtime :

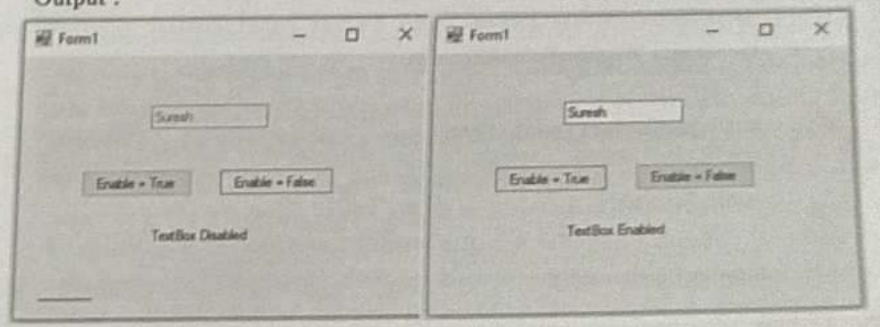
We can set manually textbox Enabled and Disable by assigning Boolean value true/ false in properties windows as shows in below figure.



Enable/Disable Textbox at Runtime

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace EnableDisableControls
{
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
    }
private void Form1_Load(object sender, EventArgs e)
    {
      label1.Text = "";
    }
private void btnTrue_Click(object sender, EventArgs e){
        textBox1.Enabled = true;
        label1.Text = "TextBox Enabled";
    }
private void btnFalse_Click(object sender, EventArgs e){
        textBox1.Enabled = false;
        label1.Text = "TextBox Disabled";
    }
  }
}
```

Output :

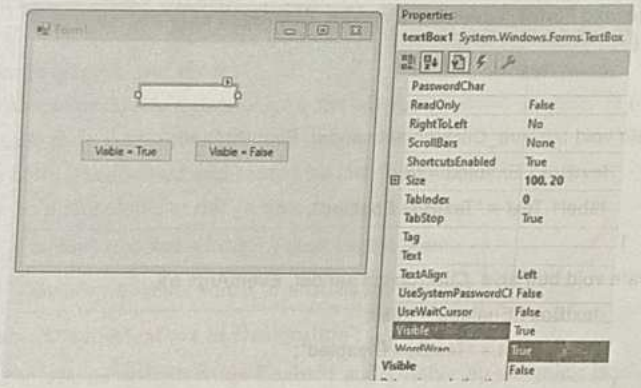


Hide, and show the Controls : To hide or show a control, you can use the **Visible** property. When Visible is set to true, the control is visible on the form. When Visible is set to false, the control is hidden from view. Visible Property of the control can be manipulated at design time or at run time.

```
label1.Visible = false;   // hides the label
checkBox1.Visible = true; // shows the check box
```

Hide/ Show TextBox at Designtime :

If you want to display the given TextBoxand its child controls, and then set the value of Visible property to true, otherwise set false.It is the easiest method to set the visibility of the button. Using the following figure.



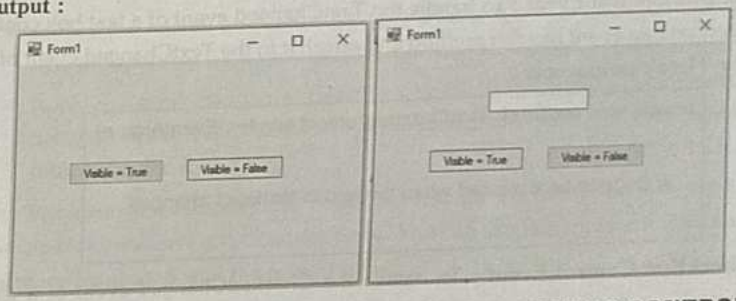
Hide/ Show TextBox at Runtime

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormUIControls
{
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
```



```
    }
private void Form1_Load(object sender, EventArgs e)
    {
    }
private void btnTrue_Click(object sender, EventArgs e){
        textBox1.Visible = true;
    }
private void btnFalse_Click(object sender, EventArgs e){
        textBox1.Visible = false;
    }
  }
}
```

Output :



5.6 HANDLING EVENTS GENERATED BY VARIOUS CONTROLS

An **event** is a notification that an action has occurred, such as a button being clicked, a key being pressed, or a text box losing focus. Handling events generated by various controls in a C# involves writing code that is executed when an event occurs.

An **event handler** is a method that contains the code that gets executed in response to a specific event that occurs in an application. Event handlers are used in graphical user interface (GUI) applications to handle events such as button clicks and menu selections, raised by controls in the user interface. A single event handler can be used to process events raised by multiple controls. An event can be associated with multiple event handlers, which will be invoked synchronously when the event occurs.

The event model is based on a publish-subscribe pattern in which a class

(publisher) triggers an event, while another class (subscriber) receives that event. An event handler is the subscriber that contains the code to handle specific events. For example, an event handler can be used to handle an event that occurs during the click of a command button in the UI.

No GUI application is complete without enabling actions. Even though arranging components is a significant issue, applying actions is also equally important. These actions are what instruct the program to act when something happens. For example, Mouse clicks, Keyboard presses, etc.

(i) **Button Control Event :** To handle the Click event of a button control, you can create an event handler method and attach it to the Click event of the button. Here's an example :

```
private void button1_Click(object sender, EventArgs e)
{
   // Code to be executed when button1 is clicked
}
```

(ii) **TextBox Control Event :** To handle the TextChanged event of a text box control, you can create an event handler method and attach it to the TextChanged event of the text box. Here's an example :

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
   // Code to be executed when the text in textBox1 changes
}
```

(iii) **CheckBox Control Event :** To handle the CheckedChanged event of a check box control, you can create an event handler method and attach it to the CheckedChanged event of the check box. Here's an example :

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
   // Code to be executed when the check state of checkBox1 changes
}
```

(iv) **RadioButton Control Event :** To handle the CheckedChanged event of a radio button control, you can create an event handler method and attach it to the CheckedChanged event of the radio button. Here's an example :


```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
   // Code to be executed when radioButton1 is checked or unchecked
}
```

(v) **ListBox Control Event :** To handle the SelectedIndexChanged event of a list box control, you can create an event handler method and attach it to the SelectedIndexChanged event of the list box. Here's an example :

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
   // Code to be executed when the selected item in listBox1 changes
}
```

Handling Button Events : As compared to the above, Event handling in C# is much more simplified. It is possible to handle various Mouse and Key related events quickly and in a more efficient manner. Learning will be easier if you understand the basic principles behind handling events, which are elaborated below

- Invoke the related event by supplying a custom method or event handler using the **+= operator** as shown here:b1.Click += new EventHandler(OnClick);
- Apply the event handler as described below. It must be in conformance with a delegate of the class System.EventHandler:public delegate void EventHandler (object sender, Event args)

The first argument indicates the object sending the event and the second argument contains information for the current event. You can use this argument object to handle functions associated with the related event.

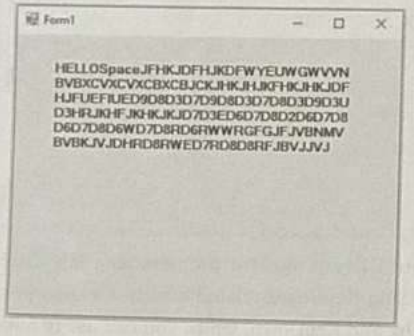
Examplebelow illustrates how to print "Button Event" inside a Textbox when a Button is clicked :

EXAMPLE-1

```
using System;
using System.Windows.Forms;
namespace ButtonEventExample
{
public partial class Form1 : Form
  {
```

```
public partial class Form1 : Form
  {
public Form1()
    {
InitializeComponent();
this.KeyUp += new KeyEventHandler(OnKeypress);
    }
private void OnKeypress(object sender, KeyEventArgs e)
    {
        label1.Text += e.KeyCode.ToString();
    }
  }
}
```

Output :

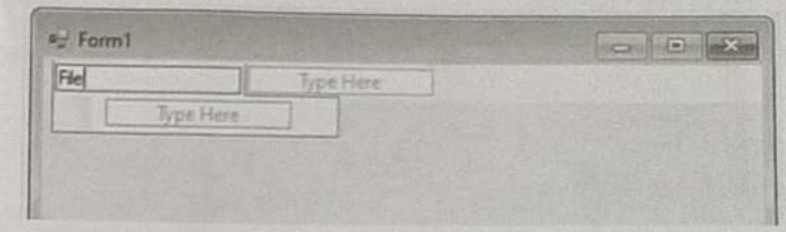


5.7 CREATION OF MENUS AT DESIGN TIME

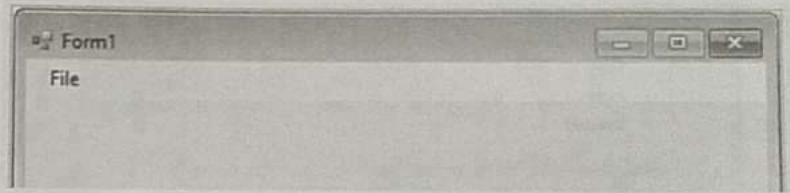
We can create Menus at design time using **MenuStrip** control using a Forms designer at design-time. To create a MenuStrip control at design-time, you simply drag and drop a MenuStrip control from Toolbox to a Form in Visual Studio. After you drag and drop a MenuStrip on a Form, the MenuStrip1 is added. Once a MenuStrip is added on the Form, you can add menu items and sub menu items and set their properties and events using **ToolStripMenuItem** control.

Menu Item : To add new items to the menu, locate the box that says Type Here in grayletters and click there. Type the text of the new menu item you want to add.

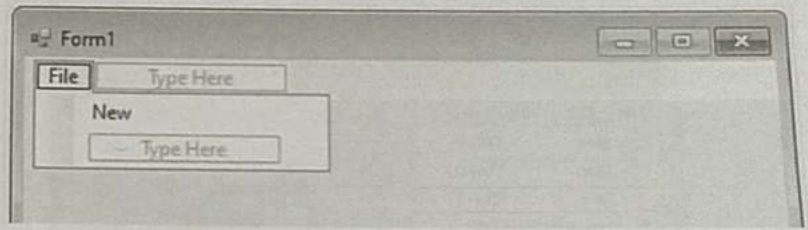
Example : Click inside of the area at the top, where it says "Type Here". Now type the word **File**.



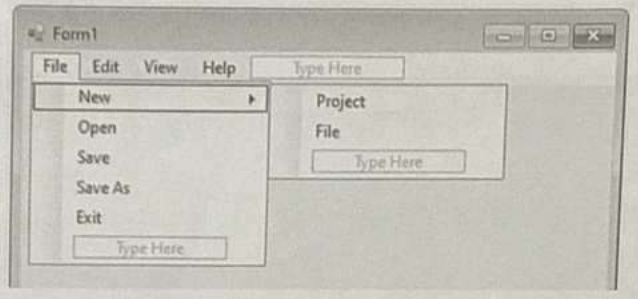
Hit the Enter key on your keyboard and your menu will look like this :

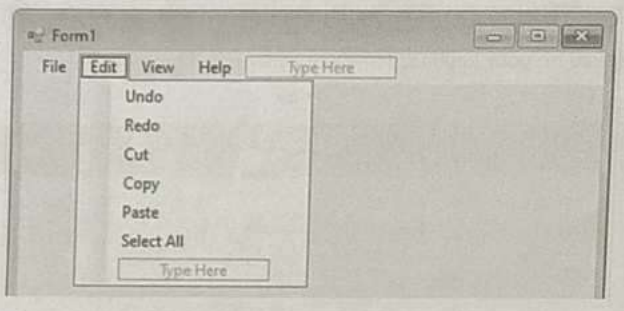


To add items to your File menu, click inside of the second "Type Here" below the File menu area pictured above. Now type the word **New**. Hit the Enter key on your keyboard to add the menu item :



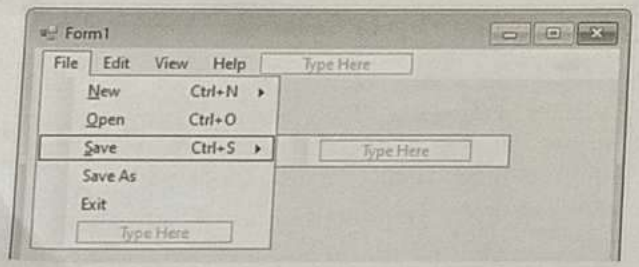
To add a menu item **"Edit"**,type Edit into the grayed Text box "Type Here" besides the File Menu. Repeat the above process to add the other main menu items and sub menu items.





Short Cut Keys : These are combination of keys which are used to select the action performed by menu item by pressing them from the keyboard.

If you want to display the shortcut keys in the menu item text, you can use the "&" character as a prefix for the character key. For example, to display "Ctrl+S" as the shortcut for a menu item, set the Text property to "&Save" and the ShortcutKeys property to "Ctrl+S".

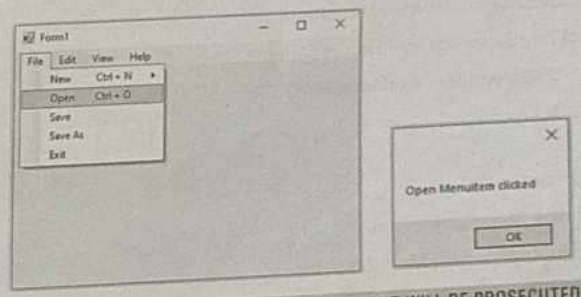


Click Handlers : To add actions to the menu items in your MenuStrip, double clickthem and then a Click event handler for that ToolStripMenuItem will be inserted.Then, you can add your custom C# code to thatmethod's body.

EXAMPLE

```
using System;
using System.Windows.Forms;
namespace MenusAtDesignTime
{
public partial class Form1 : Form
  {
public Form1(){
InitializeComponent();
    }
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
MessageBox.Show("New Menuitem clicked");
    }
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
MessageBox.Show("Open Menuitem clicked");
    }
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
Close();
    }
  }
}
```

Output :



5.8 CREATION OF MENUS AT RUN TIME

In C#, you can create menus at runtime using the **MenuStrip** and **ToolStripMenuItem** classes. This can be useful when you want to dynamically create menus based on some condition, or when you want to give the user the ability to customize the menus.

MenuStrip Class : The MenuStrip class is a Windows Forms control that provides a menu bar for an application. The MenuStrip control is typically used to create a standard menu bar with drop-down menus, similar to the menus found in many desktop applications. The MenuStrip control can be added to a form using the form's Controls property, and it can be customized using its properties and methods.

ToolStripMenuItem Class : The ToolStripMenuItem class is a Windows Forms control that represents a menu item in a MenuStrip or ContextMenuStrip control. A ToolStripMenuItem can contain a text label, an image, a keyboard shortcut, a drop-down menu with sub-items and a Click event.

Individual menu items going to menu strip that go to menustrip. They can Represent top level menus & sub menus.

Example Program

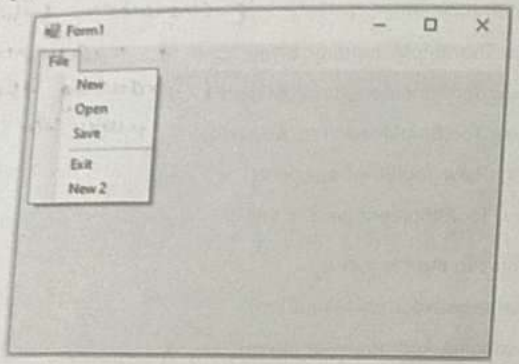
```
using System;
using System.Windows.Forms;
namespace MenusAtRunTime
{
public partial class Form1 : Form
  {
 private MenuStrip menuStrip1;
 private ToolStripMenuItem fileMenu;
 private ToolStripMenuItem newMenuItem;
 private ToolStripMenuItem openMenuItem;
 private ToolStripMenuItem saveMenuItem;
 private ToolStripSeparator toolStripSeparator1;
 private ToolStripMenuItem exitMenuItem;
 public Form1()
    {
 InitializeComponent();
 // Create a new MenuStrip control
       menuStrip1 = new MenuStrip();
```



```
// Create a new file menu with some menu items
fileMenu = new ToolStripMenuItem("&File");
newMenuItem = new ToolStripMenuItem("&New");
openMenuItem = new ToolStripMenuItem("&Open");
saveMenuItem = new ToolStripMenuItem("&Save");
toolStripSeparator1 = new ToolStripSeparator();
exitMenuItem = new ToolStripMenuItem("E&xit");

// Add the menu items to the file menu
fileMenu.DropDownItems.Add(newMenuItem);
fileMenu.DropDownItems.Add(openMenuItem);
fileMenu.DropDownItems.Add(saveMenuItem);
fileMenu.DropDownItems.Add(toolStripSeparator1);
fileMenu.DropDownItems.Add(exitMenuItem);

// Add the file menu to the MenuStrip control
menuStrip1.Items.Add(fileMenu);

// Add the MenuStrip control to the form
Controls.Add(menuStrip1);

// Set the event handler for the exit menu item
exitMenuItem.Click += new EventHandler(exitMenuItem_Click);

// Add a new menu item at runtime
ToolStripMenuItem newMenuItem2 = new ToolStripMenuItem("&New 2");
fileMenu.DropDownItems.Add(newMenuItem2);
    }
private void exitMenuItem_Click(object sender, EventArgs e){
// Close the form
Close();
    }
  }
}
```

& character before f makes f a short key Alt+F. underlining the f in the menu item.

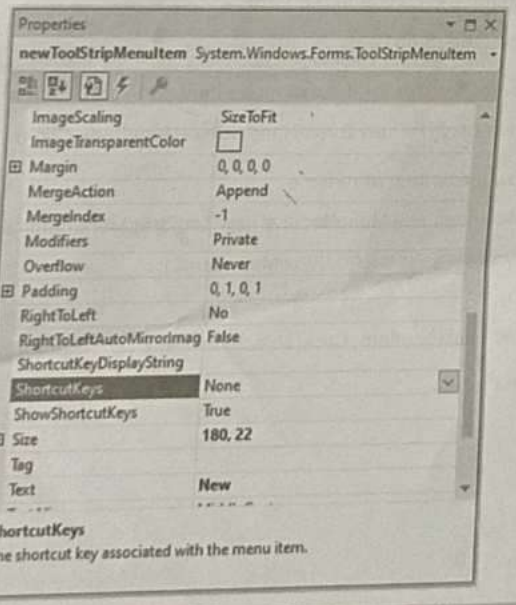
Output :



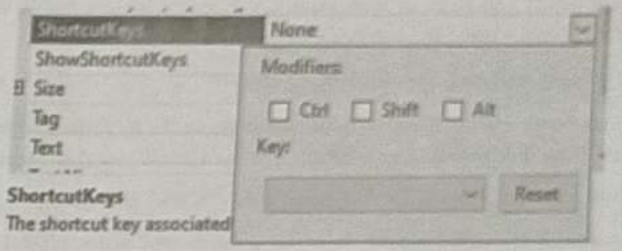
5.9 CREATE SHORT CUT KEYS FOR PULL DOWN MENUS

In C#, you can create keyboard shortcuts for pull-down menus at design time and also at run time.

Creation of Short Cut Keys at Design Time : The key combination shortcuts are just easy to add. Click on your **New** menu item to select it. Locate the **ShortcutKeys** Property in the Properties Window :

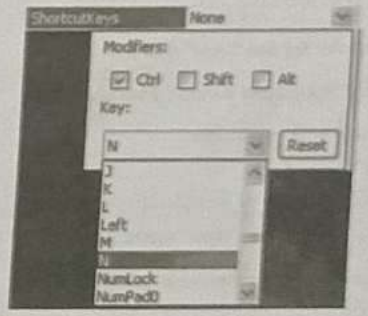


At the moment, it's set to None. Click the down arrow to see the following options :

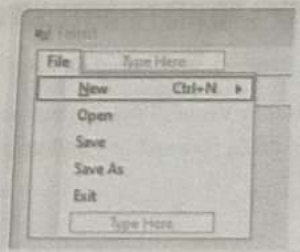


The Modifiers are the CTRL, Shift, and ALT keys. You can select one or all of these, if you want. To activate a shortcut, you would then have to hold down these keys first. So if you want your users to hold down the CTRL and Shift keys, plus a letter or symbol, then you would check the relevant Modifier boxes above.

The letters and symbols can be found on the Key drop down list. Click the down arrow to see the following :



In the image above, we've gone for the CTRL modifier, and the letter "N". Clicking back on the menu, here's what it now looks like :



As you can see, the shortcuts for the New menu item are an Underline, and Ctrl + N.

Creation of Short Cut Keys at Run Time : At Runtime we can create shortcut keys for pull down menus using the following steps:

(i) Add a menu item to your form's menu strip.

(ii) Set the menu item's "ShortcutKeys" property to the keyboard combination you want to use. For example, you could set it to "Ctrl+N" for the "New" menu item.

(iii) Create an event handler for the menu item's "Click" event.

(iv) Add code to the event handler to perform the desired action when the menu item is clicked.

Here's an example code snippet :

```
// Add a "New" menu item to the menu strip
ToolStripMenuItem newItem = new ToolStripMenuItem("New");
newItem.ShortcutKeys = Keys.Control | Keys.N;
menuStrip1.Items.Add(newItem);

// Create an event handler for the "New" menu item
private void newItem_Click(object sender, EventArgs e){
// Code to create a new item goes here
}

// Attach the event handler to the "New" menu item
newItem.Click += new EventHandler(newItem_Click);
```

In the above example, pressing the "Ctrl+N" keyboard combination will trigger the "newItem_Click" event handler, which can be used to perform any desired action.

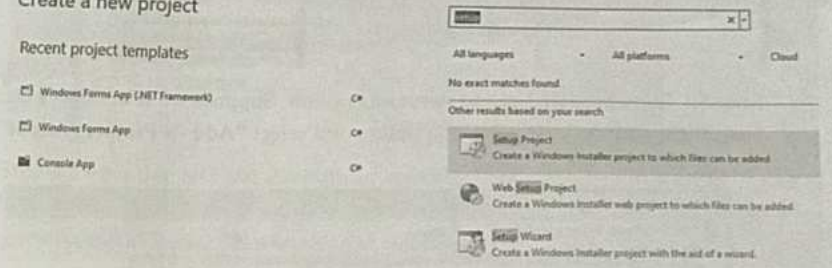
5.10 STEPS TO DEPLOY AND DISTRIBUTE WINDOWS APPLICATION

Here are the general steps to deploy and distribute a C# Windows application :

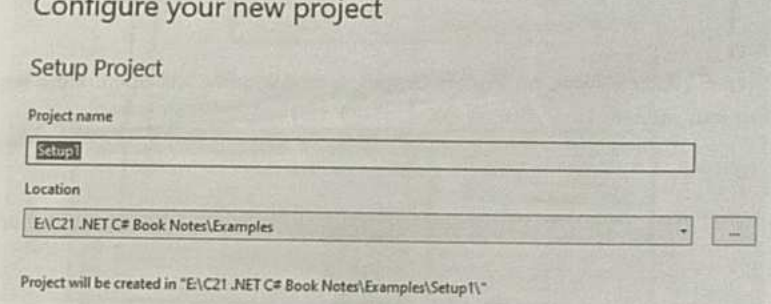
Step-1 : Build your application in Visual Studio using the **"Release"** configuration. This will produce a set of executable files and any additional files that your application requires.

Step-2 : Test your application to make sure that it runs correctly and all features are working as intended.

Step-3 : Create a setup project in Visual Studio by choosing **"Add—>NewProject"** in Solution Explorer and selecting **"Setup Project"** under "All Project Types". Then, select "Setup Project" to create a new setup project.



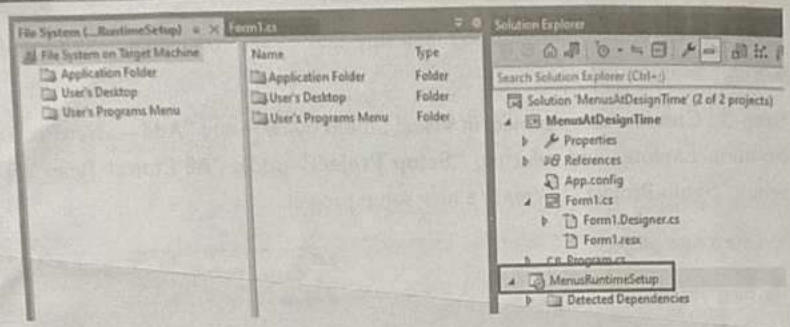
Step-4 : Configure the Setup project by selecting the desired project name, location.Click **"Create"** to create the project.



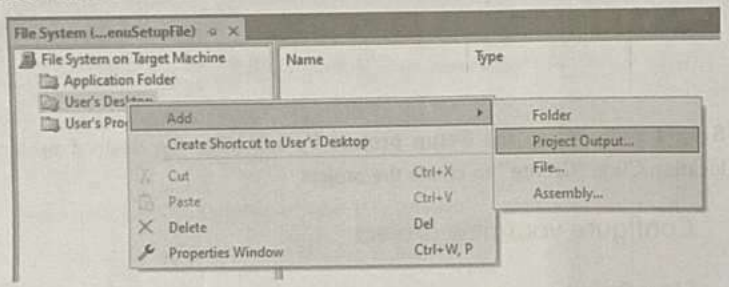
Here we find three options as following :

1. Application Folder
2. User's Desktop
3. User's Program Menu

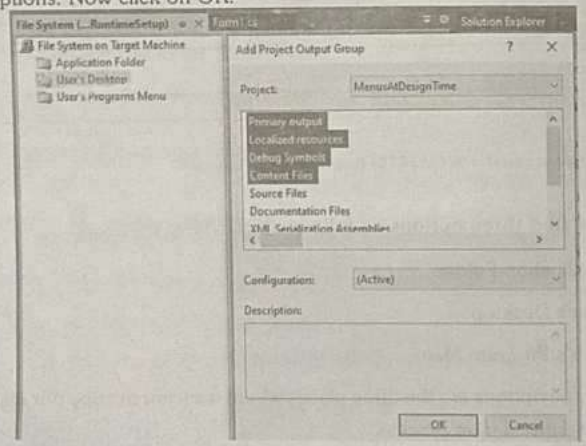
These three options are the three places where we want to copy our installer setup files during the installation process.



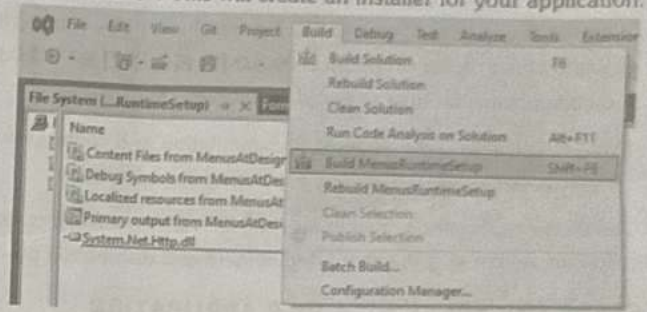
Step-5 : From here we have to select one option. Suppose we select User's Desktop. Then Right click on User's Desktop option and select **"Add -> Project Output".** The window will look like as below.



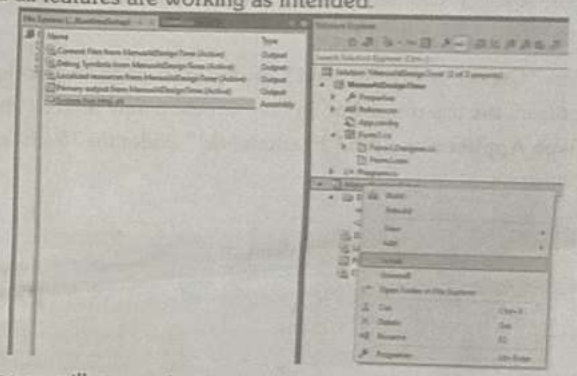
Step-6 : After clicking on Project Output, a new window will open. Here we select first four options. Now click on OK.



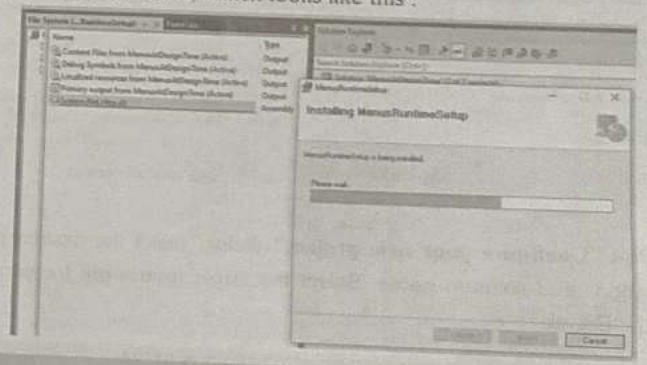
Step-7 : After clicking on OK, build the setup project by choosing **"Build -> Build setup project name"**. This will create an installer for your application.



Step-8 : After this, let's test the installer setup by right clicking on the setup project in Solution Explorer and click on Install menu item. Make sure that it installs your application correctly and all features are working as intended.



The setup process will start, which looks like this :



Step-9 : After completing the installation go on desktop there you will find all files you have included in the setup on the desktop.

Step-10 : Distribute the installer to your users. This can be done by creating a download link or by providing the installer on physical media like CD/DVD, or USB drive. Make sure to include instructions on how to install and run the application.

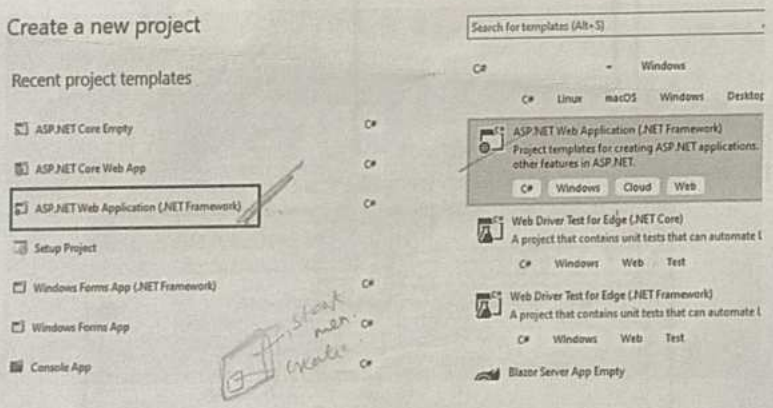
Step-11 : Provide end-users with support documentation and contact information in case they encounter issues while installing or running the application.

Step-12 : Update the application regularly with bug fixes and new features, and distribute the updates to end-users using the same deployment and distribution process.

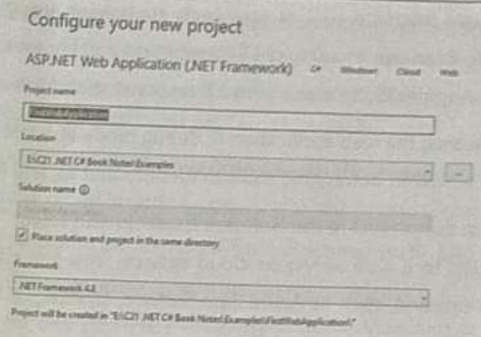
5.11 STEPS FOR CREATING A WEB APPLICATION

Here are the general steps for creating a C# web application :

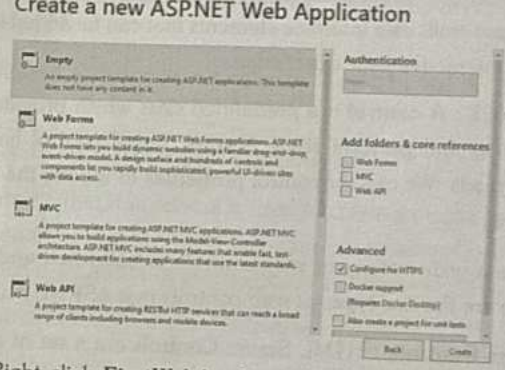
1. Open Microsoft Visual Studio IDE from the Start menu or by double-clicking the desktop icon.
2. Click on **"Create a new project"** on the start page (or) go to File—>New—> Project from the top menu bar. In the "Create a new project" dialog, select **"ASP .NETWeb Application (.NET Framework)"** under the "Web" category, and then click "Next."



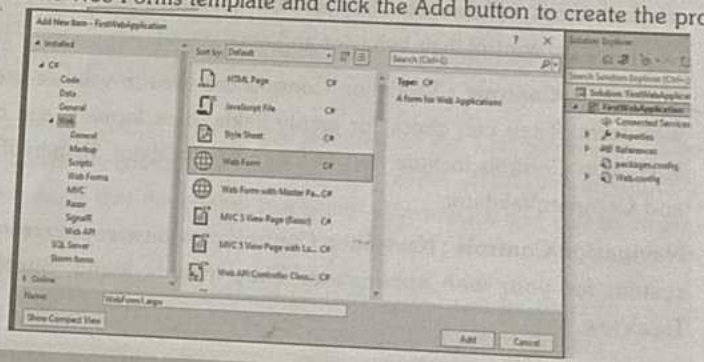
3. In the **"Configure your new project"** dialog, select the desired project name, location, and solution name. Select the target framework for your project and click "Create".



4. You can create a new ASP .NET web application choosing from different web templates, such as **"Empty,"** "Web Forms," "MVC", or "Web API" etc. You can also choose to include "Authentication" and "HTTPS" support.Click **"Create"** to create the project.



5. Next, Right click **FirstWebApplication** solution explorer and add new item to select the Web Forms template and click the Add button to create the project.



6. Design the user interface (UI) of your web application using HTML, CSS, and JavaScript. You can use a variety of UI frameworks and libraries, such as Bootstrap, jQuery, and AngularJS, to make your UI responsive and interactive.
7. Press **F5** to launch the web application in debug mode and test it in a web browser. Use the IDE's built-in debugging tools to identify and fix any issues in your code.
8. Right-click on the project in the Solution Explorer and select **"Publish"** to deploy the application to a web server or cloud service. You can choose from different deployment options, such as Azure App Service, Docker, or FTP.

5.12 USAGE OF TEXTBOX, LABEL, BUTTON, CHECKBOX, RADIO BUTTON, LISTBOX, DROPDOWNLIST, DATAGRID, HYPERLINK, IMAGE, PANEL, AND HIDDENFIELD WEB CONTROLS

Web controls are essential components of C# ASP .NET web application development. These controls are pre-built user interface elements that can be added to a web form to provide a rich and interactive user experience.

Controls in ASP.NET : A control is a predefined class which provides user interface which we can access and perform some actions. Every control has some specific properties and methods. We can set control properties to modify the appearance (or) behavior of controls.

ASP.NET provides a wide variety of web controls to build rich and interactive web applications. Here are the five types of web controls that ASP.NET offers :

1. **HTML Server Controls :** HTML Server Controls are a set of standard controls that represent HTML elements such as text boxes, checkboxes, radio buttons, labels, buttons and images. They provide server-side events and properties that can be set to control their behavior and appearance.
2. **Validation Controls :** Validation Controls are used to validate user input on a web form. They can check for empty fields, data formats, and other criteria. Validation Controls include RegularExpressionValidator, RequiredFieldValidator, and CompareValidator.
3. **Navigation Controls :** Navigation Controls provide a way to create a navigation system for your web application. They include Menu, SiteMapPath, and TreeView.

4. **Data Controls :** Data Controls are used to display data from a database or other data source on a web form. They can be used to bind to a single record, a collection of records, or a hierarchical structure. Examples of Data Controls include GridView, ListView, Repeater, and DataList.
5. **Login Controls :** Login Controls provide login functionalityto authenticate users on a web form. They include Login, LoginStatus, and CreateUserWizard.

All the ASP.NET controls are inherited from **System. Web. UI. Web Controls. Web Control** namespace.

General syntax for any ASP.NET controls :

```
<asp:ControlName Property1="Value1" Property2="Value2" runat="server"></
asp:ControlName>
```

Example :

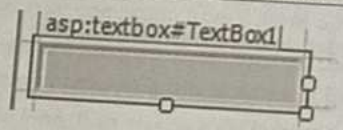
```
<asp:Label ID="Label1" runat="server" Text="Hello World" Font-Size="18"></
asp:Label>

<asp:Button ID="Button1" runat="server" Text="Click Me" OnClick="Button1_Click"></
asp:Button>
```

Some of the ASP .NET controls are described below :

(i) **TextBox Control :** Text Box control is used to create an input field where the user can enter text or numeric values. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Syntax :

```
<asp:TextBox ID= "TextBox1" runat= "server"> </asp:TextBox>
```



The **ID attribute** is used to set a unique identifier for the Text Box control, which can be used to access the control from the code-behind file. The **runat attribute** is set to "server" to indicate that the control is a server-side control and can be manipulated from the server-side code.

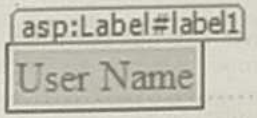
The Text Box control provides several properties that can be used to customize its appearance and behavior. Some of the commonly used properties are :

- **Text :** Gets or sets the text displayed in the Text Box control.
- **MaxLength :** Gets or sets the maximum number of characters that can be entered in the Text Box control.
- **AutoPostBack :** Gets or sets a value indicating whether the page should be posted back to the server when the Text Box control loses focus.
- **Enabled :** Gets or sets a value indicating whether the Text Box control is enabled or disabled.
- **ReadOnly :** Gets or sets a value indicating whether the Text Box control is read-only or editable.

(ii) **Label Control :** Label control is used to display text on a web form. The mostly used attribute for a label control is 'Text', which implies the text displayed on the label.

Syntax :

```
<asp:Label ID= "label1" runat= "server" Text= "User Name"></asp:Label>
```



Some of the commonly used Label control properties are :

- **Text :** Gets or sets the text displayed by the Label control.
- **ForeColor :** Gets or sets the color of the text displayed by the Label control.
- **BackColor :** Gets or sets the background color of the Label control.
- **Font :** Gets or sets the font used to display the text in the Label control.
- **BorderStyle :** Gets or sets the border style of the Label control.

(iii) **Button Control :** Button control is used to create a clickable button on a web form. ASP.NET provides three types of button control :

Button : It displays text within a rectangular area.

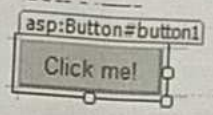
LinkButton : It displays text that looks like a hyperlink.

ImageButton : It displays an image.

When a user clicks a button, two events are raised: Click and Command.

Syntax :

```
<asp:Button ID= "button1 "runat="server" Text="Click me!" OnClick=" button1_
Click"></asp:Button>
```



Some of the commonly used Button control properties are :

- **Text :** Gets or sets the text displayed on the button.
- **ForeColor :** Gets or sets the color of the text displayed on the button.
- **BackColor :** Gets or sets the background color of the button.
- **Font :** Gets or sets the font used to display the text on the button.
- **Enabled :** Gets or sets a value indicating whether the button is enabled or disabled.

Example :

```
<asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick="
btnSubmit_Click" ></asp:Button>
```

In the code-behind file, the OnClick event is used to handle the click event of the Button control :

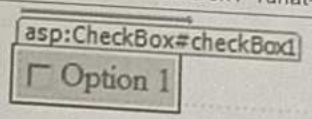
```
protected void btnSubmit_Click(object sender, EventArgs e){
  // Code to handle button click event
}
```

(iv) **CheckBox and RadioButton Control :** CheckBox control is used to allow the user to select multiple options from a list. RadioButton control is used to allow the user to select only one option from a list of options.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

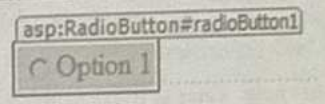
Syntax of CheckBox Control :

```
<asp:CheckBox ID="checkBox1" runat="server" Text="Option 1"></asp:CheckBox>
```



Syntax of RadioButton Control :

```
<asp:RadioButton ID="radioButton1" runat="server" Text="Option 1" Group Name="
group1"></asp:RadioButton>
```



Some of the commonly used properties of CheckBox and RadioButton controls are:

- **Text :** Gets or sets the label displayed next to the checkbox or radiobutton.
- **Checked :** Gets or sets a value indicating whether the checkbox or radiobutton is checked or not.

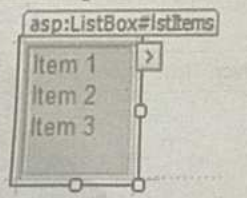
(v) **List Controls :** List controls are used to display lists of items on a web page.ASP.NET provides the following List controls.

- ListBox,
- DropDownList,
- CheckBoxList,
- RadioButtonlist,
- BulletedList

ListBox control is used to display a list of items from which the user can select one or more items.

Syntax of ListBox control :

```
<asp:ListBox ID="lstItems" runat="server" SelectionMode="Multiple">
<asp:ListItem Text="Item 1" Value="1"></asp:ListItem>
<asp:ListItem Text="Item 2" Value="2"></asp:ListItem>
<asp:ListItem Text="Item 3" Value="3"></asp:ListItem>
</asp:ListBox>
```



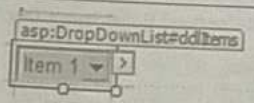
Some of the commonly used ListBox control properties are :

- **SelectionMode :** Gets or sets the selection mode for the ListBox control. It can be set to Single, Multiple, or Extended.
- **SelectedIndex :** Gets or sets the index of the selected item in the ListBox control.
- **SelectedValue :** Gets or sets the value of the selected item in the ListBox control.

DropDownList control is used to display a list of items from which the user can select one item.

Syntax of DropDownList control :

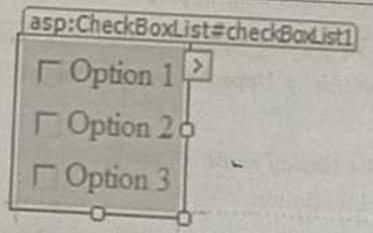
```
<asp:DropDownList ID="ddlItems"runat="server">
<asp:ListItem Text="Item 1"></asp:ListItem>
<asp:ListItem Text="Item 2"></asp:ListItem>
<asp:ListItem Text="Item 3"></asp:ListItem>
</asp:DropDownList>
```



CheckBoxList control is used to display a list of options with checkboxes on a web page.

Syntax of CheckBoxList control :

```
<asp:CheckBoxList ID="checkBoxList1"runat="server" RepeatDirection="Vertical">
<asp:ListItem Text="Option 1" Value="1"></asp:ListItem>
<asp:ListItem Text="Option 2" Value="2"></asp:ListItem>
<asp:ListItem Text="Option 3" Value="3"></asp:ListItem>
</asp:CheckBoxList>
```



Here are some of the most commonly used properties of the HyperLink control :

- **NavigateUrl :** Specifies the URL that the hyperlink should point to.
- **Target :** Specifies the target window or frame where the linked document will open. For example, "_blank" will open the linked document in a new window.
- **Text :** Specifies the text that should be displayed for the hyperlink.
- **ToolTip :** Specifies the text that appears when the mouse pointer hovers over the hyperlink.
- **CssClass :** Specifies one or more CSS classes to apply to the hyperlink.
- **ImageUrl :** Specifies the URL of an image to display as the hyperlink.
- **Enabled :** Specifies whether the hyperlink is enabled or disabled.

(viii) **Image Control :** The Image control is used for displaying images on the web page, or some alternative text, if the image is not available. It allows developers to specify the source of the image, as well as various properties such as size, alignment, and border.

Syntax :

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/Images/rose_pic .jpg"
Alternate Text="My Image" Width="200" Height=" 200" BorderStyle="Solid"
BorderWidth="1px" BorderColor="Gray"/>
```

Here are some of the most commonly used properties of the Image control:

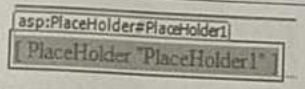
- **ImageUrl :** Specifies the URL of the image to display.
- **AlternateText :** Specifies the alternate text for the image, which is used by screen readers and other assistive technologies.

- **Width and Height :** Specifies the width and height of the image, in pixels.
- **BorderStyle, BorderWidth, and BorderColor :** Specifies the style, width, and color of the border around the image.
- **Align :** Specifies the horizontal alignment of the image within its container.
- **VAlign :** Specifies the vertical alignment of the image within its container.
- **CssClass :** Specifies one or more CSS classes to apply to the image.

(ix) **PlaceHolder Control :** Placeholder control is a container control that allows developers to add and manipulate child controls dynamically at runtime. It does not render any HTML of its own, but instead serves as a placeholder to load other controls at a specific place on the web page.

Syntax :

```
<asp:PlaceHolder ID="PlaceHolder1" runat="server"></asp:PlaceHolder>
```



Here are some of the most commonly used properties of the Placeholder control :

- **ID :** Specifies a unique identifier for the control, which can be used to reference it in code-behind or using data binding.
- **Visible :** Specifies whether the control should be visible on the page or hidden.
- **EnableViewState :** Specifies whether the control should save its state between postbacks.
- **CssClass :** Specifies one or more CSS classes to apply to the control.
- **Controls :** Provides access to the collection of child controls contained within the Placeholder.

The Controls property is particularly useful for dynamically adding child controls to the Placeholder. Here's an example of how to add a Button control to a Placeholder programmatically :

```
Button myButton = new Button();
myButton.ID = "myButton";
myButton.Text = "Click me!";
PlaceHolder1.Controls.Add(myButton);
```

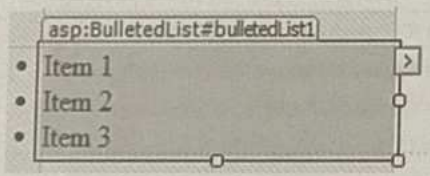
Some of the commonly used CheckBoxList control properties are :

- **RepeatLayout :** Gets or sets the layout of the CheckBoxList control. It can be set to Table or Flow.
- **RepeatDirection :** Gets or sets the direction in which the CheckBoxList control is rendered. It can be set to Vertical or Horizontal.
- **SelectedValue :** Gets or sets the selected value of the CheckBoxList control.
- **DataTextField :** Gets or sets the name of the field in the data source to bind to the CheckBoxList control for displaying text.
- **DataValueField :** Gets or sets the name of the field in the data source to bind to the CheckBoxList control for storing values.

BulletedList control is used to display a bulleted list of items on a web page.

Syntax of BulletedList Control :

```
<asp:BuletedList ID="buletedList1" runat="server" BulletStyle="Disc">
<asp:ListItem Text="Item 1"></asp:ListItem>
<asp:ListItem Text="Item 2"></asp:ListItem>
<asp:ListItem Text="Item 3"></asp:ListItem>
</asp:BuletedList>
```



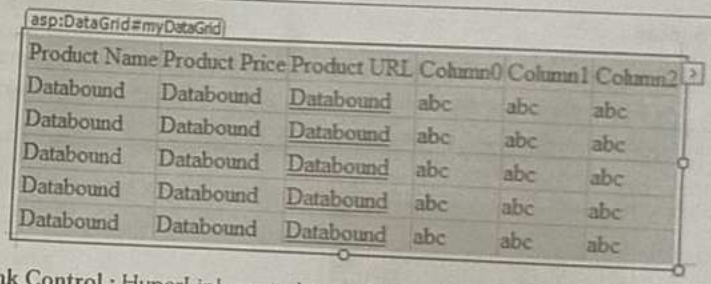
Some of the commonly used BulletedList control properties are :

- **BulletStyle :** Gets or sets the style of the bullets used in the BulletedList control. It can be set to Numbered, LowerAlpha, UpperAlpha, LowerRoman, UpperRoman, or Disc.
- **DisplayMode :** Gets or sets the display mode for the BulletedList control. It can be set to HyperLink, Text, or LinkButton.
- **DataSource :** Gets or sets the data source for the BulletedList control.

(vi) **DataGrid Control :** The DataGrid control is used to display data in a tabular format, allowing users to view and interact with large sets of data. It can be bound to various data sources, including databases, XML files, and collections, making it a versatile tool for data display and manipulation.

Here's the syntax for adding a DataGrid control to an ASP.NET web form :

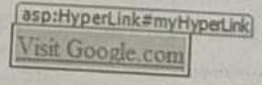
```
<asp:DataGrid id ="myDataGrid" runat="server">
<Columns>
<asp:BoundColumn DataField="ProductName" HeaderText="Product Name"/>
<asp:BoundColumn DataField="ProductPrice" HeaderText="Product Price"/>
<asp: Hyper LinkColumn DataNavigateUrlField=" ProductURL" DataTextField="
ProductURL" HeaderText="Product URL"/>
</Columns>
</asp:DataGrid>
```



(vii) **HyperLink Control :** HyperLink control is used to create clickable hyperlinks on a web page. It allows developers to specify the URL that the hyperlink should point to, as well as the text that should be displayed for the link. The HyperLink control is like the HTML <a> element.

Syntax :

```
<asp:HyperLink id="myHyperLink"runat="server" NavigateUrl="https://www. google.
com" Text="Visit Google.com">
</asp:HyperLink>
```

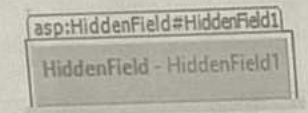


(x) **HiddenField Control :** HiddenField control is used to store a value on the client side that can be accessed and modified by server-side code. It is typically used to store data that needs to be persisted between postbacks, but should not be visible to the user.

HiddenField controls are not encrypted or protected and can be changed by any one. However, from the security point of view, this is not suggested. ASP.NET uses HiddenField control for managing the ViewState. So, dont store any important or confidential data like password and credit card details with this control.

Syntax :

```
<asp:HiddenField ID="HiddenField1" runat="server" Value="some value"/>
```



EXAMPLE

Creating a Simple Registration form using ASP.NET server controls :

1. Go to File-->New-->Project--> and select ASP.NET Web Application (.NET Framework) to create a new web application project.
2. Right click on web application project—>Add—>Add New Item—>Web Form->write Web form name with **Registration.aspx** extension and click on Add button.

Registration.aspx

Aspx file defines the user interface of webpage while the ASPX-cs file Contains the code that controls the behaving of the interface.

```
<%@PageLanguage=" C#"Auto EventWireup=" true"CodeBehind=" Registration.
aspx.cs" Inherits="RegistrationForm.Registration"%>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<headrunat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<table class="auto-style1">
```

This property defines whether the Asp.net framework automatically wire up events like pageload, buttonclick, etc to their respective event handlers in the code behind file.


```
<tr>
<td>Name :</td>
<td><asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></td>
</tr>
<tr>
<td>Password</td>
<td><asp:TextBox ID="TextBox2" runat="server" TextMode="Password"></
asp:TextBox></td>
</tr>
<tr>
<td>Confirm Password</td>
<td><asp:TextBox ID="TextBox3" runat="server" TextMode="Password"></
asp:TextBox></td>
</tr>
<tr>
<td>City</td>
<td>
<asp:DropDownList ID="DropDownList1" runat="server">
<asp:ListItem Text="Select City" Value="select" Selected="True"></asp:ListItem>
<asp:ListItem Text="Bangalore" Value="Bangalore"></asp:ListItem>
<asp:ListItem Text="Mysore" Value="Mysore"></asp:ListItem>
<asp:ListItem Text="Hyderabad" Value="Hyderabad"></asp:ListItem>
</asp:DropDownList></td>
</tr>
<tr>
<td>Gender</td>
<td><asp:RadioButtonList ID="RadioButtonList1" runat="server">
<asp:ListItem>Male</asp:ListItem>
<asp:ListItem>Female</asp:ListItem>
</asp:RadioButtonList></td>
```

when user interacts with page (eg by clicking button, the browser sends a request back to server. The server then executes the corresponding server side code in the code behind file to handle the event.

Defines the layout and components of webpage Containing markup and ASP.NET server controls).

- aspx.cs file contains the server side logic for the page including event handler that respond to user interaction.

```
</tr>
<tr>
<td>Gmail</td>
<td><asp:TextBox ID="TextBox4" runat="server"></asp:TextBox>
</td>
</tr>
<tr>
<td><asp:Button ID="Button1" runat="server" Text="Button"/></td>
</tr>
</table>
</div>
</form>
</body>
</html>
```

Name :
Password
Confirm Password
City Select City
Gender Male Female
Gmail
Button

Registration.aspx.cs

```
using System;
namespace RegistrationForm
{
public partial class Registration : System.Web.UI.Page
  {
```



```
protected void Page_Load(object sender, EventArgs e){
    }
protected void Button1_Click(object sender, EventArgs e){
Response.Write("Name: " + TextBox1.Text);
Response.Write("Password: " + TextBox2.Text);
Response.Write("City Selected: " + DropDownList1.SelectedValue);
Response.Write("Gender Selected: " + RadioButtonList1.SelectedValue);
Response.Write("Mail: " + TextBox4.Text);
    }
  }
}
```

https://localhost:44357/Registration.aspx

Name: Raj kiran, Password: raju@123, City Selected: Hyderabad, Gender Selected: Male, Mail: raju_kiran@gmail.com

Name : Raj kiran
Password
Confirm Password
City Hyderabad
Gender Male Female
Gmail raju_kiran@gmail.com
Button

5.13 DATA VALIDATION CONTROLS

Data validation controls are a powerful toolset that allows developers to create web applications that can validate user input and prevent incorrect or malicious data from being submitted. These controls are a fundamental aspect of ASP .NET web development, and they play a critical role in ensuring the security and integrity of web applications.

Each control is designed to perform a specific type of data validation and can be customized to meet the specific needs of a web application.

ASP.NET provides the following validation controls :

- RequiredFieldValidator
- RangeValidator

- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

BaseValidator Class : The base validator class is the base class for all validation controls in ASP.NET. It provides several properties and methods that are common to all validation controls. Some of the important properties and methods of the base validator class are:

S.No.	Properties & Methods	Description
1.	ID	This property is used to set the ID of the Validation control.
2.	ControlToValidate	This property is used to specify the ID of the input control that should be validated by the Validation controls.
3.	Display	This property is used to specify when the error message should be displayed. The possible values are None (never), Static (always), and Dynamic (only when validation fails).
4.	Enabled	This property specifies whether the validation control is enabled or not. If set to false, the validation control will not perform any validation.
5.	ErrorMessage	This property is used to set the error message that should be displayed if validation fails.You can set this property to a string that contains the error message.
6.	Text	This property is used to set the text that should be displayed next to the input control if validation fails.It is typically used to provide a label for the input control.
7.	ValidationGroup	This property specifies the validation group to which the validation control belongs. You can use this property to group validation controls and validate them together as a single unit.
8.	SetFocusOnError	This property is used to specify whether focus should be set to the input control if validation fails. The default value is true.
9.	EvaluateIsValid()	This method evaluates the validity of the input control's value. It returns a boolean value indicating whether the input control's value is valid or not.
10.	IsValid()	This method returns a boolean value indicating whether the validation control passed validation. It first calls the EvaluateIsValid() method and then returns its value.
11.	Validate()	This method performs validation on the input control's value. If validation fails, it adds an error message to the page's validation summary. is method revalidates the control and updates the IsValid property.

(i) RequiredFieldValidator Control : The RequiredFieldValidator control is used to ensure that a user has entered a value in a required field. It is a server-side validation control that performs validation when the user submits the form. If the user does not enter a value in the required field, the control displays an error message indicating that the field is required. It is generally tied to a text box to force input into the text box.

Here is the syntax for using the RequiredFieldValidator control :

```
<asp:TextBox ID="txtName" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID=" rfvName" runat="server" ControlToValidate="
txtName" ErrorMessage=" Name is required">* </asp:Required FieldValidator>
```

(ii) RangeValidator Control : The RangeValidator control is used to ensure that a user has entered a value within a specified range. It is a server-side validation control that performs validation when the user submits the form. If the user enters a value outside of the specified range, the control displays an error message indicating that the value is outside of the valid range.

Here is the syntax for using the RangeValidator control :

```
<asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
<asp:RangeValidator ID="rvAge" runat="server"
ControlToValidate="txtAge"ErrorMessage="Age must be between 18 and 65"
Type="Integer" MinimumValue="18" MaximumValue="65">*</asp:RangeValidator>
```

It has the following specific properties :

S.No.	Properties	Description
1.	Type	This property is used to specify the data type of the input control. The possible values are String, Integer, Double, Currency, and Date.
2.	MinimumValue	This property is used to set the minimum value of the range. It should be set to a value of the same data type as the input control.
3.	MaximumValue	This property is used to set the maximum value of the range. It should be set to a value of the same data type as the input control.
4.	MinimumValueInclusive	This property is used to specify whether the minimum value should be included in the range. The default value is true.
5.	MaximumValueInclusive	This property is used to specify whether the maximum value should be included in the range. The default value is true.

(iii) CompareValidator Control : The CompareValidator control is a server control that is used to compare the value of one control with a fixed value or value of another control

on a web page. The control is used to perform validation on the user input and ensure that it meets certain criteria. This control is typically used for comparing input values such as passwords, dates, and numbers.

It checks whether the value of one control is equal to, greater than, or less than the value of another control.Here is the syntax for the CompareValidator control :

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
ControlToValidate=" TextBox1" ControlToCompare=" TextBox2" Operator="Equal"
ErrorMessage=" The values must match"/>
```

It has the following specific properties :

S.No.	Properties	Description
1.	Type	This property is used to specify the data type of the input controls being compared. The possible values are String, Integer, Double, Currency, and Date.
2.	ControlToCompare	This property is used to specify the ID of the input control to compare against.
3.	ValueToCompare	This property is used to specify the value to compare against. It should be set to a value of the same data type as the input controls being compared.
4.	OperatorThis property is	used to specify the comparison operator to use. The possible values are Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, and LessThanEqual.

(iv) **RegularExpressionValidator Control :** The RegularExpressionValidator control is a server control that is used to validate user input against a regular expression pattern. This control ensures that user input matches a specified pattern.

Here is the syntax for the RegularExpressionValidator control :

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
runat="server" ControlToValidate="TextBox1"
ValidationExpression="^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
ErrorMessage="Please enter a valid email address"/>
```

It has the following specific properties :

Properties	Description
ValidationExpression	This property is used to specify the regular expression pattern that the input control must match.

(v) **CustomValidator Control :** The CustomValidator control is used to perform custom validation on user input. It allows you to write your own validation code that can validate input based on specific business rules or requirements. It allows writing application specific custom validation routines for both the client side and the server side validation.

Here is the syntax of CustomValidator control :

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
ControlToValidate="TextBox1" OnServerValidate="CustomValidation" ErrorMessage="
Please enter a valid input"/>
```

It has the following specific properties :

Properties	Description
OnServerValidate	This property is used to specify the name of the server-side method that will perform the custom validation. This method should take two parameters : the source of the validation event and the arguments for the event.

To use the CustomValidator control, you need to define a server-side method that will perform the custom validation. This method should take two parameters: the source of the validation event and the arguments for the event. Here is an example of a server-side method that performs custom validation on a text box input :

```
protected void CustomValidation(object source, ServerValidateEventArgs args)
{
string input = args.Value;
if(input.Length < 5){
    args.IsValid = false;
  }
else{
    args.IsValid = true;
  }
}
```

allows to access the data enter by the user.

(vi) **Validation Summary Control :** The ValidationSummary control does not perform any validation, but it is used to display a summary of all validation errors that occur on a web page.The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

Here is the syntax for the ValidationSummary control :

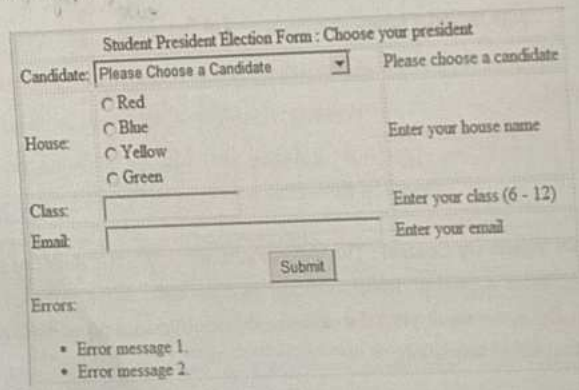
```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
HeaderText="Please correct the following errors:" ShowMessageBox=" true" Show
Summary="false" DisplayMode="BulletList"/>
```

It has the following specific properties :

S.No.	Properties	Description
1.	HeaderText	This property specifies the text that will be displayed at the top of the validation summary.
2.	ShowMessageBox	This property specifies whether a message box should be displayed when validation fails. Default is False.
3.	ShowSummary	This property specifies whether the summary should be displayed. Default is True.
4.	ValidationGroup	This property specifies the validation group for the ValidationSummary control.
4.	DisplayMode	This property specifies the display mode of the validation summary. The possible values are BulletList, List, SingleParagraph, and None.
6.	EnableClientScript	This property specifies whether client-side validation is enabled. Default is True.
7.	CssClass	This property specifies the CSS class for the ValidationSummary control.

Example : The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view :



The content file **ElectionForm.aspx** code is as given below

```
<%@Page Language="C#" AutoEvent Wireup="true" Unobtrusive Validation Mode="
None" CodeBehind=" Election Form.aspx.cs" Inherits=" Validation ControlsDemo.
Election Form"%>
<!DOCTYPE html>
<htm lxmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<table style="width: 66%;">
<tr>
<td class="style1" colspan="3" align="center">
<asp:Label ID="lblmsg"
Text="Student President Election Form : Choose your president"
runat="server"/>
</td>
</tr>
<tr>
<td class="style3">Candidate:</td>
<td class="style2">
<asp:DropDownList ID="ddlcandidate" runat=" server" style="width:239px">
<asp:ListItem>Please Choose a Candidate</asp:ListItem>
<asp:ListItem>Raju</asp:ListItem>
<asp:ListItem>Kiran</asp:ListItem>
<asp:ListItem>Suresh</asp:ListItem>
<asp:ListItem>Venkatesh</asp:ListItem>
</asp:DropDownList>
</td>
<td>
```

```
<asp:RequiredFieldValidator ID="rfvcandidate"
runat="server" ControlToValidate="ddlcandidate"
ErrorMessage="Please choose a candidate"
InitialValue="Please choose a candidate" ForeColor="Red">
</asp:RequiredFieldValidator>
</td>
</tr>
<tr>
<td class="style3">House:</td>
<td class="style2">
<asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
<asp:ListItem>Red</asp:ListItem>
<asp:ListItem>Blue</asp:ListItem>
<asp:ListItem>Yellow</asp:ListItem>
<asp:ListItem>Green</asp:ListItem>
</asp:RadioButtonList>
</td>
<td>
<asp:RequiredFieldValidator ID="rfvhouse" runat="server"
ControlToValidate="rblhouse" ErrorMessage="Enter your house name" ForeColor="
Red">
</asp:RequiredFieldValidator>
<br/>
</td>
</tr>
<tr>
<td class="style3">Class:</td>
<td class="style2">
<asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
</td>
<td>
```

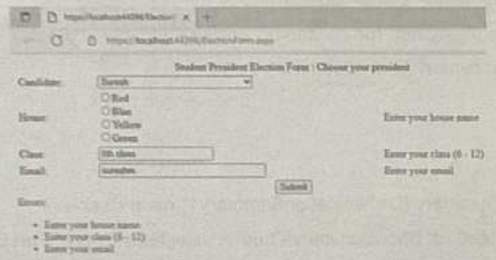


```
<asp:RangeValidator ID="rvclass"
runat="server" ControlToValidate="txtclass"
ErrorMessage="Enter your class (6 - 12)"MaximumValue="12"
MinimumValue="6" Type="Integer" ForeColor="Red">
</asp:RangeValidator>
</td>
</tr>
<tr>
<td class="style3">Email:</td>
<td class="style2">
<asp:TextBox ID="txtemail" runat="server" style="width:250px">
</asp:TextBox>
</td>
<td>
<asp:RegularExpressionValidator ID="remail" runat="server"
ControlToValidate="txtemail" ErrorMessage="Enter your email"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" ForeColor="Red">
</asp:RegularExpressionValidator>
</td>
</tr>
<tr>
<td class="style3" align="center" colspan="3">
<asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
style="text-align: center" Text="Submit"/>
</td>
</tr>
</table>
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
DisplayMode="BulletList" ShowSummary="true" HeaderText="Errors:" ForeColor="Red"/
>
</div>
</form>
</body>
</html>
```

The code behind file **ElectionForm.aspx.cs** code is as given below

```
using System;
using System.Web.UI;
namespace ValidationControlsDemo
{
public partial class ElectionForm : System.Web.UI.Page
   {
protected void Page_Load(object sender, EventArgs e){
      }
protected void btnsubmit_Click(object sender, EventArgs e){
if (Page.IsValid){
            lblmsg.Text = "Thank You";
         }
else{
            lblmsg.Text = "Fill up all the fields";
         }
      }
   }
}
```

Output :



5.14 IMPORTANCE OF DATA TRANSFER BETWEEN PAGES

In an ASP.NET Web Forms application, if you redirect from one web form page to another, you will frequently want to pass information from the source page to the target page. For example, you might have a page where users can select items to

purchase. When users submit the page, you want to call another page that can process the information that the user has entered.

Data transfer between pages in ASP.NET is an important aspect of building web applications. It allows for the exchange of information and data between different pages within an application, which can improve the user experience and enhance the functionality of the application.

Here are some of the reasons why data transfer between pages is so important :

- **Improved User Experience :** Data transfer between pages helps in providing a better user experience to the users by reducing the number of steps they need to complete a task. It allows users to enter information in one page and use that data on multiple pages without having to re-enter it.
- **Enhanced Security :** When dealing with sensitive information, data transfer between pages can help in maintaining the security of the data by allowing it to be stored on the server side and not being exposed to the client-side. It reduces the risk of cross-site scripting (XSS) attacks and other security vulnerabilities.
- **Improved Flexibility and Modularity :** Data transfer between pages enables developers to create web applications that are more flexible and modular. By breaking down complex processes into smaller, manageable parts, developers can create web applications that can be easily maintained, updated, and scaled.
- **Improved Performance :** Data transfer between pages can help to improve the performance of web applications by reducing the amount of data that needs to be transferred between the client and server. This can help to reduce network latency and improve page load times, resulting in a better user experience.

There are several ways to transfer data between pages. There are different ways of passing data across and within applications. Here are the most common methods:

Passing Data across Applications : Passing data across applications is possible through the following three techniques.

(i) **Using a Query String :** The query string is a string of text that is appended to the end of a URL. It can be used to pass data from one application to another by encoding the data in the query string. However, this method is not secure as the data can be easily seen in the URL.

(ii) **Using Cookies :** Cookies are small text files that can be stored on the client's computer by the web server. They can be used to store and pass data between web applications. In this method, data is stored in a cookie on the source page, which is then accessed by the target page. This method is more secure than using query strings but has a limit on the amount of data that can be passed.

(iii) **Getting HTTP POST Information from the Source Page :** In this method, the source page sends an HTTP POST request to the target page with the required data. The target page then retrieves the data from the request.The following options are available only when the source and target pages are in the same ASP.NET Web application.

Passing Data within Applications : Passing data within applications is possible through the following three techniques.

(i) **Using Session State :** Session state is a server-side storage mechanism that allows data to be stored and retrieved across multiple pages within an application. It can be used to pass data between pages within an application.

(ii) **Using Public Properties in the Source Page and Accessing the Property Values in the Target Page :** Public properties can be used to store data in the source page and access the property values in the target page. This method is suitable for passing small amounts of data between pages.

(iii) **Getting Control Information in the Target Page from Controls in the Source Page :** This method involves using controls on the source page to collect data from the user and then accessing the control values from the target page. This method is suitable for passing data between pages that are related to each other, such as a parent page and a child page.

5.15 USING QUERY STRING, COOKIES AND POST METHOD TO TRANSFER DATA BETWEEN PAGES WITH EXAMPLES

Passing data across application is possible through the following three techniques

1. Using a Query String
2. Using Cookies
3. Getting HTTP POST information from the source page.

1. **Using a Query String :** Data transfer between pages using a query string is one of the simplest ways to pass data. In this method, the data is encoded in the URL of the target

page as a query string parameter. The target page can then retrieve the data from the query string and use it as required.

This technique is one of the good ways when the data is not sensitive and is not too large. The default maximum length of a query string is 2048 chars. Here is the basic example of passing data through query string.

When you use a hyperlink or **Response.Redirect** to navigate from one page to another, you can add information in a query string at the end of the URL.

To use a query string to pass information :

1. In the source web forms page when you specify the URL of the target page, include the information that you want to pass in the form of key-value pairs at the end of the URL. The first pair is preceded by a question mark (?) and subsequent pairs are preceded by ampersands(&), as shown in the following example:

```
Response.Redirect(Target.aspx?field1=value1)

Response.Redirect (Target.aspx?field1=value1&field2=value2)
```

2. In the target page, access query string values by using the QueryString property of the HttpRequest object, as shown in the following example:

```
String s = Request.QueryString["field1"];
```

Code Example : Assume that we have a web page named **"Source.aspx"** that contains a textbox and a button. When the user enters a name in the textbox and clicks the button, the user is redirected to a second web page named **"Target.aspx"** that displays a message containing the user's name.

Here are the steps to achieve this using a query string :

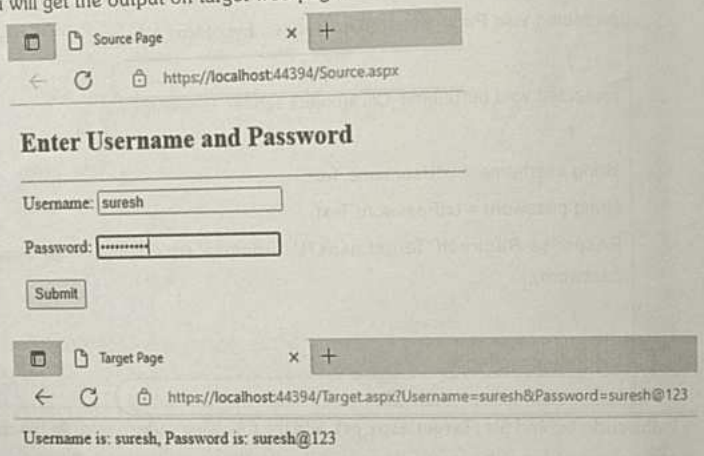
Step-1 : Add the following code to the **"Source.aspx"** page to redirect the user to the "Target.aspx" page with the name encoded in the query string.

```
protected void btnSubmit_Click(object sender, EventArgs e){
string name = txtName.Text;
Response.Redirect("Target.aspx?Name=" + name);
    }
```

Here, the "Response.Redirect" method is used to redirect the user to the "Target.aspx" page with the user's name encoded as a query string parameter.

```
public partial class Target : System.Web.UI.Page
  {
protected void Page_Load(object sender, EventArgs e)
    {
if (Request.QueryString["Username"] != null&& Request.QueryString["Password"] !=
null)
        {
string username = Request.QueryString["Username"].ToString();
string password = Request.QueryString["Password"].ToString();
Response.Write("Username is: "+username);
Response.Write(", Password is: "+password);
        }
      }
   }
}
```

Now run the source web page in a browser and fill the form and click on submit button. You will get the output on target web page.



2. **Using Cookies :** Cookies are small pieces of data that can be stored on the client-side and used to transfer data between pages. In ASP.NET, cookies can be used to transfer data between pages in the same web application or between different applications hosted on the same domain. This is the method used most often to store data.

Here is an example of how to use cookies to transfer data between pages in ASP.NET:

Step-1 : Set the cookie in the source page :

```
protected void btnSubmit_Click(object sender, EventArgs e){
   // Create a new cookie
   HttpCookie userCookie = new HttpCookie("UserInfo");
   // Set the cookie value
userCookie.Values["Username"] = txtUsername.Text;
userCookie.Values["Password"] = txtPassword.Text;
   // Set the cookie expiration date
   userCookie.Expires = DateTime.Now.AddDays(1);
   // Add the cookie to the response
Response.Cookies.Add(userCookie);
   // Redirect to the target page
Response.Redirect("TargetPage.aspx");
}
```

Step-2 : Retrieve the cookie value in the target page :

```
protected void Page_Load(object sender, EventArgs e){
   // Check if the cookie exists
if (Request.Cookies["UserInfo"] != null){
      // Get the cookie value
string username = Request.Cookies["UserInfo"]["Username"];
string password = Request.Cookies["UserInfo"]["Password"];
      // Use the cookie value
      lblUsername.Text = "Username: " + username;
      lblPassword.Text = "Password: " + password;
  }
}
```

Example Program : Here's an example program using cookies to transfer the username and password from the source page to the target page:

Step-2 : Add the following code to the **"Target.aspx"** page to retrieve the name from the query string and display a message containing the name.

```
protected void Page_Load(object sender, EventArgs e){
if (Request.QueryString["Name"] != null){
string name = Request.QueryString["Name"].ToString();
lblMessage.Text = "Hello " + name + ", welcome to the Target page!";
        }
    }
```

Data Transfer with Query String Example Program : Here's an example program that demonstrates how to transfer username and password from the source page to the target page using query string:

Source Page (Source.aspx) :

```
<%@Page Language= "C#" AutoEventWireup= "true" CodeBehind=" Source.aspx.cs"
Inherits="QueryStringDemo.Source"%>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Source Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<h2>Enter Username and Password</h2>
<hr/>
<label>Username:</label>
<asp:TextBox ID="txtUsername" runat="server"></asp:TextBox>
<br/><br/>
<label>Password:</label>
```



```
<asp:TextBox ID="txtPassword" runat="server"TextMode="Password">< asp:TextBox>
<br/><br/>
<asp:Button ID=" btnSubmit" runat=" server" Text="Submit" OnClick=" btnSubmit_
Click" style=" height: 26px"/>
</div>
</form>
</body>
</html>
```

In the code-behind file **(Source.aspx.cs),** add the following code to redirect the user to the target page with the username and password encoded as query string parameters:

```
using System;
namespace QueryStringDemo
{
public partial class Source : System.Web.UI.Page
  {
protected void Page_Load(object sender, EventArgs e){
    }
protected void btnSubmit_Click(object sender, EventArgs e)
    {
string username = txtUsername.Text;
string password = txtPassword.Text;
Response.Redirect("Target.aspx?Username="+username + "&Password="+
password);
    }
  }
}
```

In the code-behind file **(Target.aspx.cs),** add the following code to retrieve the username and password from the query string and display them:

```
using System;
namespace QueryStringDemo
{
```

WebForm1

[Design] Add Source page (Source.aspx):

```
<%@PageLanguage= "C#" AutoEventWireup=" true" CodeBehind=" Source.aspx.cs"
Inherits=" UsingCookiesDemo.Source"%>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Cookie Demo - Source Page</title>
</head>
<body>
<form id="form1"runat="server">
<div>
<label for="txtUsername">Username:</label>
<input type="text"id="txtUsername" runat="server"/><br/><br/>
<label for="txtPassword">Password:</label>
<input type="password" id="txtPassword"runat="server"/><br/><br/>
<asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick="btnSubmit_Click">
</div>
</div>
</form>
</body>
</html>
```

[Code Behind] Source.aspx.cs :

```
using System;
using System.Web;
namespace UsingCookiesDemo
{
public partial class Source : System.Web.UI.Page
   {
```



```
protected void Page_Load(object sender, EventArgs e){
    }
protected void btnSubmit_Click(object sender, EventArgs e){
// Create a new cookie
HttpCookie userCookie = new HttpCookie("UserInfo");
// Set the cookie values
userCookie.Values.Add("Username", txtUsername.Value);
userCookie.Values.Add("Password", txtPassword.Value);
// Set the cookie expiration date
userCookie.Expires = DateTime.Now.AddDays(1);
// Add the cookie to the response
Response.Cookies.Add(userCookie);
// Redirect to the target page
Response.Redirect("Target.aspx");
    }
  }
}
```

WebForm2

[Design] Add Target page (Target.aspx) :

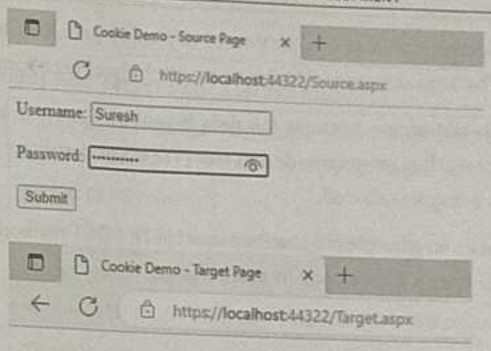
```
<%@Page Language="C#" AutoEventWireup="true" CodeBehind=" Target.aspx.cs"
Inherits="UsingCookiesDemo.Target"%>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Cookie Demo - Target Page</title>
</head>
<body>
```

```
<form id="form1" runat="server">
<div>
<h2>User Information:</h2>
<asp:Label ID="lblUsername" runat="server"/><br/>
<asp:Label ID="lblPassword" runat="server"/><br/>
</div>
</form>
</body>
</html>
```

[Code Behind] Target.aspx.cs:

```
using System;
namespace UsingCookiesDemo
{
public partial classTarget : System.Web.UI.Page
   {
protected void Page_Load(object sender, EventArgs e)
      {
// Check if the cookie exists
if (Request.Cookies["UserInfo"] != null){
// Get the cookie values
string username = Request.Cookies["UserInfo"]["Username"];
string password = Request.Cookies["UserInfo"]["Password"];
// Display the values in labels
            lblUsername.Text = "Username: " + username;
            lblPassword.Text = "Password: " + password;
         }
      }
   }
}
```

Now run the source web page in a browser and fill the form and click on submit button. You will get the output on target web page.



3. **Getting HTTP Post Information from the Source Page :** Data can be transferred between pages by getting HTTP POST information from the source page. This involves creating a form on the source page that posts the data to the target page. The target page can then retrieve the data from the form post.

We can also get the values using **Request.Form** but the thing that we have to keep in our mind is that the values from the source page should be posted using form element attribute **method="post".** In other words the source page should use HTTP POST action for redirection.

Here's an example : On the source page, create a form that posts the data to the target page :

```
<form method="post" action="TargetPage.aspx">
<input type="text" name="username" />
<input type="password" name="password" />
<input type="submit" value="Submit" />
</form>
```

On the target page, retrieve the data from the form post using the **Request.Form** collection :

```
string username = Request.Form["username"];
string password = Request.Form["password"];
```

The Request.Form collection contains a collection of key-value pairs that correspond to the form post data. The keys correspond to the name attribute of each form element.

Note that this method is not secure because the data is sent in plain text and can be intercepted by a third party. It is recommended to use HTTPS and encrypt the data if sensitive information is being transferred.

Example Program : Here's an example program that uses HTTP POST method to transfer data (username and age) from a source page to a target page :

WebForm-1 :

[Design] Add Source page (Source.aspx) :

```
<%@PageLanguage=" C#" Auto EventWireup=" true" CodeBehind=" Source.aspx.cs"
Inherits="UsingHTTPPostMethodDemo.Source"%>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>HTTP POST Method - Source Page</title>
</head>
<body>
<form id="form1" runat="server" method="post" action="Target.aspx">
<div>
<label for="username">Username:</label>
<input type="text" id="username" name="username"/>
<label for="age">Age:</label>
<input type="text" id="age"name="age"/>
<br/>
<input type="submit" value="Submit"/>
</div>
</form>
</body>
</html>
```


WebForm-2 :

[Design] Add Target page (Target.aspx) :

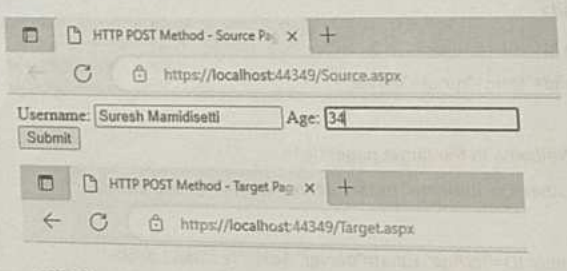
```
<%@Page Language="C#" AutoEventWireup="true" CodeBehind=" Target.aspx.cs"
Inherits="UsingHTTPPostMethodDemo.Target"%>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>HTTP POST Method - Target Page<</title>
</head>
<body>
<form id="form1"runat="server">
<div>
<h1>Welcome to the target page!</h1>
<asp:Label ID="lblName" runat="server" Text=""></asp:Label>
<br/>
<asp:Label ID="lblAge" runat="server" Text=""></asp:Label>
</div>
</form>
</body>
</html>
```

[Code Behind] Target.aspx.cs:b

```
using System;
namespace UsingHTTPPostMethodDemo
{
public partial classTarget : System.Web.UI.Page
  {
protected void Page_Load(object sender, EventArgs e)
    {
string username = Request.Form["username"];
string age = Request.Form["age"];
if (!string.IsNullOrEmpty(username) && !string.IsNullOrEmpty(age)){
```

```
            lblName.Text = "Name:" + username;
            lblAge.Text = "Age:" + age + " years";
        }
    }
}
```

Now run the source web page in a browser and fill the form and click on submit button. You will get the output on target web page.



REVIEW QUESTIONS

One Mark Questions :

1. What is label control?
2. What is button control?
3. What is check box control?
4. What is combo box control?
5. What is list box control?
6. What id Datagrid control?
7. What is MenuStrip control?

8. Define web application.
9. Define asp.net server controls.
10. Write the general syntax of asp.net server control.
11. Define validation controls.
12. Define Query String.
13. Define cookie.

Three Mark Questions :

14. Write the properties of combo box control.
15. Write the properties of button control.
16. Write the properties of list box control.
17. Write the properties of radio button control.
18. Write the properties of Datagrid control.
19. How do you enable and disable the controls in windows applications.
20. Write about create short cut keys for pull down menus.
21. Write the differences between web application and Desktop Application/Network Enabled application.
22. Write about asp.net web controls.
23. List the categories of asp.net server controls.
24. List various data validation controls.
25. Write about passing data across web application.
26. Write about passing data within web application.
27. Write about Getting HTTP Post Information from the Source Page.

Five Mark Questions :

28. Write a C# program to perform arithmetic operations like addition, subtraction, multiplication, division using label, textbox and button controls.
29. Write a C# program using checkbox control
30. Write a C# program using radio button control.

BOARD DIPLOMA EXAMINATION

MID-I

.NET PROGRAMMING THROUGH C#

V Semester

Time : 1 : 00 Hour Max. Marks : 20

PART – A $4 \times 1 = 4$

Note : *Answer **all** questions and each question carries **One** marks.*

1. Define CLR.
2. Define MSIL.
3. Define Methodoverriding.
4. Write the syntax to create a structure.

PART – B $2 \times 3 = 6$

Note : *Answer any **Two** questions and each question carries **Three** marks.*

5. (a) Demonstrate the architecture of CLR.

   (or)

   (b) List the features of .net framework.

6. (a) Write the differences betweeen structures and class.

   (or)

   (b) Discuss different access modifiers.

PART – C $2 \times 5 = 10$

Note : *Answer **all** questions and each question carries **Five** marks.*

7. (a) Explain the features of visual studio.

   (or)

   (b) Explain different windows in visual studio.

8. (a) Write a C# program to implment multiple inheritance.

   (or)

   (b) Write a C# program to access the members of a structure.

BOARD DIPLOMA EXAMINATION

MID-II

.NET PROGRAMMING THROUGH C#

V Semester

Time : 1 : 00 Hour Max. Marks : 20

PART – A

4 × 1 = 4

Note : *Answer **all** questions and each question carries **One** marks.*

1. Write the syntax of multiple catch blocks with a single try block.
2. What is the class used to define user defined exception.
3. Write the syntax for lambda expression.
4. What is the use of 'is' operator.

PART – B

2 × 3 = 6

Note : *Answer any **Two** questions and each question carries **Three** marks.*

5. (a) Discuss the keyword related to exception handling.

   (or)

   (b) Write about thread life cycle.

6. (a) Write about indexers and properties in a class.

   (or)

   (b) Write the need of generic programming.

PART – C

2 × 5 = 10

Note : *Answer **all** questions and each question carries **Five** marks.*

7. (a) Write a C# program to create multiple threads.

   (or)

   (b) Write a C# program to illustrate that program is not terminated when exception occurs.

8. (a) Write a C# program with anonymous method that accepts arguments and return parameters.

   (or)

   (b) Write a C# program to define a class with generic data members.

WARNING.....XEROX/PHOTOCOPYING OF THIS BOOK IS ILLEGAL

BOARD DIPLOMA EXAMINATION

SEMISETER END EXAMINATION (MODEL PAPER)

.NET PROGRAMMING THROUGH C#

V Semester V Semester

Time : 2 : 00 Hours **Max. Marks : 40**

PART – A 8 × 1 = 4

Note : *Answer **all** questions and each question carries **One** marks.*

1. Write the syntax to define a constructor in a class.
2. Write the syntax to create lambda expression.
3. Define Cookie.
4. Define instance variable.
5. What is Windows Form?
6. What is the use of run at attribute in a server control?
7. Define data grid control.
8. List any two LINQ operators.

PART – B 4 × 3 = 12

Note : *Answer **all** questions and each question carries **Three** marks.*

9. (a) Draw and Explain CLR architecture.

   (or)

   (b) Write any five properties of textbox and list box controls.

10. (a) Discuss about the five methods in a thread class.

    (or)

    (b) List the features and advantages of ADO.NET.

11. (a) Discuss the steps for creating a web application.

    (or)

    (b) List various Data validation controls.

12. (a) Write about different types of LINQ objects.

    (or)

    (b) Discuss connection, Dataset and Data adaptor object.

PART – C

$4 \times 5 = 20$

Note : *Answer* ***all*** *questions and each question carries* ***Five*** *marks.*

13. (a) Write about the following windows (a) Object browser window (b) solution explorer window (c) server explorer window.

(or)

(b) Develop a C# application to sort the items in the list box.

14. (a) Write a C# program to pass parameters and return value from anonymous methods.

(or)

(b) Write a C# program to insert rows in a table.

15. (a) Write a C# program to create menus at runtime.

(or)

(b) Write the asp.net code to create a student registration form.

16. (a) Write a C# code to navigate all the records in the table.

(or)

(b) Write a C# program to access the students records using LINQ objects.

6.1 FAMILIARIZE WITH ADO.NET

ADO.NET is a data access technology provided by Microsoft that is used to access and manipulates data from different data sources, primarily databases, in .NET applications. It stands for **ActiveX Data Objects** for .NET.It is widely used for accessing, selecting, saving, deleting or managing data with the database.

ADO.NET is a data bridge between your applications and databases that carry data between them. ADO.NET has a rich set of classes, methods, and interfaces that allow you to handle data in the database more efficiently and also enable developers to build data-oriented applications. It offers a consistent programming model and a set of features for working with data, regardless of the underlying data source.

The following are few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.

- Console Applications
- Windows Applications
- ASP.NET Web Applications

Various Connection Architectures

In ADO.NET, there are two main types of connection architectures :

1. **Connected Architecture :** In the connected architecture, a connection is established with the database throughout the duration of the interaction with the data source. The connection remains open until the data operation is completed, and it is explicitly closed when no longer needed. In this architecture, the data is retrieved and manipulated directly while the connection is open.
2. **Disconnected Architecture :** In the disconnected architecture, the connection with the data source is only established when necessary to retrieve or update data. Once the data is fetched, the connection is closed, and the data is stored in a local data structure such as a **DataSet** or **DataTable**. The data can then be manipulated locally without maintaining an open connection to the data source. The connection is reopened when it is time to synchronize the changes made in the local data structure back to the data source.

ADO.NET has a set of classes that allows you to connect and work with data sources like databases, excel file, access file, xml file, mysql, sql or notepad. To connect your application with different sources of database you need to know the right Data Provider. There are several Data Providers in **ADO.NET** that connect with different types of sources. A list of ADO.NET Data Providers is listed below :

Here are s

DataProvider
SQL Server
Oracle
OLEDB
ODBC
MySQL
SQLite

Core con
used to c

1. Connect
provides
specifyin
database
is DbCo
2. Comma
data sou
procedu
Connec
base cla
3. DataRe
from a
data fro
retrieve

Here are some of the popular ADO.NET Data Providers :

taProvider	Description
SQL Server	**System.Data.SqlClient :** This data provider is used to connect to Microsoft SQL Server databases. It provides a rich set of features for working with SQL Server, including support for stored procedures, transactions, and user-defined types.
Oracle	**System.Data.OracleClient :** This data provider is used to connect to Oracle databases. It provides support for Oracle-specific data types, such as BLOBs and CLOBs, and enables developers to use Oracle-specific features like PL/SQL.
OLEDB	**System.Data.OleDb :** This data provider is a low-level interface to OLE DB, which is a Microsoft technology for accessing different types of data sources. It provides a way to connect to a wide range of data sources, including databases, spreadsheets, and text files.
ODBC	**System.Data.Odbc :** This data provider is a low-level interface to ODBC, which is a standard interface for accessing different types of data sources. It provides a way to connect to a wide range of data sources, including databases (.mdb),spreadsheets, and text files.
MySQL	This data provider is used to connect to MySQL databases. It provides a rich set of features for working with MySQL, including support for stored procedures and transactions.
SQLite	This data provider is used to connect to SQLite databases. It provides a way to access and manipulate SQLite databases, which are popular for embedded systems and mobile devices.

Core components of ADO.NET : There are 4 Core components of ADO.NET that are used to connect access and retrieve data from the database.

1. **Connection :** The Connection object represents a connection to a data source. It provides methods for opening and closing connections, as well as properties for specifying connection strings, which contain information such as the server name, database name, credentials, and other connection-specific settings. The base class is **DbConnection**.

2. **Command :** The Command object is used to execute queries or commands against a data source. It supports different types of commands, including SQL queries, stored procedure invocations, and more. The Command object can be associated with a Connection object to execute commands against the corresponding data source. The base class is **DbCommand**.

3. **DataReader :** The DataReader object provides a forward-only, read-only stream of data from a data source. It is optimized for retrieving large result sets efficiently, as it fetches data from the data source on a row-by-row basis. DataReader is ideal when you need to retrieve and process large amounts of data quickly.The base class is **DbDataReader**.

4. **DataAdapter :** The DataAdapter object acts as a bridge between a DataSet and a data source. It provides methods to fill a DataSet with data from a data source and update changes made to the DataSet back to the data source. The DataAdapter uses Command objects to retrieve and update data. The base class is **DbDataAdapter**.

Some other key components and concepts in ADO.NET include :

Data Providers : ADO.NET includes data providers for different types of data sources, such as SQL Server, Oracle, MySQL, and more. Each data provider implements the common interfaces and classes defined by ADO.NET, allowing developers to work with data in a consistent manner.

DataSet : The DataSet object represents an in-memory cache of data retrieved from a data source. It can hold multiple tables, relationships, and constraints. The DataSet provides a disconnected data model, meaning it can be populated with data from a data source and then disconnected from the source. This allows you to work with the data offline and make changes without affecting the original data source.

Transactions : ADO.NET supports transactions to ensure data integrity and consistency. You can use the Transaction object to group multiple database operations into a single unit of work, allowing you to commit or roll back the changes as a whole.

6.2 FEATURES AND ADVANTAGES OF ADO.NET

ADO.NET offers several features that make it a powerful and versatile data access technology for .NET applications. Here are some key features of ADO.NET :

1. **Disconnected Data Model :** ADO.NET provides a disconnected data model, which allows data to be retrieved from a data source and stored in memory without maintaining a continuous connection to the source. This enables offline data manipulation, improved performance, and reduced network traffic. This is done using the **DataSet** and **DataAdapter** classes.
2. **Data Providers :** ADO.NET supports various data providers that facilitate connectivity to different data sources such as SQL Server, Oracle, MySQL, and more. Each data provider implements the common ADO.NET interfaces, ensuring a consistent programming model regardless of the underlying database.
3. **Data Provider Independence :** ADO.NET provides a set of interfaces and classes that abstract the underlying data source and data provider. This means that developers can use the same ADO.NET code to access and manipulate data from different data sources without needing to know the specifics of each data source.

Connection Management : ADO.NET includes classes and methods for establishing and managing database connections. Developers can open connections to data sources, close them when they are no longer needed, and efficiently manage connection pooling to improve performance.

Connection Pooling : ADO.NET provides connection pooling, which enables developers to reuse existing connections rather than creating new connections for each data access operation. This improves application performance and reduces the load on the data source.

Command Execution : ADO.NET offers a Command object that allows the execution of commands against a data source. It supports various types of commands, including SQL queries, stored procedure invocations, and parameterized queries. This enables developers to interact with the database and retrieve or modify data.

7. **Data Retrieval :** ADO.NET provides a DataReader object for retrieving data from a data source in a fast, forward-only, read-only manner. The DataReader is optimized for efficiently processing large result sets, making it ideal for scenarios where fast data retrieval is required.

8. **DataSet and DataAdapter :** ADO.NET introduces the DataSet and DataAdapter classes, which together provide a powerful mechanism for working with data. The DataSet represents an in-memory cache of data retrieved from a data source and can hold multiple tables, relationships, and constraints. The DataAdapter acts as a bridge between the DataSet and the data source, facilitating data retrieval, updates, and synchronization.

9. **Data Binding :** ADO.NET provides support for data binding, which enables developers to bind data from a data source directly to User Interface (UI) controls in their applications. This simplifies the development process by reducing the amount of code required to display and manipulate data.

Transaction Support : ADO.NET supports transactions, enabling developers to group multiple database operations into atomic units of work (transaction). Transactions ensure data integrity and consistency by allowing either all changes to be committed or rolled back in case of failures.

11. **XML Integration :** ADO.NET provides robust support for working with XML data. It allows data to be retrieved from a database and represented as XML, or XML data to be stored back into the database. This integration simplifies tasks involving XML data processing and manipulation.

12. **Asynchronous Operations :** ADO.NET offers asynchronous execution capabilities, allowing long-running database operations to be performed in a non-blocking manner. Asynchronous methods enable developers to improve the responsiveness and scalability of their applications.
13. **LINQ Support :** ADO.NET provides support for Language Integrated Query (LINQ), which enables developers to write queries against in-memory data structures, including DataSets, using a syntax similar to SQL.

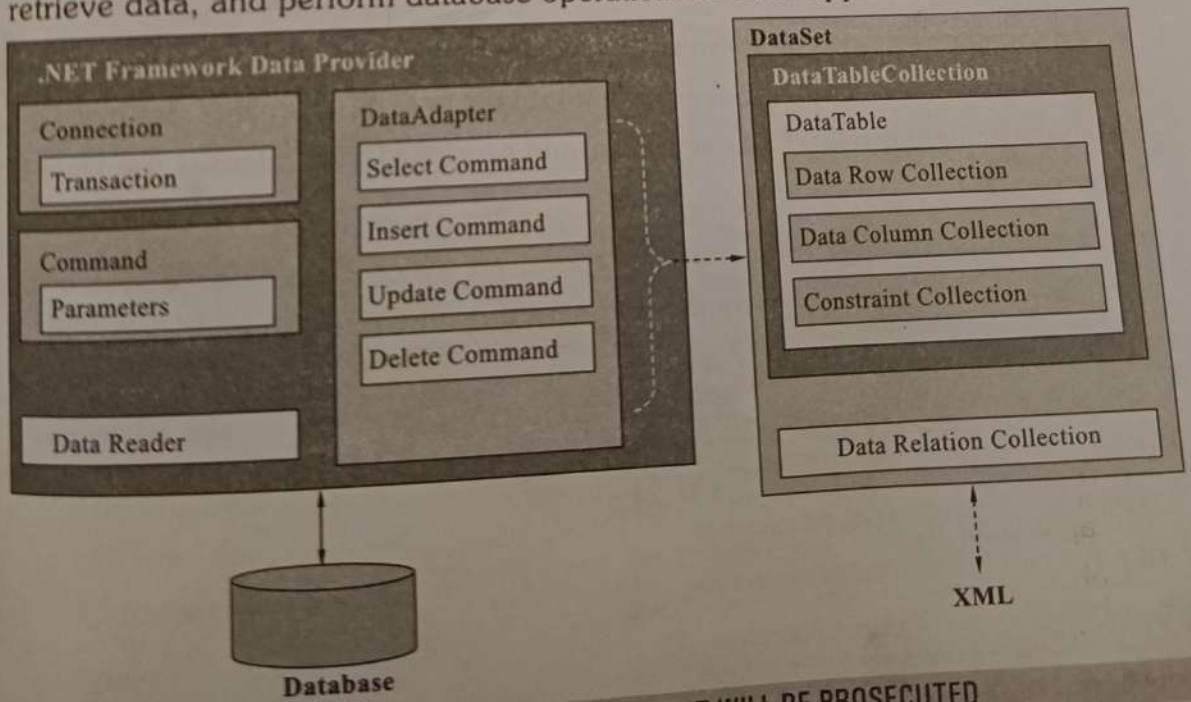
Advantages of ADO.NET

1. **Scalability :** ADO.NET is designed to be scalable, allowing applications to handle large amounts of data and multiple users. With ADO.NET, developers can manage database connections and transactions efficiently, and can retrieve and manipulate data quickly and easily.
2. **Data Source Independence :** ADO.NET provides a consistent programming model regardless of the underlying data source. Developers can use the same ADO.NET code to connect and interact with different databases like SQL Server, Oracle, MySQL, etc. This data source independence simplifies application development and maintenance.
3. **Interoperability :** As ADO.NET transmits the data using the format of XML which is not dependent on ADO.NET or windows platform.
4. **Performance :** ADO.NET is optimized for performance, making it faster and more efficient than previous data access technologies like ADO that uses **COM marshaling**. ADO.NET supports features like connection pooling, batched updates, and asynchronous data access, which help to improve application performance.
5. **Disconnected Data Model :** ADO.NET supports a disconnected data model through the use of DataSets. DataSets allow developers to retrieve data from a data source, disconnect from it, and work with the data locally.
6. **Data Binding :** ADO.NET supports data binding, which allows developers to bind data directly to user interface (UI) controls in their applications. This makes it easy to display and manipulate data in UI controls without writing a lot of code.
7. **Transaction Management :** ADO.NET offers built-in transaction management capabilities. Developers can use transactions to ensure data integrity and consistency when performing multiple database operations as a single unit of work. Transactions can be easily managed using the Transaction object provided by ADO.NET.

8. **Security :** ADO.NET includes security features that enable developers to secure their applications and protect data. For example, ADO.NET supports authentication and authorization mechanisms, and allows developers to encrypt data.
9. **Easy Development :** ADO.NET provides a rich set of classes, components, and tools that make data access development straightforward.
10. **Flexibility :** ADO.NET supports a wide range of data sources and data providers, including SQL Server, Oracle, MySQL, and SQLite. This makes it a flexible technology that can be used in a variety of scenarios.
11. **Extensibility :** ADO.NET is highly extensible, allowing developers to create custom data providers for specific data sources or to extend the functionality of existing providers.
12. **Integration with Other .NET Technologies :** ADO.NET is integrated with other .NET technologies, such as LINQ and Entity Framework. This makes it easy to work with data in .NET applications using a consistent set of APIs and programming models.

6.3 ESTABLISH CONNECTION TO DATABASE USING CONNECTION, DATASET, DATAADAPTOR AND COMMAND OBJECTS

ADO.NET Architecture : ADO.NET architecture is a data access technology provided by Microsoft as part of the .NET framework. It consists of various components and layers that work together to enable developers to interact with different data sources, retrieve data, and perform database operations in .NET applications.



The key components of ADO.NET architecture are :

1. **Data Providers :** Data Providers are responsible for establishing a connection to the data source and executing commands against it. Each data provider corresponds to a specific data source, such as SQL Server, Oracle, MySQL, etc. Data Providers implement the ADO.NET interfaces and classes that define common data access functionality such as connecting to a data source, executing commands at a data source and fetching data from a data source.

Various Data Providers :

.NET Framework data provider	Description
SQL Server	Provides data access for Microsoft SQL Server. Uses the **System.Data.SqlClient** namespace.
Oracle	Provides data access for Oracle data sources. Uses the **System.Data.OracleClient** namespace.
OLE DB	For data sources exposed by using OLE DB. Uses the **System.Data.OleDb** namespace.
ODBC	For data sources exposed by using ODBC. Uses the **System.Data.Odbc** namespace.

2. **Connection Object :** The Connection object represents a connection to the data source. It connects to the specified Data Source and open a connection between the C# application and the Data Source. It provides methods for opening and closing connections, managing transactions, and executing commands against the database. The Connection object establishes and maintains a physical connection to the data source.

When the connection is established, SQL Commands will execute with the help of the Connection Object and retrieve or manipulate data in the Data Source.Once the Database activity is over, Connection should be closed and releases the Data Source resource.

In C# the type of the Connection is depend on which Data Source system you are working with. The following are the commonly used Connection classes.

Data Provider	Connection Class
SQL Server	SqlConnection
Oracle	OracleConnection
MySQL	MySqlConnection
OLE DB	OleDbConnection
ODBC	OdbcConnection

Command Object : The Command object encapsulates a query or command to be executed against the data source. It can be associated with a Connection object and contains properties for setting the command text, command type (e.g., SQL query, stored procedure), and parameters. The Command object provides methods for executing the command and retrieving the results.

The Command Object in ADO.NET executes SQL statements and Stored Procedures against the data source specified in the Connection Object. The Command Object requires an instance of Connection Object for executing the SQL statements.

The following are the various commands that are executed by the Command Class.

- **ExecuteReader :** Returns data to the client as rows. This would typically be an SQL SELECT statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports.
- **ExecuteNonQuery :** Executes a command that changes the data in the database, such as an UPDATE, DELETE, or INSERT statement, or a Stored Procedure that contains one or more of these statements. This method returns an integer that is the number of rows affected by the query.
- **ExecuteScalar :** This method only returns a single value used to execute aggregate functions(SUM,AVG,COUNT,MAX,MIN etc.). This kind of query returns a count of rows or a calculated value.

DataReader Object : The DataReader object provides a forward-only, read-only stream of data retrieved from the data source. It means you can only read and display data but cant update or delete data. If you want to make modification in retrieved data you need to use DataAdapter instead of DataReader.It is optimized for retrieving large result sets efficiently.

The DataReader allows sequential access to the data and is suitable when you need to retrieve and process large amounts of data quickly.When you want to display information or search result, you can use DataReader. There are various advantages of using DataReader like :

- The retrieved data is stored in the network buffer in the client and then the client can read data using Read method. As data gets stored in the client network buffer it increases application performance significantly.

- By default DataReader stores only one row at a time in memory. It reduces system overhead.

Some Methods in DataReader class

Method	Description
Close()	The Close method closes the DataReader object and releases any associated resources.
Read()	The Read method advances the DataReader to the next record and returns true if there are more rows, or false if there are no more rows to read.
NextResult()	Advances the data reader to the next result during batch transactions.
Getxxx()	There are dozens of Getxxx methods. These methods read a specific data type value from a column. For example. GetChar will return a column value as a character and GetString as a string. Other methods include GetFloat, GetDouble, GetDecimal, GetBoolean, and more.

5. **DataAdapter Object :** The DataAdapter acts as a bridge between the data source and the DataSet. It fills the DataSet with data from the data source and updates changes made to the DataSet back to the data source. The DataAdapter uses Command objects to retrieve data and execute updates, inserts, and deletes.

   The DataAdapter provides four important commands/properties that allow us to control how updates are made to the database server :

Properties	Description
DeleteCommand	It is used for Deleting Records from DataSource.
InsertCommand	It is used for adding New Record to a DataSource.
SelectCommand	It is used for Selecting Records from a DataSource.
UpdateCommand	It is used for Updating Records in a DataSource.

6. **DataSet Object :** The DataSet is an in-memory representation of data that provides a disconnected data model. It can hold multiple tables, relationships, and constraints. The DataSet is populated with data retrieved from the data source using the DataAdapter and can be manipulated offline. The DataSet contains DataTable objects and DataRelation objects. The DataRelation objects represent the relationship between two tables.

   The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of DataTable objects. In turn each object contains a collection of DataColumn and DataRow objects. The DataSet also contains a Relations collection that can be used to define relations among Data Table Objects.

7. **DataTable Object :** A DataTable object contains the rows and columns of data held in memory inside a DataSet object. It is similar to a database table in a relational database,

except that all the data is available at once and there is no concept of a current row. A *DataTable* object has a collection of *DataColumn*o bjects which define the format of each column in the DataTable. The DataTable also has a collection of *DataRow* objects through which the data in the table can be accessed and updated.

(a) **DataRow Object :** A *DataRow* object contains a single record from the *DataTable* to which it belongs. All operations on the data on a record are performed using the *DataRow* object which contains the data.

(b) **DataColumn Object :** A *DataColumn* object is used to define the format of a column in a *DataTable* in terms of its data type, length, etc.

(c) **DataRelation Object :** Relational databases allow relationships to be defined between multiple tables. These relationships, which are based on the use of foreign keys, are used to ensure the consistency and integrity of data entered in to the database tables. A DataRelation object can be used to define and enforce the relationship between DataColumns in different DataTable objects. Proper use of DataRelations helps ensure referential integrity and can also provide automatic removal of all related data from a DataSet when one item of data is deleted.

To establish a connection to a database using Connection, DataSet, DataAdapter, and Command objects in ADO.NET, you can follow these steps :

1. **Create a Connection Object :**

```
string connectionString = "Data Source=myServer;Initial Catalog=myDatabase;User
ID=myUsername;Password=myPassword;";
SqlConnection connection = new SqlConnection(connectionString);
```

Replace 'myServer', 'myDatabase', 'myUsername', and 'myPassword' with the appropriate values for your database server.

2. **Open the Connection :**

```
connection.Open();
```

3. **Create a Command Object :**

```
string query = "SELECT * FROM Customers";
SqlCommand command = new SqlCommand(query, connection);
```

Replace 'Customers' with the name of the table or query you want to execute.

4. **Create a DataAdapter object and associate it with the Command object :**

```
SqlDataAdapter adapter = new SqlDataAdapter(command);
```

5. Create a DataSet object :

```
DataSet dataset = new DataSet();
```

6. Use the DataAdapter to fill the DataSet with data from the database:

```
adapter.Fill(dataset, " Customers");
```

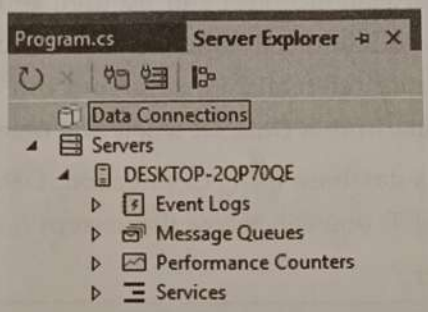
7. Close the connection:

```
connection.Close();
```

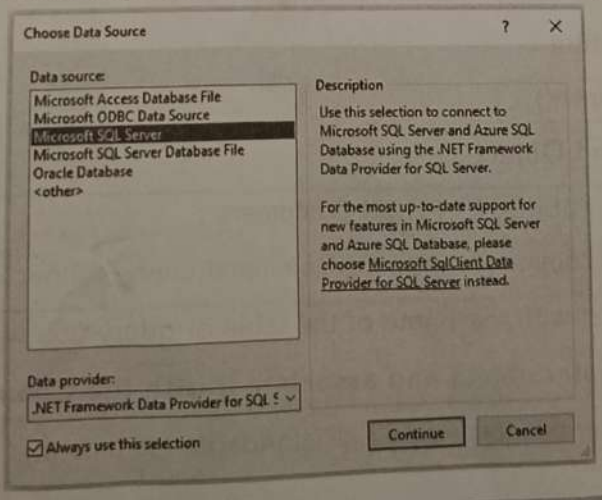
6.4 CONNECT TO DATABASE THROUGH SERVER EXPLORER

To connect to a database using a C# application through the Server Explorerin Visual Studio, you can follow these steps :

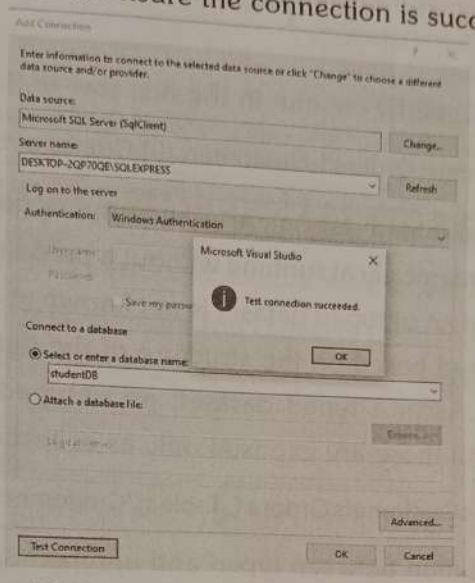
1. Open Visual Studio and create a new C# project or open an existing project.
2. Go to the "View" menu and select "Server Explorer" to open the Server Explorer windowor press **Ctrl+Alt+S**. The below figure shows the server explorer.



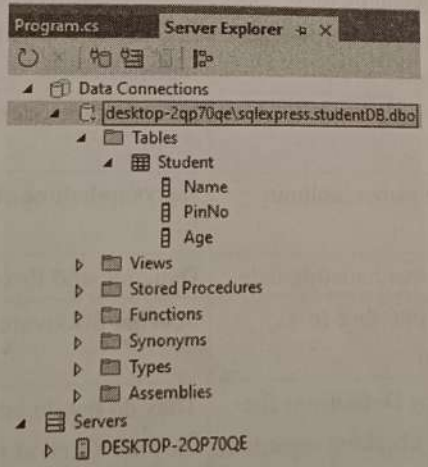
3. In the Server Explorer window, right-click on "Data Connections" and select "Add Connection".
4. The "Add Connection" dialog box will appear. Choose the appropriate data source based on your database provider (e.g., Microsoft SQL Server, MySQL, Oracle, etc.).



5. In the Add Connection dialog box, enter the necessary information to connect to your database, such as the server name, database name, and authentication credentials. Click "Test Connection" to ensure the connection is successful.



6. Click on "OK" to establish the connection. The connection will now appear under "Data Connections" in the Server Explorer window.



7. Now, in your C# application, you can use the connected database by accessing the Server Explorer's connection.

6.5 DIFFERENTIATE TYPED AND UNTYPED DATASET OBJECTS

In the context of ADO.NET, **typed** and **untyped datasets** refer to two different ways of representing and manipulating dataretrieved from a database in a disconnected manner.

Typed Dataset : A typed dataset is a dataset object that is created at design time or compile time and has a predefined schema. It is generated using a tool like Visual

Studio's Dataset Designer or **XSD.exe** (XML Schema Definition tool). The schema of a typed dataset is based on an XML schema (XSD) or a database schema, and the dataset's tables, columns, and relationships are represented as strongly-typed classes and properties.

//This accesses the CustomerID column in the first row of the Customers table.

```
string s = dsCustomersOrders1.Customers[0].CustomerID;
```

Untyped Dataset : An untyped dataset, also known as a general dataset, is a dataset object that is created dynamically at runtime without a predefined schema. Unlike typed datasets, the structure of an untyped dataset is not known at compile time. Instead, the dataset schema is inferred based on the structure of the data retrieved from the data source during runtime.As in a typed dataset, an untyped dataset contains tables, columns, and so on - but those are exposed only as collections.

```
string s = (string) dsCustomersOrders1.Tables["Customers"].Rows[0]["CustomerID"];
```

Here is a brief differentiation between typed and untyped datasets :

S.No.	Typed Dataset Objects	Untyped Dataset Objects
1.	Predefined schema generated at compile time.	Dynamically inferred schema at runtime.
2.	Strongly-typed with compile-time checking of table and column names, and data types.	Loosely-typed with no compile-time checking for schema or data.
3.	Code is generated for dataset schema and strongly-typed classes.	No code generation; schema is discovered at runtime.
4.	Compile-time checking for table names, column names, and data types.	No compile-time checking for structure or data types.
5.	Type safety when accessing and manipulating data.	Dynamic and flexible data structure.
6.	Design-time support with visual interface (e.g., Visual Studio's Dataset Designer)	Schema discovered and updated at runtime.
7.	They have .xsd file (XML Schema Definition) file associated with them and do error checking regarding their schema at design time using the .xsd definitions.	They do not do error checking at the design time as they are filled at run time when the code executes.
8.	Improved code readability and intellisense support.	More adaptability to changing data structures.We cannot get intellisense support.
9.	Suitable for scenarios with fixed or predictable data schema.	Suitable for scenarios with dynamic or changing data schema.
10.	Limited flexibility due to predefined schema.	More flexible due to dynamic schema and data structure.
11.	Slightly more efficient due to type safety and compile-time checking.	Slightly less efficient due to runtime schema discovery.

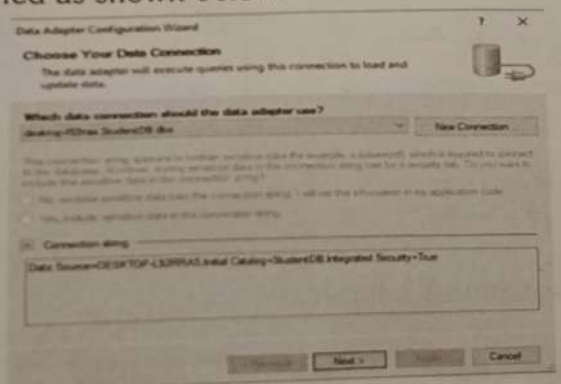
12.	Typed Datasets use explicit names and data types for their members.
	Example :
	// This accesses the for their members.
	Name column in the first row of the Employee
	table.string s = dsEmployee.Employee[0].Name;
		Untyped Datasets use table and column collections for their members.
		Example :
		string s = (string)
		dsEmployee.Tables ["Employee"]. Rows[0]
		["Name"];

6.6 ACCESS DATA WITH DATAADAPTERS AND TYPED/UNTYPED DATASETS

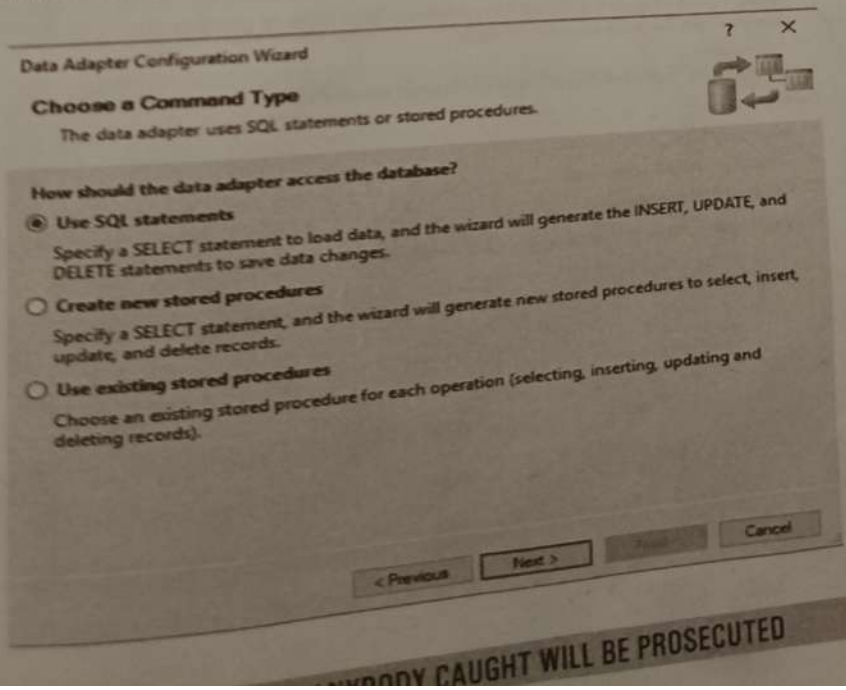
Accessing data with DataAdapters and Datasets (both typed and untyped) is a common approach in ADO.NET for retrieving, manipulating, and updating data from a database.

Method-1 : To access data using DataAdapters in **SQL Server at Design time**, you can follow these steps :

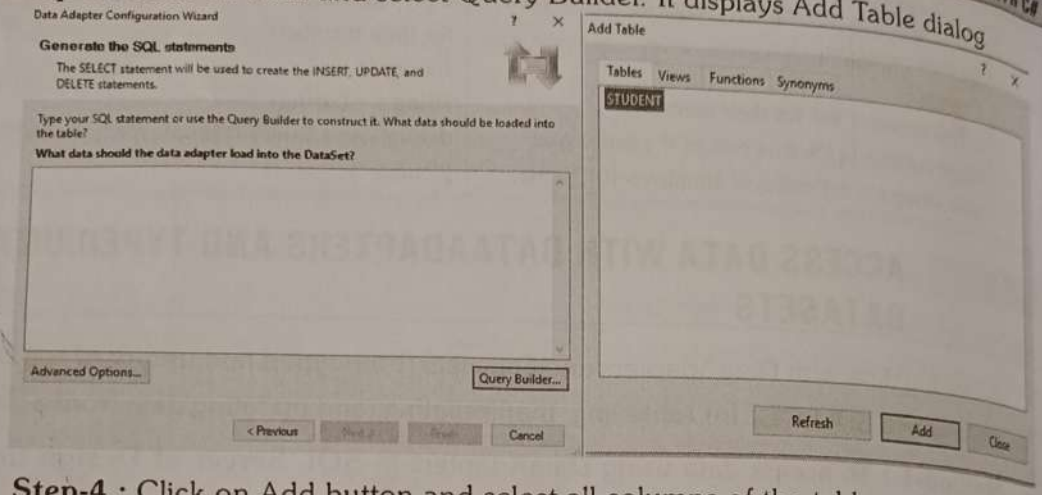
Step-1 : When we add SqlDataAdapter control Data Adapter Configuration Wizard dialog will be opened as shown below.



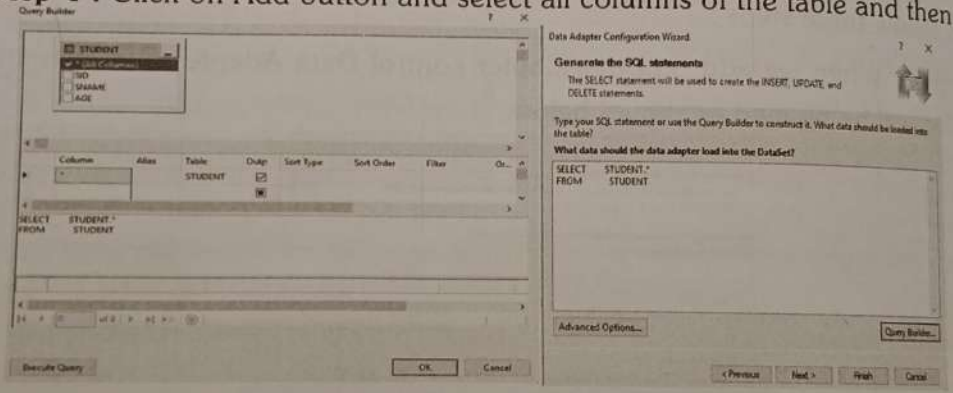
Step-2 : Click on Next and select use SQL statements as shown below.



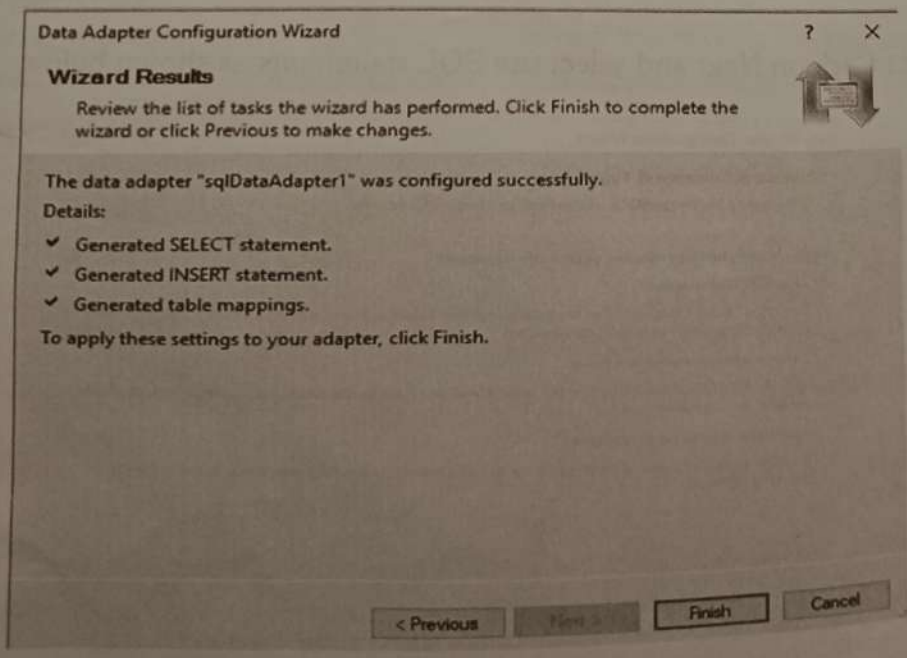
Step-3 : Click on Next and select Query Builder. It displays Add Table dialog



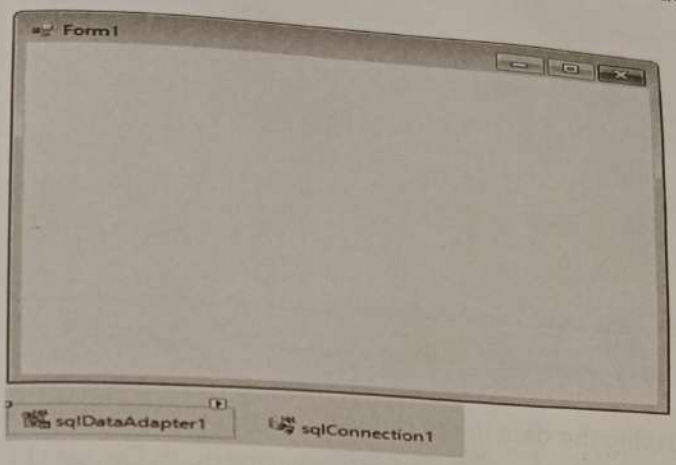
Step-4 : Click on Add button and select all columns of the table and then press ok.



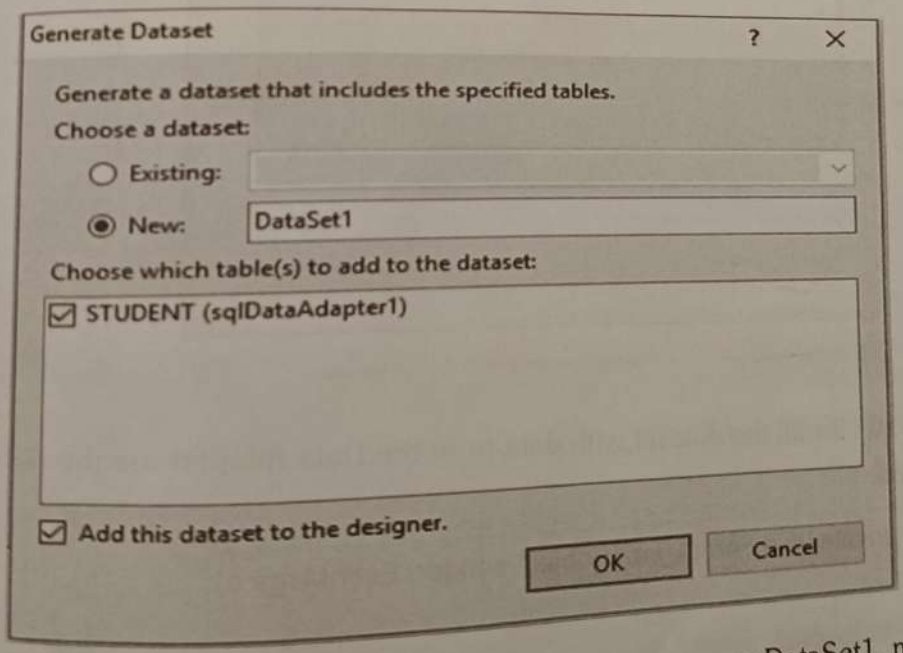
Step-5 : Click on Next and click finish.



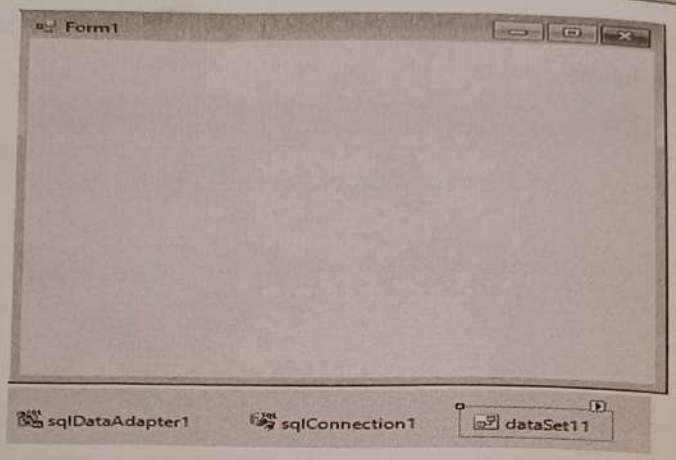
Step-6 : It will create SqlConnection1 object automatically and added to component tray as shown below.



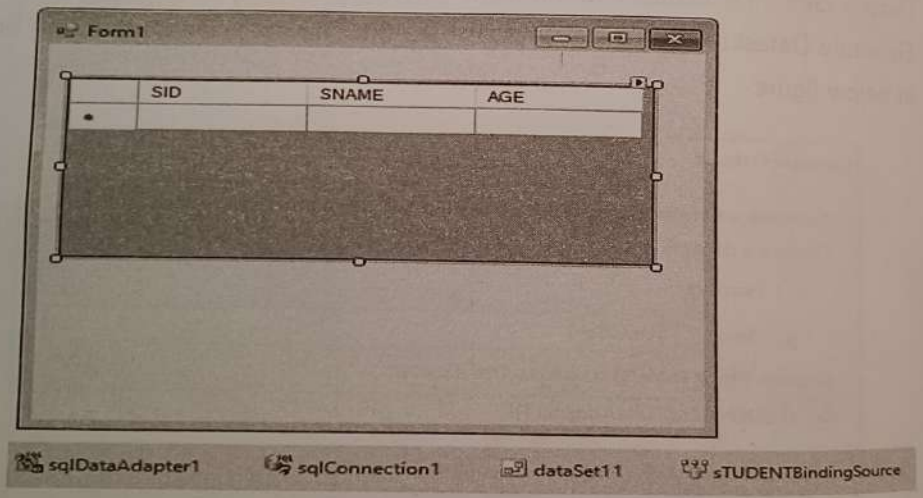
Step-7 : To generate the dataset that holds the data from the Data Adapter, select the Data-> Generate Dataset menu item (or) right click on sqlDataAdapter1 and select Generate Dataset menu item. This display the Generate Dataset dialog box as shown in below figure.



Step-8 : Click on New option to create a new Dataset name DataSet1, make sure that STUDENT table checkbox is checked as well as Add this dataset to the designer checked, then click Ok. By doing this a new dataset DataSet11 is added to the form designers component tray as shown below.

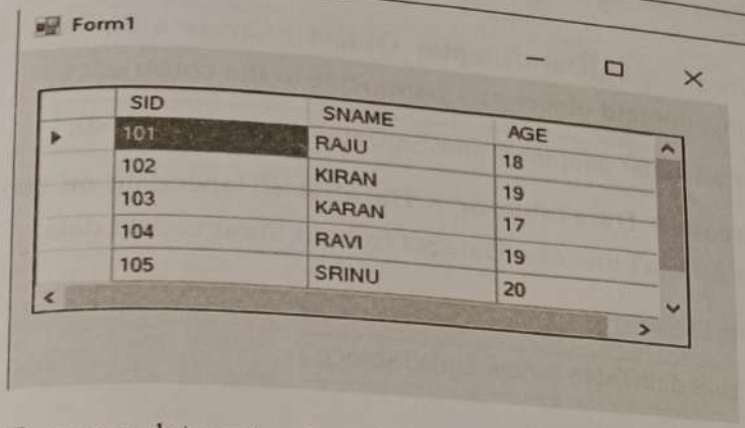


Step-9 : To display the data in the dataset on to the window form use a DataGridView control. Set the DataGridView Data Source property to Dataset11 and its Data Member property to Student. This connects the data in the dataset to the DataGridView as shown in below figure.



Step-10 : To fill the dataset with data from the Data Adapter use the Data Adapter fill method.

```
private void Form1_Load(object sender, EventArgs e)
    {
        dataSet11.Clear();
        sqlDataAdapter1.Fill(dataSet11);
}
```



Method-2 : To access data using DataAdapters and Datasets in C# with ADO.NET **at Runtime using code**, you can follow these steps:

Step-1 : Create a SqlConnection Object : Establish a connection to the SQL Server database by creating a SqlConnection object. Provide the appropriate connection string with the necessary credentials and database information.Use appropriate data provider. In this example, we'll use SqlConnection from the System.Data.SqlClient namespace.

```
using System.Data;
using System.Data.SqlClient;
string connectionString = "Your Connection String";
using (SqlConnection connection = new SqlConnection(connectionString))
    {
connection.Open();
// Step 2, 3, and 4 go here
    }
```

Step-2 : Create a Sql Command Object : Create a SqlCommand object and specify the SQL query or stored procedure that retrieves data from the database. Pass the SQL query and the SqlConnection object as parameters to the SqlCommand constructor.

```
string query = "SELECT * FROM YourTableName";
using (SqlCommand command = newSqlCommand(query, connection))
    {
// Step 3 and 4 go here
    }
```

Step-3 : Create a SqlDataAdapter Object : Create a SqlDataAdapter object and pass the SqlCommand object as a parameter to the constructor.

```
SqlDataAdapter adapter = new SqlDataAdapter(command);
```

Step-4 : Create a DataTable or a DataSet : Depending on your needs, you can create either a DataTable or a DataSet to hold the retrieved data.

To use a DataTable:

```
DataTable dataTable = new DataTable();
```

To use a DataSet:

```
DataSet dataSet = new DataSet();
```

Step-5 : Fill the DataTable or DataSet with data: Use the DataAdapter's Fill method to retrieve data from the database and fill the DataTable or DataSet.

```
adapter.Fill(dataTable); // or adapter.Fill(dataSet);
```

Step-6 : Access and work with the data: Once the DataTable or DataSet is filled with data, you can access and manipulate the data as needed.

Using a DataTable :

```
foreach (DataRow row in dataTable.Rows)
    {
// Access individual columns using the column name or index
var name = row["Name"];
var pinNo = row["PinNo"];
var age = row["Age"];
// Perform desired operations with the data
// For example, you can print the data to the console
Console.WriteLine($"Name: {name}, PinNo: {pinNo}, Age: {age}");
    }
```

Using a DataSet :

```
foreach (DataRow row in dataSet.Tables[0].Rows){
// Access individual columns using the column name or index
```



```
var name = row["Na
var pinNo = row["Pi
var age = row["Age
// Perform desired
// For example, yo
Console.WriteLine
    }
```

Step-7 : Handle excep

exceptions that might
Finally, close the conn

```
// Exception han
catch (Exceptio
// Handle the ex
Console.WriteLi
    }
// Connection c
finally{
connection.Clo
    }
```

Example Program

```
using System;
using System.
using System.
class Program
{
static void Ma
    {
```

```
var name = row["Name"];
var pinNo = row["PinNo"];
var age = row["Age"];
// Perform desired operations with the data
// For example, you can print the data to the console
Console.WriteLine($"Name: {name}, PinNo: {pinNo}, Age: {age}");
    }
```

Step-7 : Handle exceptions and close the connection: It's important to handle any exceptions that might occur during the database connection or data retrieval process. Finally, close the connection to release resources.

```
// Exception handling
catch (Exception ex){
// Handle the exception
Console.WriteLine("Error: " + ex.Message);
    }
// Connection closing
finally{
connection.Close();
    }
```

Example **Program :**

```
using System;
using System.Data;
using System.Data.SqlClient;
class Program
{
static void Main()
  {
```

```
string connectionString = "Data Source=DESKTOP-2QP70QE\SQLEXPRESS; Initial
Catalog=studentDB; Integrated Security=True";
string query = "SELECT * FROM Student";

// Step 1: Establish connection
using (SqlConnection connection = new SqlConnection(connectionString))
      {
try
        {
            connection.Open();
// Step- 2 : Create DataAdapter
            SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
// Step-3 : Create DataTable
DataTable dataTable = new DataTable();
// Step 4: Fill DataTable with data
adapter.Fill(dataTable);
// Step-5 : Access and work with the data
foreach (DataRow row indataTable.Rows){
var name = row["Name"];
var pinNo = row["PinNo"];
var age = row["Age"];
Console.WriteLine($"Name: {name}, PinNo: {pinNo}, Age: {age}");
          }
        }
catch (Exception ex){
Console.WriteLine("Error: " + ex.Message);
  }
  finally{
```

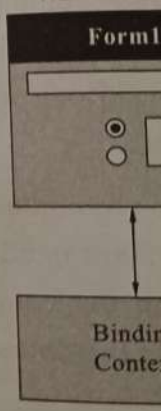


```
// Step-6 : Close c
    connection.C
        }
      }
    }
  }
```

6.7 DATA B

Data binding is
synchronization
need for manual
data binding allo
writing explicit c

Binding data to



Binding Cont
BindingConte
data source a
control and th

Each Windo
theCurrencyM
there is a sing

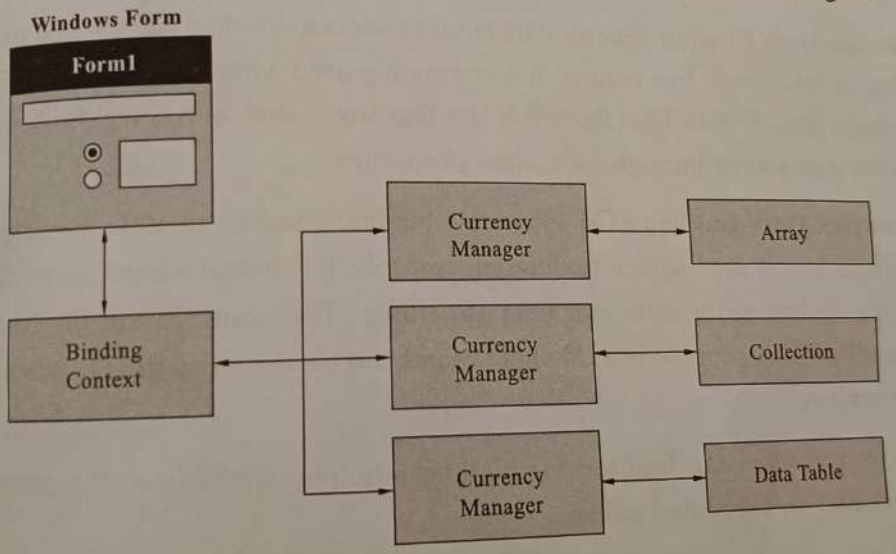
WARNING

```
// Step-6 : Close connection
    connection.Close();
            }
        }
    }
}
```

6.7 DATA BINDING TO DATAGRID, TEXTBOX AND LISTBOX UI CONTROLS

Data binding is a powerful feature in software development that enables automatic synchronization of data between different UI controls and data sources, eliminating the need for manual updates. In the context of a DataGrid, TextBox, and ListBox UI controls, data binding allows you to easily display and manipulate data from a data source without writing explicit code to handle the synchronization.

Binding data to a windows form control can be explained using below figure.



Binding Context : In Windows Forms, data binding can be achieved by using the **BindingContext** and **CurrencyManager** classes to establish a connection between a data source and a control. This allows automatic synchronization of data between the control and the underlying data source.

Each Windows Form has at least one BindingContext object that manages theCurrencyManagerobjects for the form. For each data source on a Windows Form, there is a single CurrencyManager object. Because there may be multiple data sources

associated with a Windows Form, the BindingContext object enables you to retrieve any particular CurrencyManager object associated with a data source.

CurrencyManager class Properties :

S.No.	Property	Meaning
1.	Bindings	Gets the collection of bindings being managed
2.	Count	Gets the number of items in the list
3.	Current	Gets the current item in the list.
4.	List	Gets the list for this CurrencyManager.
5.	Position	Gets or sets the position you are at within the list.

The user can bind values to the respective controls in ADO.NETdepending on the type of binding offered, they are distinguished asfollows:

1. Simple Data Binding
2. Complex Data Binding

1. **Simple Data Binding :** Simple data binding refers to binding a single value or property from a data source to a control. It is commonly used when you want to display or edit a single piece of data.The UI controls like TextBox, Label, and CheckBox can be bound to the data source through the control properties.

2. **Complex Data Binding :** Complex data binding involves binding multiple properties or fields from a data source to different controls. It is useful when you need to display or manipulate more extensive data structures. The controls can be DataGridView, Dropdown list, or combo box. Multiple values can be displayed from the dataset through the binding.

The controls that can be used for binding multiple values from the database to the Windows Form are listed below.

- **DataGridView :** It is used to display the multiple records and columns. The ***DataSource*** property of the DataGridView control is used for binding the specific data element.
- **ComboBox :** This control contains a text box for entering the data and drop down list for displaying the values. The ***DataSource*** property is useful for binding the control. The element specific information can be bind through the***DisplayMember*** property.

- **ListBox :** It is used for displaying the data for the column fromseveral records of datasets. The ***DataSource*** property is usedfor binding the control to the data source.The ***DisplayMember*** property is used for binding the control tothe specific data element.

Before binding data to DataGrid, DataGridView, TextBox and ListBox controls first connect to SQL Server database.

Connecting to the SQL Server database involves creating a Connection, Data Adapter, and a Dataset. To connect to the SQL Server database, perform the following steps:

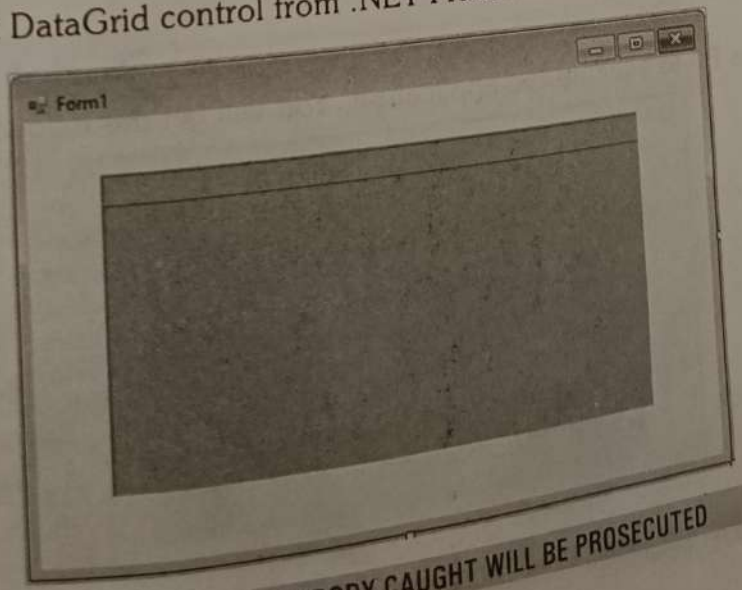
- Create a DataAdapter (Ex: SqlDataAdapter1) through the wizard. On the second screen of the wizard, click the New Connection button and establish a connection with SQL Server database. Add the table (Ex: Student) and select all its fields for the SELECT command in Query Builder.
- Generate a new Dataset object (Ex:studentDBDataSet).

Now you can bind the data to DataGrid, TextBox, ListBox controls in the following manner.

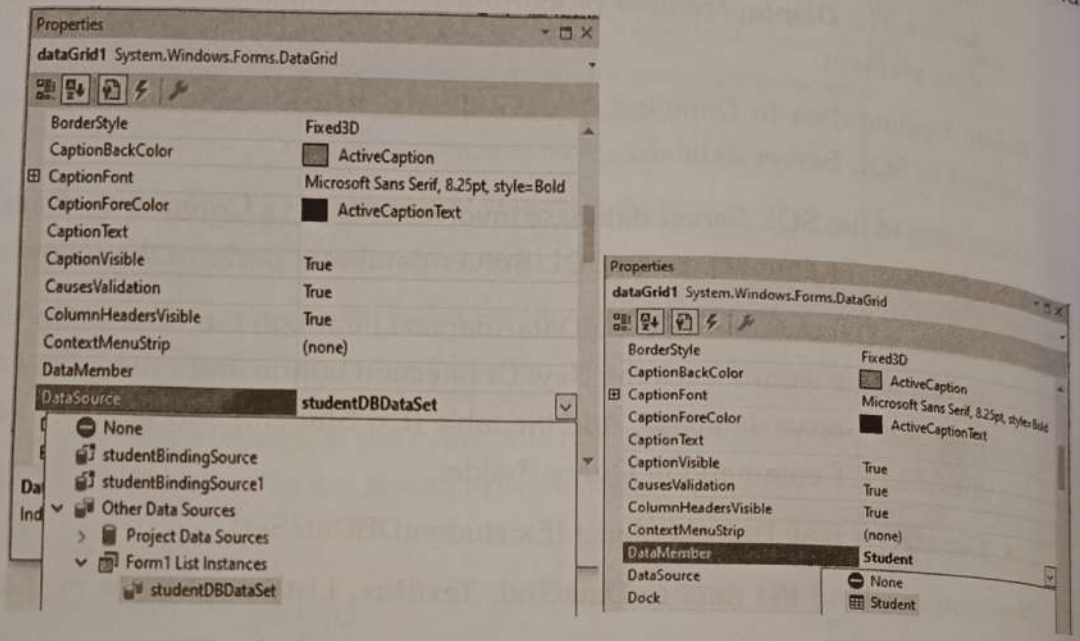
Binding Data to DataGrid Control

The DataGrid control is bound to the data source collection by setting its ***DataSource*** property to the collection. This automatically generates columns in the DataGrid, with each column representing a property of the objects in the collection, you can follow these steps :

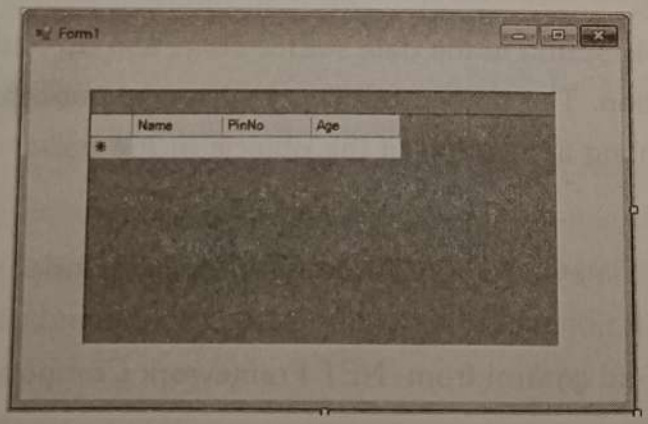
1. Drag and drop a DataGrid control from the Toolbox under the Data tab. If the DataGrid control is not available right click on Data tab and click on Choose Items and Select DataGrid control from .NET Framework Components.



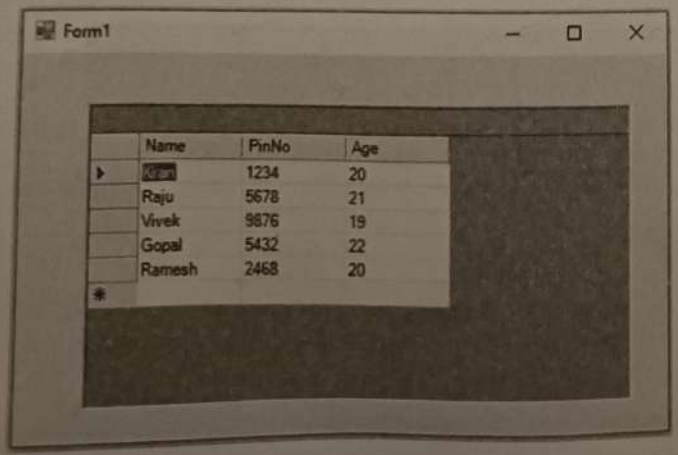
2. Go to the properties of DataGrid control and set both DataSource property and DataMember property like below.



3. Now the form is displayed, as shown in the following figure.



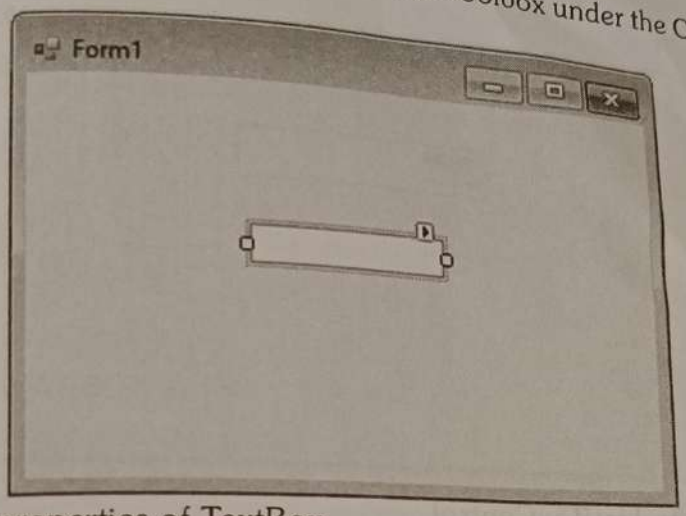
4. Press **F5** to execute the application. You will get the following output.



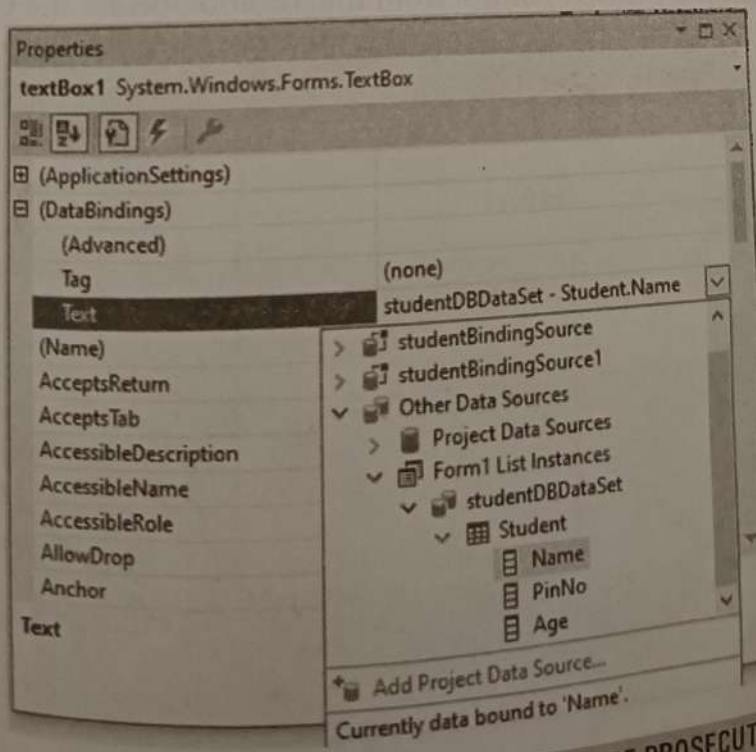
Binding Data to TextBoxControl

The *Text* property of the TextBox control is bound to the "Name" property in the data source object. Any changes made in the TextBox will automatically update the "Name" property, and vice versa. You can follow these steps:

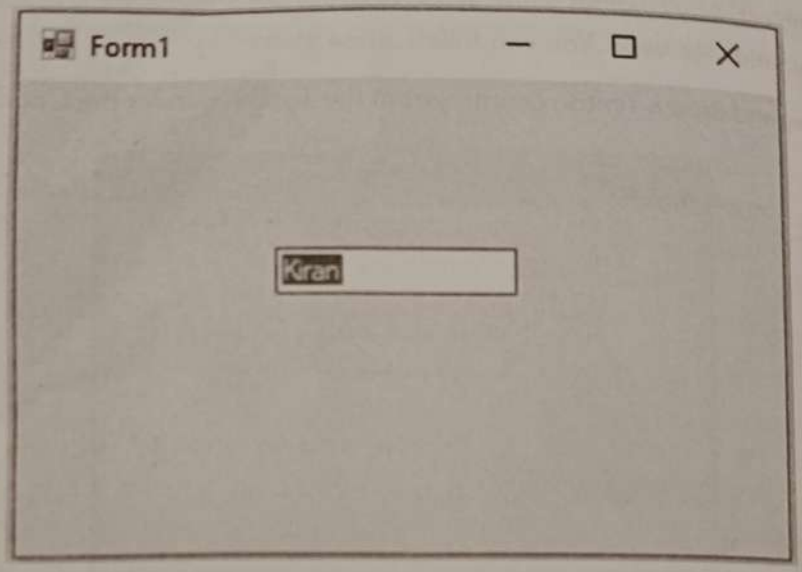
1. Drag and drop a TextBox control from the Toolbox under the Common Controls.



2. Go to the properties of TextBox
   (a) Expand the (DataBindings) node.
   (b) Click the arrow next to the Text property. The DataSource UI type editor opens. If a data source has previously been configured for the project or form, it will appear. Select the Data source in the following manner.



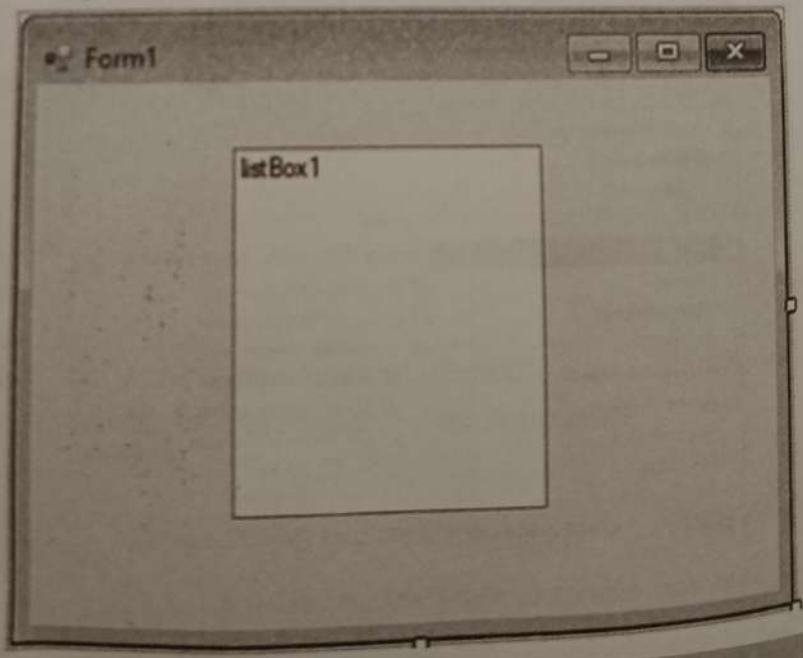
3. Press **F5** to execute the application. You will get the following output. As we configured the Student name(Name) column it will display only the first name in the Student table.



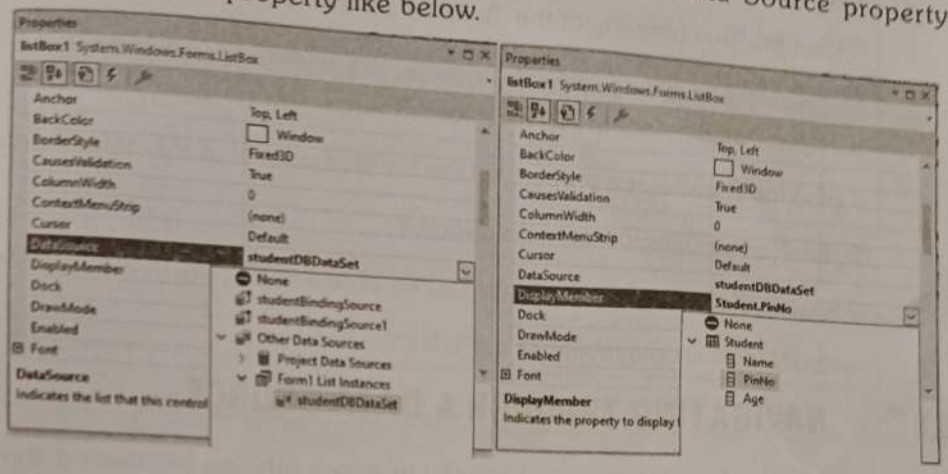
III. Binding Data to ListBoxControl

The ***DataSource*** property of the ListBox control is bound to the data source collection. The ***DisplayMember*** property is set to "PinNo" to specify which property in the data source object should be displayed for each item in the ListBox. You can follow these steps:

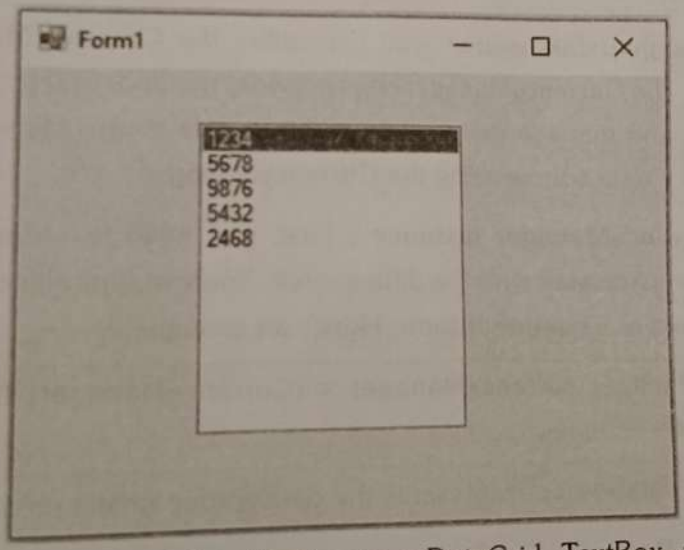
1. Drag and drop a ListBox control from the Toolbox under the Common Controls.



2. Go to the properties of ListBox and set both Data Source property and DisplayMember property like below.



3. Press **F5** to execute the application. You will get the following output. As we configured the Student Pin number (PinNo) column it will display the Pin numbers of Student table.



Here's an example of data binding to a DataGrid, TextBox, and ListBox UI controls using C# through coding :

```
//Assuming you have a DataGrid control named "dataGrid", TextBox control named
//"txtName",ListBox control named "listBoxNames", and a data source as a collection
//of objects

List<MyObject> dataSource = new List<MyObject>();

// Set the data source for the DataGrid control
```

```
dataGrid.DataSource = dataSource;
// Bind the Text property of the TextBox control to the "Name" property in the data source object
txtName.DataBindings.Add("Text", dataSource, "Name");
// Bind the DataSource property of the ListBox control to the data source collection
listBoxNames.DataSource = dataSource;
// Specify the property in the data source object to display in the ListBox
listBoxNames.DisplayMember = "PinNo";
```

6.8 NAVIGATING THROUGH A DATA SOURCE

Navigating through a data source typically involves moving between different records or items within the data source. The ability to navigate through a data source is crucial for accessing and manipulating specific data elements. The specific method for navigation depends on the type of data source being used.

To navigate through a data source, you can utilize the **CurrencyManager** class in Windows Forms. The CurrencyManager class provides methods and properties to move between records and manage the position within a data source.Here's how you can navigate through a data source using the CurrencyManager :

1. **Obtain the CurrencyManager instance :** First, you need to obtain the Currency Manager instance associated with the data source. You can typically obtain it through the BindingContext of a control or form. Here's an example :

```
CurrencyManager currencyManager = (CurrencyManager) BindingContext [dataSource];
```

In this example, "dataSource" represents the data source object or collection.

2. **Moving Through Records :** The CurrencyManager provides several methods to move through the records in the data source. Here are some commonly used methods :

- **MoveFirst() :** Moves the cursor to the first record.
- **MovePrevious() :** Moves the cursor to the previous record.
- **MoveNext() :** Moves the cursor to the next record.
- **MoveLast() :** Moves the cursor to the last record.
- **Position :** Gets or sets the current position in the data source (0-based index).

Here's an example of moving through records:

```
// Move to the first record
currencyManager.Position = 0;
// Move to the next record
currencyManager.MoveNext();
// Move to the previous record
currencyManager.MovePrevious();
// Move to the last record
currencyManager.Position = currencyManager.Count - 1;
```

3. **Accessing current Record :** You can access the current record using the Current property of the CurrencyManager. This returns the object representing the current record in the data source. For example :

```
MyObject currentRecord = (MyObject)currencyManager.Current;
```

In this example, "MyObject" represents the type of the objects in the data source.

Example :

To design a Windows Form for displaying data, retrieved from a SQL Server database, perform the following steps :

(i) Create a new Windows Forms Application project by clicking the New project button in Microsoft Visual Studio Development Environment. Form1 is added to the project by default.

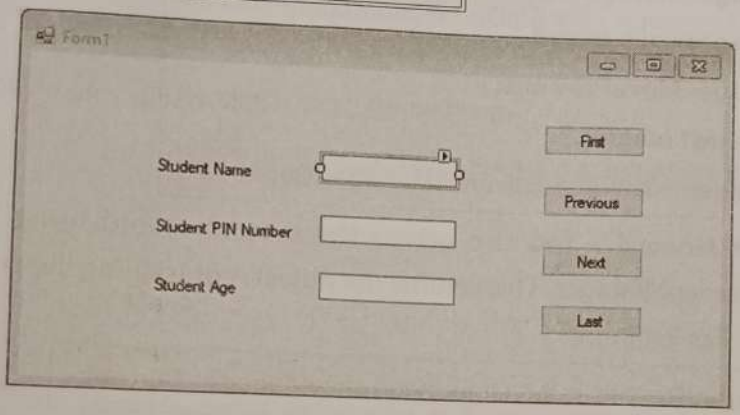
(ii) Select View from the menu bar, and then select Toolbox from the View menu. The Toolbox is displayed. From the Windows Forms tab of the Toolbox, drag three Label controls and three TextBox controls, to the form. Set the Text property of the label controls as displayed in the following table :

Label	Text Property
Label1	Student Name
Label2	Student PIN No
Label3	Student Age

TextBox	Name
TextBox1	txtName
TextBox2	txtPinNo
TextBox3	txtAge

(iii) Drag four Button controls to the form. Set the Name and Text property for each button as displayed in the following table :

Button	Name	Text
Button1	btnFirst	First
Button2	btnPrev	Previous
Button3	btnNext	Next
Button4	btnLast	Last

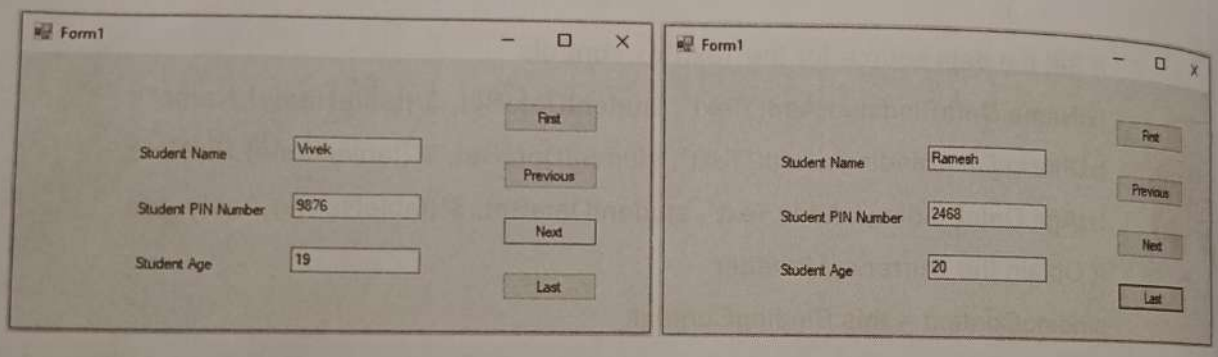



```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;
namespace StudentTableNavigation
{
public partial class MainForm : Form
  {
private DataSet studentDataSet;
private CurrencyManager currencyManager;
private BindingContext bindingContext;

public MainForm(){
InitializeComponent();
    }
private void MainForm_Load(object sender, EventArgs e)
    {
// Initialize the data source (sample student records)
```

```
studentDataSet = new DataSet();
// Fill the DataSet with data from the "Student" table
string connectionString = "Data Source=DESKTOP-2QP70QE\\SQLEXPRESS;Initial
Catalog=studentDB;Integrated Security=True";
string tableName = "Student";
using (var connection = new SqlConnection(connectionString))
{
var adapter = new SqlDataAdapter($"SELECT * FROM {tableName}", connection);
adapter.Fill(studentDataSet, tableName);
        }
// Set the data source for the TextBox controls
txtName.DataBindings.Add("Text", studentDataSet, $"{tableName}.Name");
txtPinNo.DataBindings.Add("Text", studentDataSet, $"{tableName}.PinNo");
txtAge.DataBindings.Add("Text", studentDataSet, $"{tableName}.Age");
// Obtain the CurrencyManager
bindingContext = this.BindingContext;
currencyManager = (CurrencyManager)bindingContext[studentDataSet, tableName];
    }
private void btnFirst_Click(object sender, EventArgs e){
// Move to the first record
        currencyManager.Position = 0;
    }
private void btnPrev_Click(object sender, EventArgs e){
// Move to the previous record
if (currencyManager.Position > 0)
            currencyManager.Position – –;
    }
private void btnNext_Click(object sender, EventArgs e){
// Move to the next record
if (currencyManager.Position < currencyManager.Count - 1)
            currencyManager.Position++;
```

```
        }
    private void btnLast_Click(object sender, EventArgs e){
    // Move to the last record
            currencyManager.Position = currencyManager.Count - 1;
        }
    }
}
```

Output :

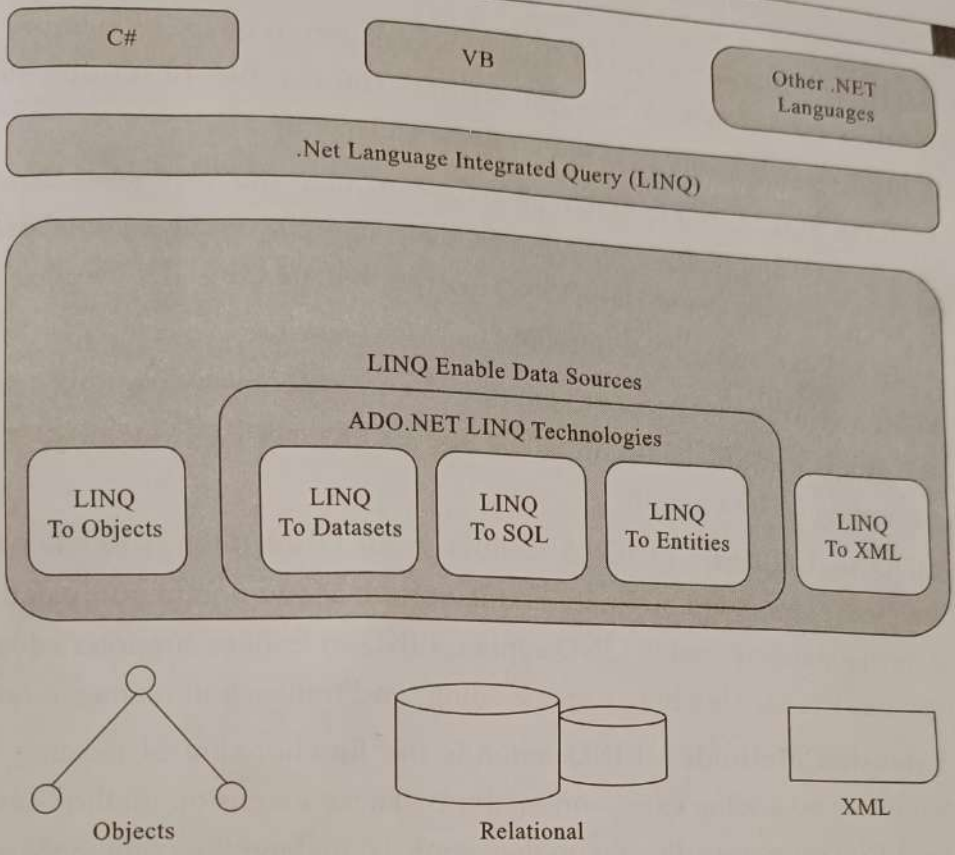


6.9 FAMILIARIZE TO LINQ

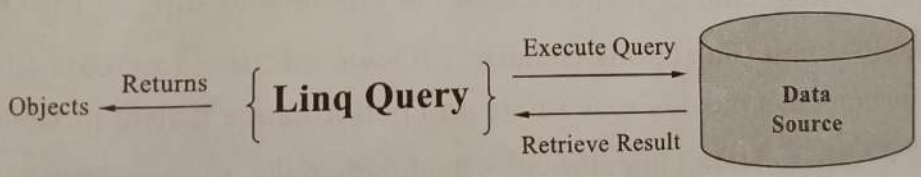
LINQ (Language Integrated Query)introduced in Microsoft Visual Studio 2008, is a powerful feature in the .NET framework that allows developers to query and manipulate data from different data sources using a consistent and expressive syntax. LINQ provides a unified way to work with data regardless of the data source, whether its collections, databases, XML, or other data formats.

Even though you don't know query languages(SQL,XML,…) you can write queries by using LINQ (In C# and VB.NET), because you write queries by using language keywords and familiar words.It is integrated in C# & VB.NET, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.

For example, SQL is a Structured Query Language used to save and retrieve data from databases. In the same way, LINQ is a structured query syntax built in C# & VB.NET to retrieve data from different types of data sources such as collections, ADO.NET Datasets, XML documents, web services, MS SQL Server and other databases.



LINQ queries return results as objects. It enables you to uses object-oriented approach on the result set and not to worry about transforming different formats of results into objects.



Here are some key concepts and components of LINQ :

- **Query Expressions :** LINQ provides a query syntax that resembles SQL-like syntax, making it easy to express queries in a readable and intuitive way. Query expressions begin with the **from keyword** followed by a **range variable**, and can include various query clauses like where, select, orderby, groupby, and join.
- **Standard Query Operators :** LINQ provides a set of standard query operators that can be used with different data sources. These operators include filtering (e.g., Where), sorting (e.g., OrderBy), projecting (e.g., Select), joining (e.g., Join), grouping (e.g., GroupBy), aggregating (e.g., Sum, Count), and more. These operators can be chained together to compose complex queries.

- **LINQ to Objects :** LINQ to Objects allows you to query in-memory collections, such as arrays, lists, or custom objects. You can use LINQ operators to filter, transform, or sort the data within these collections.
- **LINQ to SQL :** LINQ to SQL enables you to query and manipulate data in relational databases using LINQ syntax. It provides a mapping between database tables and objects, allowing you to write LINQ queries that are translated into SQL statements and executed against the database.
- **LINQ to XML :** LINQ to XML provides a convenient way to query, create, modify, and transform XML data. It allows you to traverse XML hierarchies and extract data using LINQ operators.
- **LINQ to Entities :** LINQ to Entities is an ORM (Object-Relational Mapping) technology provided by Entity Framework. It allows you to query and manipulate data in a database using LINQ syntax. LINQ to Entities supports advanced query features like lazy loading, eager loading, and transaction management.
- **Extension Methods :** LINQ extends the functionality of existing classes and interfaces by adding extension methods. These extension methods enable you to call LINQ operators directly on data sources, making the code more readable and concise.

Core Assemblies in LINQ

1. **using System.Linq :** Provides Classes & Interface to support LINQ Queries
2. **using System.Collections.Generic :** Allows the user to create Strongly Typed collections that provide type safety and performance (LINQ to Objects)
3. **using System.Data.Linq :** Provides the functionality to access relational databases (LINQ to SQL)
4. **using System.Xml.Linq :** Provides the functionality for accessing XML documents using LINQ (LINQ to XML)
5. **using System.Data.Linq.Mapping :** Designates a class as an entity associated with a database.

6.10 SYNTAX OF LINQ

LINQ introduces a set of standard query operators that can be used with any data source that implements the **IEnumerable** or **IQueryable** interfaces. These operators enable you to perform various operations on data, such as filtering, sorting, grouping, projecting, joining, and aggregating.

The syntax of LINQ (Language Integrated Query) involves using **query expressions** or **method syntax** to perform data queries and manipulations.

1. Query Syntax or Query Expression Syntax
2. Method Syntax or Method Extension Syntax or Fluent Syntax

1. **LINQ Query Syntax :** The Query syntax allows you to write LINQ queries in a declarative and SQL (Structured Query Language) -like manner. It starts with the from clause, followed by optional where, orderby, groupby, and select clauses.

```
from<range variable> in <IEnumerable<T> or IQueryable<T> Collection>
<standardqueryoperators><lambda expression>
<select or groupby or orderby operator>
<result formation>
```

The LINQ query syntax starts with **from** keyword and ends with **select** keyword.

Example :

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var query = from num in numbers
where num%2 == 0
orderby num descending
select num;
foreach (var num in query){
Console.WriteLine(num);
}
```

The following is a sample LINQ query that returns a collection of strings which contains a word "Tutorials".

EXAMPLE-1

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace LINQQuerySyntaxEx
{
class Program
  {
```

```
static void Main(string[] args)
    {
// string collection
var strings = new List<string>
    {
"Welcome to Tutorials Point",
"Learn from Tutorials",
"Practical experiments are great",
"Explore the Tutorials section"
    };

// LINQ Query Syntax
var result = from s in strings
where s.Contains("Tutorials")
select s;
foreach (var str in result){
Console.WriteLine(str);
    }
  }
 }
}
```

Output :

```
Welcome to Tutorials Point
Learn from Tutorials
Explore the Tutorials section
```

In the following example, we use LINQ query syntax to find out teenager students from the Student collection (sequence).

EXAMPLE-2

```
using System;
using System.Linq;
using System.Collections.Generic;
```

ut teenager students from


```
public class Student
{
public int StudentID { get; set; }
public string StudentName { get; set; }
public int Age { get; set; }
}
public class Program
{
public static void Main()
   {
var students = new List<Student>
      {
new Student() { StudentID = 1, StudentName = "Srinu", Age = 13},
new Student() { StudentID = 2, StudentName = "Kiran",  Age = 21},
new Student() { StudentID = 3, StudentName = "Raju",  Age = 18},
new Student() { StudentID = 4, StudentName = "Ram" , Age = 20},
new Student() { StudentID = 5, StudentName = "Rana" , Age = 15}
      };
// LINQ Query Syntax to find out teenager students
var teenagerStudents = from student in students
where student.Age>= 13 &&student.Age<= 19
select student;
Console.WriteLine("Teen age Students are:");
foreach (var student in teenagerStudents){
Console.WriteLine(student.StudentName);
      }
   }
}
```

Output :

```
Teen age Students are :
Srinu
Raju
Rana
```

2. **LINQ Method Syntax :** The Method syntax(also known as fluent syntax)uses **extension methods**and **lambda expressions** to chain LINQ operators together. It is a more fluent and concise way to write LINQ queries compared to query expressions. You start with a data source, apply various LINQ operators, and end with a terminal operator like ToList(), ToArray(), FirstOrDefault(), etc. Here's the same example using method syntax:

```
var query = collection
.[operator1]
.[operator2]
.[operator3]
   ...
.[terminal operator];
```

Example :

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var query = numbers
    • Where(num =>num % 2 == 0)
    • OrderByDescending(num =>num);
foreach (var num in query){
Console.WriteLine(num);
}
```

The following is a sample LINQ method syntax query that returns a collection of strings which contains a word "Tutorials".

EXAMPLE-1

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace LINQMethodSyntaxEx
```

```
{
class Program
  {
public static void Main()
    {
// string collection
var strings = new List<string>
        {
"Welcome to Tutorials Point",
"Learn from Tutorials",
"Practical experiments are great",
"Explore the Tutorials section"
        };
// LINQ Method Syntax
var filteredstrings = strings.Where(s =>s.Contains("Tutorials")).ToList();
foreach (var str in filtered Strings)
        {
Console.WriteLine(str);
        }
    }
  }
}
```

Output :

```
Welcome to Tutorials Point
Learn from Tutorials
Explore the Tutorials section
```

The following example shows how to use LINQ method syntax to find the teenager students from a collection of Student objects:

EXAMPLE-2

```
using System;
using System.Linq;
using System.Collections.Generic;
```

```
public class Student
{
public int StudentID { get; set; }
public string StudentName { get; set; }
public int Age { get; set; }
}
public class Program
{
public static void Main()
  {
// Student collection
var students = new List<Student>
    {
new Student() { StudentID = 1, StudentName = "Srinu", Age = 13},
new Student() { StudentID = 2, StudentName = "Raju",  Age = 21},
new Student() { StudentID = 3, StudentName = "Kiran",  Age = 18},
new Student() { StudentID = 4, StudentName = "Ram" , Age = 20},
new Student() { StudentID = 5, StudentName = "Rana" , Age = 15}
    };
// LINQ Query Method Syntax to find out teenager students
var teenagerStudents = students.Where(student =>student.Age>= 13 && student.
Age<=19).ToList();
foreach (var student in teenagerStudents)
    {
Console.WriteLine(student.StudentName);
    }
  }
}
```


Output :

Srinu
Kiran
Rana

6.11 TY

LINQ (Lang
querying an

1. **Enumerabl**
implement t
or any othe
using LINQ
operators
methodsdef
like filtering

2. **Queryable**
databases,
which exte
providers l
of LINQ
objects en
queries by

3. **Anonymo**
class decl
shape of
new keyw
results an

4. **Groupin**
data. The
in LINQ
contains
are usefu
sums or

Output :

```
Srinu
Kiran
Rana
```

6.11 TYPES OF LINQ OBJECTS

LINQ (Language Integrated Query) encompasses various types of objects that facilitate querying and manipulating data. Here are the main types of LINQ objects :

1. **Enumerable Objects :** Enumerable objects represent in-memory collections that implement the **IEnumerable<T>** interface. These can include arrays, lists, dictionaries, or any other collection type. LINQ allows you to query and manipulate these objects using LINQ extension methods such as Where, Select, OrderBy, and more. LINQ operators can be directly applied to enumerable objects using LINQ extension methodsdefined in the **System.Linq** namespace, allowing you to perform operations like filtering, sorting, projecting, and aggregating data.

2. **Queryable Objects :** Queryable objects represent queryable data sources such as databases, data sets, or remote services. They implement the **IQueryable<T>** interface, which extends the **IEnumerable<T>** interface. Queryable objects work with LINQ providers like LINQ to SQL, LINQ to Entities, or LINQ to XML, enabling the translation of LINQ queries into appropriate queries for the underlying data source. Queryable objects enable query composition and deferred execution, allowing you to build complex queries by chaining LINQ operators.

3. **Anonymous Types :** Anonymous types are dynamically created types without an explicit class declaration. They are useful for defining and projecting query results when the shape of the result is not known beforehand. Anonymous types are created using the **new** keyword and an **object initializer**. They provide a convenient way to shape query results and select specific properties from the data source.

4. **Grouping Objects :** Grouping objects are used when performing group operations on data. They are used to group elements based on a common key. The **GroupBy** operator in LINQ returns a sequence of **IGrouping<TKey, TElement>**objects, where each group contains a key and a collection of elementsthat share the same key. Grouping objects are useful for performing aggregate operations within each group, such as calculating sums or counts.

5. **Joining Objects :** Joining objects are used to combine data from multiple sources based on specified join conditions. The **Join operator** in LINQ allows you to join two or more collections based on matching keys and project the combined results. The Join operator in LINQ performs inner joins, while other operators like GroupJoin, LeftJoin, or FullJoin offer different types of join operations. Joining objects enable data merging and combining operations, allowing you to establish relationships between data from different sources.
6. **Ordered Objects :** Ordered objects are used to perform sorting operations on data. The **OrderBy** and **ThenBy** operators in LINQ allow you to sort data in ascending order, while **OrderByDescending** and **ThenByDescending**operators enable sorting in descending orderbased on specified keys. Ordered objects maintain the order of elements according to the specified sorting conditions.Ordered objects are useful when you need to retrieve data in a particular order.
7. **XML Objects :** LINQ to XML provides XML-specific objects and types for querying, creating, modifying, and transforming XML data. These objects include **XDocument, XElement, XAttribute, XNamespace**, and more. LINQ to XML allows you to work with XML documents using LINQ syntax, making it easy to query and manipulate XML data.

6.12 ADVANTAGES OF LINQ

LINQ (Language Integrated Query) provides several advantages that make it a powerful and versatile tool for data querying and manipulation in .NET applications. Here are some of the key advantages of LINQ :

- **Familiar Language :** Developers dont have to learn a new query language for each type of data source or data format.
- **Simplified and Readable Code :** LINQ provides a clear and concise declarative syntax that resembles natural language, improving code readability and maintainability.LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Less Coding :** It reduces the amount of code to be written as compared with a more traditional approach.
- **Shaping data:** You can retrieve data in different shapes.
- **Standardized way of querying multiple data sources :** LINQ supports querying databases, XML documents, in-memory collections, and more, providing unified query syntax for different data sources. The same LINQ syntax can be used to query multiple data sources.

- **Type Safety and Compile-Time Checking :** LINQ is strongly typed, provides type checking of objects at compile time, allowing for early detection of errors at compile time and reducing runtime exceptions.
- **Language Integration :** LINQ seamlessly integrates with many .NET languages, leveraging language features and eliminating the need for separate query languages.
- **Code Reusability :** LINQ promotes code reusability by encapsulating query logic into reusable query expressions or methods.
- **Integration with Visual Studio :** LINQ integrates well with Visual Studio, providing features like IntelliSense, debugging support, and query profiling for enhanced development experience.
- **Flexibility and Extensibility :** LINQ offers a rich set of standard query operators and supports custom operators, allowing developers to tailor queries and integrate with custom data sources or behaviours.
- **Deferred Execution and Optimization :** LINQ supports deferred execution, optimizing query execution and reducing unnecessary data retrieval for improved performance.

6.13 VARIOUS LINQ OPERATORS

LINQ (Language Integrated Query) provides a rich set of operators that enable querying, filtering, transforming, and aggregating data. These operators allow you to express complex data operations in a concise and readable manner when working with collections and other data sources. LINQ provides more than 50 query operators for different functionalities. Every query operator is an extension method. Here are some of the commonly used LINQ operators :

S.No.	Operator Category	LINQ Query Operator Names
1.	Filtering	Where, OfType, Take, Skip
2.	Projection	Select, SelectMany, SelectIndex
3.	Sorting/Ordering	OrderBy, OrderByDescending, ThenBy, ThenByDescending
4.	Join	Join, GroupJoin
5.	Set	Distinct, Union, Intersect, Except
6.	Aggregation	Count, Sum, Average, Min, Max, Aggregate
7.	Partitioning	Take, Skip, TakeWhile, SkipWhile,TakeLast, SkipLast
8.	Conversion	ToArray, ToList, ToDictionary, ToLookup, Cast, AsEnumerable
9.	Quantifier	All, Any, Contains

10.	Grouping	GroupBy, ToLookup, GroupJoin
11.	Sequencing	Range, Repeat, Empty
12.	Equality	Equals, SequenceEqual
13.	Element	First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault, ElementAt, ElementAtOrDefault
14.	Concatenation	Concat, Union, Zip

Below is basic explanation of important LINQ operators with examples.

1. **Filtering Operators :** LINQ provides filtering operators that allow you to select elements from a sequence based on specified conditions. These operators enable you to filter out unwanted elements and retrieve only the elements that meet specific criteria. For example, suppose we have ten names in a collection sequence and we have to filter out those names that start with letter K.

Here are some commonly used filtering operators in LINQ with examples

S.No.	Filtering Operators	Description
1.	Where()	Filters a sequence based on a specified condition and returns a new sequence that contains only the elements that satisfy the condition. It allows you to retrieve elements that match a particular criterion.
2.	OfType()	Filters a sequence and returns only the elements of a specific type. It allows you to filter out elements that are not of a particular type from a heterogeneous sequence.
3.	TakeWhile()	Retrieves elements from the beginning of a sequence while a specified condition is true. It returns a new sequence containing the elements until the condition is false.
4.	SkipWhile()	It skips elements from the beginning of a sequence until a specified condition is false, and then returns the remaining elements.

Example Program using Filtering Operators :

```
using System;
using System.Collections.Generic;
using System.Linq;
class Program
{
class Student{
public string Name { get; set; }
public int Age { get; set; }
```

```
    }
static void Main(string[] args)
    {
// Sample list of students
        List<Student> students = new List<Student>()
        {
new Student { Name = "Kiran", Age = 18 },
new Student { Name = "Raju", Age = 20 },
new Student { Name = "Vivek", Age = 17 },
new Student { Name = "Shilpa", Age = 19 },
new Student { Name = "Ramesh", Age = 18 }
        };
// Using Where() to filter students older than 18
var older Students = students.Where(s =>s.Age> 18);
Console.WriteLine("Students older than 18:");
foreach (var student in olderStudents){
Console.WriteLine($"Name: {student.Name}, Age: {student.Age}");
        }
// Using OfType() to filter numbers from a mixed list
        List<object> mixedList = new List<object>()
        {
            1, "two", 3, "four", 5
        };
var numbers = mixedList.OfType<int>();
Console.WriteLine("\nNumbers in the mixed list:");
foreach (var number in numbers){
Console.WriteLine(number);
        }
// Using TakeWhile() to retrieve students until a condition is true
var studentsUntilAge20 = students.TakeWhile(s =>s.Age<= 20);
Console.WriteLine("\nStudents until age 20:");
foreach (var student in studentsUntilAge20){
Console.WriteLine($"Name: {student.Name}, Age: {student.Age}");
```

```
        }
    }
}
```

Output :

```
Students older than 18:
Name: Raju, Age: 20
Name: Shilpa, Age: 19
Numbers in the mixed list:
1
3
5
Students until age 20:
Name: Kiran, Age: 18
Name: Raju, Age: 20
Name: Vivek, Age: 17
Name: Shilpa, Age: 19
Name: Ramesh, Age: 18
```

2. **Projection Operators :** LINQ provides projection operators that allow you to transform the elements of a sequence into a new form or shape. These operators enable you to extract specific properties or perform calculations on elements, generating a new sequence with the projected results.

S.No.	Projection Operators	Description
1.	**Select()**	Transforms each element of a sequence into a new form. It applies a specified function to each element and returns a new sequence containing the projected results.
2.	**SelectMany()**	Projects each element of a sequence to another sequence and then flattens the resulting sequences into one sequence. It is useful when working with nested collections or when you want to combine multiple sequences into one.
3.	**SelectIndex()**	Projects each element in a sequence into a new form that includes the index of the element. It can be useful when you need to access the index along with the element.

Example Program using Projection Operators :

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
class Program
{
class Product
  {
public string Name { get; set; }
public decimal Price { get; set; }
  }
static void Main(string[] args)
  {
// Sample list of products
      List<Product> products = new List<Product>()
      {
new Product { Name = "Apple", Price = 100.00m },
new Product { Name = "Banana", Price = 60.00m },
new Product { Name = "Orange", Price = 50.50m },
new Product { Name = "Grapes", Price = 70.50m }
      };
// Using Select() to project product names
var productNames = products.Select(p =>p.Name);
Console.WriteLine("Product Names:");
foreach (var name in productNames)
      {
Console.WriteLine(name);
      }
// Using Select() to calculate total prices
var total Price = products.Select(p =>p.Price).Sum();
Console.WriteLine("\nTotal Price: Rs. " + totalPrice +"/-");
// Using SelectMany() to flatten nested collections
      List<List<int>> numbers = new List<List<int>>()
      {
new List<int> { 1, 2, 3 },
```

```
        new List<int> { 4, 5 },
        new List<int> { 6, 7, 8, 9 }
            };
        var flattened Numbers = numbers.SelectMany(list => list);
        Console.WriteLine("\nFlattened Numbers:");
        foreach (var number in flattenedNumbers)
            {
        Console.Write(number + " ");
            }
        }
    }
```

Output :

```
Product Names :
Apple
Banana
Orange
Grapes
Total Price: Rs. 281.00/-
Flattened Numbers:
1 2 3 4 5 6 7 8 9
```

3. **Sorting/Ordering Operators :** LINQ provides sorting or ordering operators that allow you to arrange the elements of a sequence in a specific order (ascending or descending). These operators enable you to sort the elements based on one or more properties or apply custom sorting logic.

S.No.	Sorting Operators	Description
1.	OrderBy()	Sorts the elements of a sequence in ascending order based on a specified key. It returns a new sequence with the sorted elements.
2.	OrderByDescending()	Sorts the elements of a sequence in descending order based on a specified key. It returns a new sequence with the sorted elements.
3.	ThenBy()	Again sort elements in ascending order. Must use after OrderBy or OrderByDescending operator. This operator is useful when we have to apply sorting based on multiple fields. For example, we want to sort first based on name and then age.

4.	ThenByDescending()	Again sort elements in descending order. Must use after OrderBy or OrderByDescending operator. This operator is useful when we have to apply sorting based on multiple fields. For example, we want to sort descending first based on age and then name.

Example Program using Sorting Operators :

```
using System;
using System.Collections.Generic;
using System.Linq;
class Program
{
class Student{
public string Name { get; set; }
public int  Age { get; set; }
public string Grade { get; set; }
  }
static void Main(string[] args)
  {
// Sample list of students
    List<Student> students = new List<Student>()
    {
new Student { Name = "Raju", Age = 21, Grade = "A" },
new Student { Name = "Kiran", Age = 19, Grade = "B" },
new Student { Name = "Suresh", Age = 20, Grade = "B" },
new Student { Name = "Venkatesh", Age = 22, Grade = "A" },
new Student { Name = "Lavanya", Age = 20, Grade = "C" }
    };
// Using OrderBy() to sort students by name in ascending order
var sortedByName = students.OrderBy(student =>student.Name);
Console.WriteLine("Students sorted by Name:");
foreach (var student in sortedByName){
Console.WriteLine($"Name: {student.Name}, Age: {student.Age}, Grade: {student.
Grade}");
  }
```

```
// Using OrderByDescending() to sort students by age in descending order
var sortedByAgeDescending = students.OrderByDescending(student =>student.
Age);
Console.WriteLine("\nStudents sorted by Age (descending):");
foreach (var student in sortedByAgeDescending){
Console.WriteLine($"Name: {student.Name}, Age: {student.Age}, Grade:
{student.Grade}");
    }
// Using ThenBy() to sort students by grade and then by name
var sortedByGradeThenName = students.OrderBy(student =>student. Grade).
ThenBy(student =>student.Name);

Console.WriteLine("\nStudents sorted by Grade and then by Name:");
foreach (var student in sortedByGradeThenName){
Console.WriteLine($"Name: {student.Name}, Age: {student.Age}, Grade:
{student.Grade}");
    }
  }
}
```

Output :

```
Students sorted by Name:
Name: Kiran, Age: 19, Grade: B
Name: Lavanya, Age: 20, Grade: C
Name: Raju, Age: 21, Grade: A
Name: Suresh, Age: 20, Grade: B
Name: Venkatesh, Age: 22, Grade: A
Students sorted by Age (descending):
Name: Venkatesh, Age: 22, Grade: A
Name: Raju, Age: 21, Grade: A
Name: Suresh, Age: 20, Grade: B
Name: Lavanya, Age: 20, Grade: C
Name: Kiran, Age: 19, Grade: B
Students sorted by Grade and then by Name:
```

```
Name: Raju, Age: 21, Grade: A
Name: Venkatesh, Age: 22, Grade: A
Name: Kiran, Age: 19, Grade: B
Name: Suresh, Age: 20, Grade: B
Name: Lavanya, Age: 20, Grade: C
```

4. **Join Operators :** LINQ provides join operators that allow you to combine two or more sequences based on a common key and perform relational operations similar to SQL joins.Join operators offers inner join and left outer joins like functionality.

S.No.	Join Operators	Description
1.	Join()	Combines two sequences based on matching keys and returns a new sequence that contains elements with matching key values from both sequences. It is similar to an inner join in SQL.
2.	GroupJoin()	Combines two sequences based on matching keys and returns a new sequence that groups elements from the first sequence with matching elements from the second sequence. It is similar to a left outer join in SQL.

5. **Set Operators :** LINQ provides set operators that allow you to perform operations on sequences to obtain unique elements or combine sequences based on set operations. These set operators include union, intersect, except, and distinct.

S.No.	Set Operators	Description
1.	**Union()**	Combines two sequences and returns a new sequence that contains unique elements from both sequences. It eliminates duplicate elements and keeps only distinct values.
2.	**Intersect()**	Returns a new sequence that contains common elements present in both sequences. It returns only the elements that are common to both sequences, eliminating any duplicates.
3.	**Except()**	Returns a new sequence that contains elements from the first sequence that do not exist in the second sequence. It removes any elements that are common to both sequences
4.	**Distinct()**	Returns a new sequence that contains distinct elements from a single sequence. It removes any duplicate elements and returns only unique values.

Example Program using Set Operators

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
class Program
{
static void Main(string[] args)
  {
// Sample lists of numbers
      List<int> numbers1 = new List<int>{ 1, 2, 3, 4, 5 };
      List<int> numbers2 = new List<int>{ 4, 5, 6, 7, 8 };
// Union: combines both lists and returns distinct elements
var union = numbers1.Union(numbers2);
Console.WriteLine("Union:");
foreach (var num in union){
Console.Write(num + " ");
    }
// Intersect: finds common elements in both lists
var intersect = numbers1.Intersect(numbers2);
Console.WriteLine("\nIntersect:");
foreach (var num in intersect){
Console.Write(num + " ");
    }
// Except: returns elements in the first list that are not in the second list
var except = numbers1.Except(numbers2);
Console.WriteLine("\nExcept:");
foreach (var num in except){
Console.Write(num + " ");
    }
// Distinct: returns distinct elements from a single list
List<int> numbers = new List<int>{ 1, 2, 3, 2, 4, 5, 1 };
var distinct = numbers.Distinct();
Console.WriteLine("\nDistinct:");
```

```
        foreach (var num in distinct){
        Console.Write(num + " ");
            }
        }
    }
```

Output :

```
Union:
1 2 3 4 5 6 7 8
Intersect:
4 5
Except:
1 2 3
Distinct:
1 2 3 4 5
```

6. **Aggregation Operators :** LINQ provides aggregation operators that allow you to perform calculations on sequences and obtain a single result. These operators allow you to perform common aggregation operations such as summing, averaging, counting, finding the maximum or minimum value, and more.

S.No.	Aggregation Operators	Description
1.	Count()	Returns the number of elements in a sequence. It can be used with or without a condition.
2.	Sum()	Calculates the sum of all elements in a numeric sequence. It is commonly used with numeric types such as integers or doubles.
3.	Average()	Calculates the average value of a numeric sequence. It computes the sum of all elements and divides it by the total count.
4.	Min()	Returns the minimum value in a sequence.
5.	Max()	Returns the maximum value in a sequence.
6.	Aggregate()	Performs a custom aggregation operation on a sequence. It takes a lambda function that defines the custom aggregation logic.

Example Program using Aggregation Operators :

```
using System;
using System.Linq;
class Program
```

```
{
static void Main()
{
int[] numbers = { 1, 2, 3, 4, 5 };
// Count: returns the number of elements in the sequence
int count = numbers.Count();
Console.WriteLine($"Count: {count}");
// Sum: calculates the sum of all elements in the sequence
int sum = numbers.Sum();
Console.WriteLine($"Sum: {sum}");
// Average: calculates the average value of the elements in the sequence
double average = numbers.Average();
Console.WriteLine($"Average: {average}");
// Min: finds the minimum value in the sequence
int min = numbers.Min();
Console.WriteLine($"Min: {min}");
// Max: finds the maximum value in the sequence
int max = numbers.Max();
Console.WriteLine($"Max: {max}");
// Aggregate: performs a custom aggregation operation on the sequence
int product = numbers.Aggregate((acc, num) =>acc * num);
Console.WriteLine($"Product: {product}");
}
}
```

Output :

```
Count: 5
Sum: 15
Average: 3
Min: 1
Max: 5
Product: 120
```


7. **Partitionin**

a specific s
These ope
and take a

S.No.	Part
1.	Take
2.	Skip
3.	Take
4.	Skip
5.	TakeL
6.	SkipL

Example :

```
int[]
var
int[]
var
int[]
var
int[]
var
int[]
var
int[]
var
```

8. **Conversi**

or transfo
you to co

7. **Partitioning Operators :** LINQ provides partitioning operators that allow you to retrieve a specific subset of elements from a sequence based on certain criteria or conditions. These operators enable you to divide a sequence into smaller portions, skip elements, and take a specific number of elements.

S.No.	Partitioning Operators	Description
1.	Take()	Returns a specified number of elements from the beginning of a sequence. It returns a new sequence containing the first N elements.
2.	Skip()	Skips a specified number of elements from the beginning of a sequence and returns the remaining elements. It allows you to exclude the first N elements.
3.	TakeWhile()	Returns elements from the beginning of a sequence as long as a specified condition is true. It returns elements until the condition becomes false.
4.	SkipWhile()	Skips elements from the beginning of a sequence as long as a specified condition is true and returns the remaining elements. It skips elements until the condition becomes false.
5.	TakeLast()	Returns a specified number of elements from the end of a sequence. It returns the last N elements.
6.	SkipLast()	Skips a specified number of elements from the end of a sequence and returns the remaining elements. It excludes the last N elements.

Example :

```
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.Take(3); // returns a sequence: 1, 2, 3
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.Skip(2); // returns a sequence: 3, 4, 5
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.TakeWhile(num =>num< 4); // returns a sequence: 1, 2, 3
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.SkipWhile(num =>num< 4); // returns a sequence: 4, 5
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.TakeLast(2); // returns a sequence: 4, 5
int[] numbers = { 1, 2, 3, 4, 5 };
var result = numbers.SkipLast(2); // returns a sequence: 1, 2, 3
```

8. **Conversion Operators :** LINQ provides conversion operators that allow you to convert or transform data from one type or representation to another. These operators enable you to convert sequences to arrays, lists, dictionaries, and other collection types.

S.No.	Conversion Operators	Description
1.	ToList()	Converts a sequence into a List<T>, where T is the type of the elements in the sequence. It creates a new List<T> containing the elements from the source sequence.
2.	ToArray()	Converts a sequence into an array of the specified type. It creates a new array containing the elements from the source sequence.
3.	ToDictionary()	Converts a sequence to a dictionary, where each element of the sequence is transformed into a key-value pair. It creates a new dictionary where the keys and values are extracted from the elements of the source sequence.
4.	ToLookup()	Converts a sequence to a lookup (a specialized dictionary for grouping). It creates a new lookup table where elements are grouped by the key extracted from the elements of the source sequence.
5.	Cast()	Converts a sequence of objects to a specified type. It performs a type cast operation on each element of the source sequence.
6.	AsEnumerable()	It can be used to explicitly type a sequence as IEnumerable<T>.

Example :

```
//Converting array to List Example
int[] numbers = { 1, 2, 3, 4, 5 };
List<int>numberList = numbers.ToList();
//Converting List to array Example
List<int>numberList = new List<int>{ 1, 2, 3, 4, 5 };
int[] numberArray = numberList.ToArray();
// Converting array to dictionary Example
var students = new List<Student>{
new Student { Id = 1, Name = "Alice" },
new Student { Id = 2, Name = "Bob" },
new Student { Id = 3, Name = "Charlie" }
};
Dictionary<int, string>studentDictionary = students.ToDictionary(student =>
student.Id, student =>student.Name);
// Cast operator Example
object[] mixedTypes = { 1, "two", 3.0, "four", 5 };
var numbers = mixedTypes.Cast<int>();
```

9. **Quantifier Operators :** LINQ provides quantifier operators that allow you to check if certain conditions are true for any or all elements in a sequence. These operators evaluate the elements of a sequence and return a Boolean value indicating the result.

S.No.	Quantifier Operators	Description
1.	All()	Checks if all elements in a sequence satisfy a specified condition. It returns true if the condition is true for all elements, otherwise false.
2.	Any()	Checks if there is at least one element in a sequence that satisfies a specified condition. It returns true if at least one element satisfies the condition, otherwise false.
3.	Contains()	Checks if a sequence contains a specified element. It returns true if the element is found, otherwise false.

Example Program using Quantifier Operators :

```
using System;
using System.Linq;
class Program
{
static void Main(string[] args)
   {
int[] numbers = { 1, 3, -6, 7 };
// All: Check if all elements are positive
bool allPositive = numbers.All(num =>num> 0);
Console.WriteLine($"All elements are positive: {allPositive}");
// Any: Check if any element is even
bool anyEven = numbers.Any(num =>num % 2 == 0);
Console.WriteLine($"Any element is even: {anyEven}");
// Contains: Check if the sequence contains a specific number
bool containsThree = numbers.Contains(3);
Console.WriteLine($"Contains 3: {containsThree}");
bool containsTen = numbers.Contains(10);
Console.WriteLine($"Contains 10: {containsTen}");
   }
}
```

Output :

```
All elements are positive: False
Any element is even: True
Contains 3: True
Contains 10: False
```

10. **Grouping Operators :** LINQ provides grouping operators that allow you to group elements in a sequence based on a common key or propertyjust like groups in SQL. Grouping is useful when you want to organize and analyze data based on specific criteria. LINQ provides several grouping operators to perform grouping operations on sequences.

S.No.	Grouping Operators	Description
1.	**GroupBy()**	Group elements in a sequence based on a specified key. It returns a collection of groups where each group contains a key and all the elements that share that key.
2.	**ToLookup()**	It is similar to GroupBy and is used to group elements in a sequence based on a key. It returns a lookup table that allows direct access to the grouped elements by key.
3.	**GroupJoin()**	Performs a group join between two sequences based on a common key.

Example :

```
var numbers = new List<int> { 1, 2, 3, 4, 5, 6 };
var groups = numbers.GroupBy(num =>num % 2 == 0 ? "Even" :"Odd");
foreach (var group in groups){
Console.WriteLine($"Key: {group.Key}");
foreach (var number in group){
Console.WriteLine(number);
  }
Console.WriteLine();
}
```

11. **Sequencing Operators :** LINQ provides sequencing operators that allow you to generate sequence of elements based on specific patterns or conditions. These operators enable you to create sequences dynamically, iterate over them, or combine them with other LINQ operators to perform various data manipulation tasks.

S.No.	Sequencing
1.	Range()
2.	Repeat()
3.	Empty()

Example :

```
var numbers
foreach (var
{
Console.Wri
}
var repeate
foreach (va
{
Console.W
}
var empty
Console.W
```

12. Equality Operat
or elements with
equality checks

S.No.	Equality
1.	Equals()
2.	Sequence

S.No.	Sequencing Operators	Description
1.	Range()	Generates a sequence of consecutive integers within a specified range. It takes two parameters: the starting value and the number of elements to generate. It creates a sequence of integers starting from the specified value and incrementing by one for the specified number of elements.
2.	Repeat()	Generates a sequence that contains a specified element repeated a specified number of times. It takes two parameters: the element to repeat and the number of times to repeat it.
3.	Empty()	Generates an empty sequence of a specified type. It creates a sequence that contains no elements. This operator is useful when you need to represent an empty collection without allocating memory or storing any elements.

Example :

```
var numbers = Enumerable.Range(1, 5);
foreach (var number in numbers)
{
Console.Write(number + " "); //1 2 3 4 5
}
var repeatedValues = Enumerable.Repeat("Hello", 3);
foreach (var value in repeatedValues)
{
Console.WriteLine(value); // Hello HelloHello
}
var emptySequence = Enumerable.Empty<int>();
Console.WriteLine(emptySequence.Any()); // Outputs: False
```

12. **Equality Operators :** LINQ provides equality operators that allow you to compare sequences or elements within sequences based on their equality. These operators help you perform equality checks and determine whether sequences or elements are equal or not.

S.No.	Equality Operators	Description
1.	**Equals()**	Checks for equality between two objects or values. It returns true if the two objects are equal based on their overridden Equals() method or equality comparer; otherwise, it returns false.
2.	**SequenceEqual()**	Compares two sequences to check if they have the same elements in the same order. It returns true if the sequences are equal; otherwise, it returns false.

Example :

```
string name1 = "Suresh";
string name2 = "Suresh";
bool areEqual = name1.Equals(name2); // returns true
List<int> numbers1 = new List<int>{ 1, 2, 3 };
List<int> numbers2 = new List<int>{ 1, 2, 3 };
bool areEqualLists = numbers1.Equals(numbers2); // returns false (reference comparison)
int[] numbers1 = { 1, 2, 3 };
int[] numbers2 = { 1, 2, 3 };
bool areEqualSequences = numbers1.SequenceEqual(numbers2); // returns true
List<string> names1 = GetNames();
List<string> names2 = GetOtherNames();
bool areEqualNames = names1.SequenceEqual(names2); // returns true if the names are in the same order and have the same elements
```

13. **Element Operators :** LINQ provides element operators that allow you to retrieve specific elements or information from a sequence. These operators enable you to access individual elements, find elements based on certain conditions, or retrieve specific information about the elements in a sequence.

S.No.	Element Operators	Description
1.	**First()**	Returns the first element in a sequence that satisfies a specified condition. If no such element is found, it throws an exception.
2.	**FirstOrDefault()**	Returns the first element in a sequence that satisfies a specified condition, or a default value if no such element is found. It does not throw an exception.
3.	**Last()**	Returns the last element in a sequence that satisfies a specified condition. If no element satisfies the condition, it throws an exception.
4.	**LastOrDefault()**	Returns the last element in a sequence that satisfies a specified condition. If no element satisfies the condition, it returns the default value for the element type.
5.	**Single()**	Returns the only element in a sequence that satisfies a specified condition. If no such element is found or multiple elements match the condition, it throws an exception.

6.	Sing
7.	Elen
8.	Elen

14. Concatena
multiple se
or merge s
from the so

S.No.	
1.	Conca
2.	Union
3.	Zip()

Example :

```
int[]
int[]
var
int[]
int[]
var
int[]
int[]
var
```

6.	**SingleOrDefault()**	Returns the only element that satisfies a condition, or a default value if no such element is found.
7.	**ElementAt()**	Returns the element at a specified index in a sequence. It throws an exception if the index is out of range.
8.	**ElementAtOrDefault()**	Returns the element at a specified index, or a default value if the index is out of range.

14. **Concatenation Operators :** In LINQ, concatenation operators allow you to combine multiple sequences into a single sequence. These operators enable you to concatenate or merge sequences together to create a new sequence containing all the elements from the source sequences.

S.No.	Concatenation Operators	Description
1.	**Concat()**	Combines two sequences by appending the elements of the second sequence to the end of the first sequence. It returns a new sequence that contains all the elements from both sequences.
2.	**Union()**	Combines two sequences by removing duplicate elements and returning a new sequence that contains distinct elements from both sequences. It considers the default equality comparer for the element type unless a custom equality comparer is provided.
3.	**Zip()**	Combines two sequences by pairing corresponding elements from each sequence into a new sequence of tuples. It takes two input sequences and applies a specified function to each pair of elements to create the result.

Example :

```
int[] sequence1 = { 1, 2, 3 };
int[] sequence2 = { 4, 5, 6 };
var concatenated = sequence1.Concat(sequence2); //1,2,3,4,5,6
int[] sequence1 = { 1, 2, 3 };
int[] sequence2 = { 3, 4, 5 };
var union = sequence1.Union(sequence2); //1,2,3,4,5
int[] sequence1 = { 1, 2, 3 };
int[] sequence2 = { 4, 5, 6 };
var zipped = sequence1.Zip(sequence2, (a, b) => (a, b)); //(1, 4), (2, 5), (3, 6)
```

REVIEW QUESTIONS

One Mark Questions :

1. What is ADO.NET?
2. List the advantages of ADO.NET.
3. What is Connection class?
4. What is Command object?
5. What is DataSet?
6. Define DataTable.
7. Define DataRow and DataColumn.
8. Define DataRelation.
9. Define simple data binding.
10. Define complex data binding.
11. What is Binding context?
12. Define LINQ.
13. Write the syntax to create LINQ.

Three Mark Questions :

14. Write about .NET data providers.
15. What are the core components of .NET data providers?
16. Write the features of ADO.NET.
17. Write about Dataset and DataAdapter classes.
18. Write the types of DataSets?
19. Write about DataReader object.
20. Compare typed and untyped dataset objects.
21. Write the core assemblies in LINQ.
22. Write about LINQ query syntax.
23. Write about LINQ method syntax.
24. Write the types LINQ objects.
25. Write the advantages of LINQ.

26. Write about LINQ filtering operators.
27. Write about LINQ sorting operators.
28. Write about LINQ set operators.
29. Write about LINQ quantifier operators.
30. Write about LINQ projection operators.
31 Write about LINQ partitioning operators.
32. Write about LINQ join operators.
33. Write about LINQ grouping operators.
34. Write about LINQ sequencing operators.
35. Write about LINQ element operators.
36. Write about LINQ conversion operators.
37. Write about LINQ aggregation operators.

Five Mark Questions :

38. Explain ADO.NET architecture with a neat sketch.
39. Explain the steps to access data with DataAdapters and DataSets.
40. Explain data binding with DataGrid control.
41. Explain data binding with textbox and listbox control.
42. Explain Navigate through a data source with an example program.
43. Write a C# program to explain LINQ query syntax.
44. Write a C# program to explain LINQ method syntax.
45. Explain LINQ to objects with an example program.
46. Explain LINQ to XML with an example program.
47. Explain LINQ to SQL with an example program.
48. Explain various LINQ operators.
43. Write programs using any two LINQ operators.