Royal Education Society's

College of Computer Science and Information Technology, Latur.

Department of Computer Science

Academic Year (2022-23)

Choice Based Credit System (CBCS Revised)

Class/Semester: **B.Sc.(CS) TY SEM-V**

Name of Paper: **Windows Programming(BCS-501)**

Prepared by: **Ms. Tayyaba Shaikh**

Question Paper

Q.1 Attempt any FIVE of the following (3 Marks each) 15

a) Explain properties of Windows form.

b) Explain intellisence in visual studio.

c) Explain important classes used in Windows.

d) Define Event.What are mouse events in C#.Net?

e) Explain out parameter in detail.

f) Explain try, catch and finally block.

g)Explain project types in C#.Net.

Q. 2 Attempt any three of the following (5 Marks each) 15

a) Explain difference between Java and C#.

b) Explain .Net Architecture in detail.

c) Explain Jagged Array in detail.

d) Explain textbox and button control with properties.

e) Write a program to calculate factorial of a number using windows form.

Q. 3 Attempt any three of the following (5 Marks each) 15

a) Write a Program for demonstration of handling exception.

b) Explain ListBox control in detail.

c) Explain C# function.

d) Explain concept of delegates.

e) Explain Dialog Boxes in C#.

Q. 4 Attempt any three of the following (5 Marks each) 15

a) Explain Arraylist class with example.

b) Explain creating and using interfaces.

c) Explain connected data access in C#.

d) Explain CLR architecture.

e) WAP for demonstration of accessing data from database.

Q .5 Short notes on any three of the following (5 Marks each) 15

a) IDE and .Net

b) ADO.NET Architecture

c) String Class

d) Namespace (DLL Library)

e) Properties in C#

Modal Answer Paper

Q.1 Attempt any FIVE of the following (3 Marks each) 15

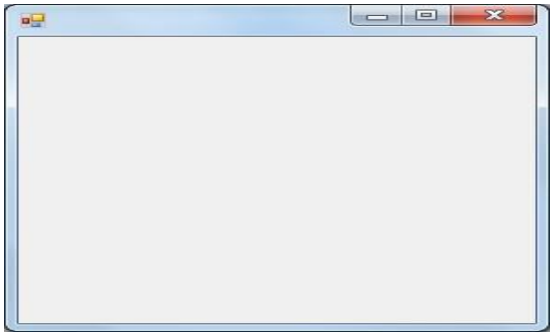
a) Explain properties of Windows form.

Form is the very first entity typically included in a Windows-based application. It hosts a number of other controls for performing desired functions.

In C#. a form can be created by inheriting the Form class contained in the System.Windows.Forms namespace. We can customize form's look by making use of the various properties of the Form class.

Following code shows to create a simple blank form,

```
Form f1 = new Form( );
F1.Show( );
```



Properties of Windows Form:

1. **ControlBox :** Enables or disables the control box.
2. **MaximizeBox:** Enables or disables the Maximize button.
3. **MinimizeBox:** Enables or disables the Minimize button.
4. **Text:** Helps add a caption for the form
5. **DefaultSize:** Sets the default size of a form.
6. **Height:** Sets the height of a form.
7. **Width:** Sets the width of a form.
8. **MaximumSize:** Sets the maximum size of a form.
9. **MinimumSize:** Sets the minimum size of a form.
10. **StartPosition:** specifies the initial position of a form.
11. **BackColor:** It enables us to modify the background color of the form

Example-

```
Form f1 = new Form( );
f1.ControlBox = true;
f1.MaximizeBox = false;
```

```
f1.MinimizeBox = true;
f1.Text = "My Form";
f1.Height = 250;
f1.Width = 250;
f1.Text =”My Form”;
f1.BackColor = Color.Red;
f1.Show( );
```

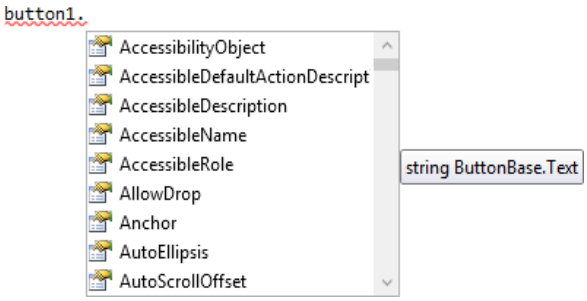
b) Explain intellisence in visual studio.

IntelliSense is a code completion tool that is built into Microsoft Visual Studio. Auto completing is one way to offer a productive and programmer friendly user interface to programmers and IntelliSense feature of Visual Studio .NET is one of them. It provides following features,

1) List Members

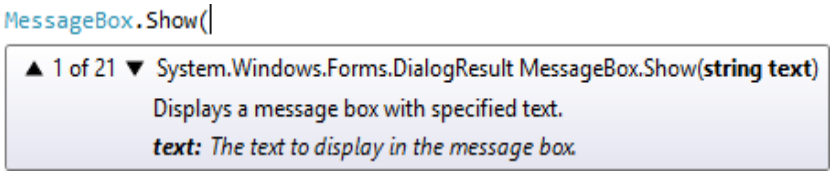
A list of valid members from a type appears after you type a period character (.).

In the member list, the icon to the left represents the type of the member, such as namespace, class, function, or variable. You can invoke the List Members feature manually by typing Ctrl + J, choosing Edit > IntelliSense > List Members, or by choosing the List Members button on the editor toolbar.



2) Parameter Info

Parameter Info gives you information about the number, names, and types of parameters required by a method. The parameter in bold indicates the next parameter that is required as you type the function.

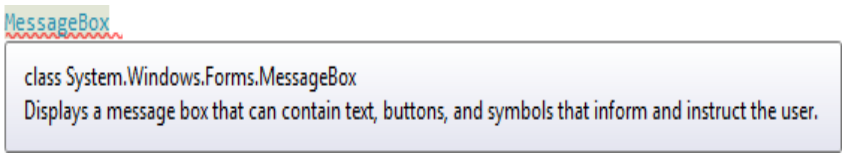


We can manually invoke Parameter Info by choosing Edit > IntelliSense > Parameter Info, by pressing Ctrl + Shift + Space, or by choosing the Parameter Info button on the editor toolbar.

3) Quick Info

Quick Info displays the complete declaration for any identifier in your code. When youselect a member from the List Members box, Quick Info also appears.

We can manually invoke Quick Info by choosing Edit > IntelliSense > Quick Info, bypressing Ctrl + I, or by choosing the Quick Info button on the editor toolbar.



4) Complete Word

Complete Word completes the rest of a variable, command, or function name after you have entered enough characters to disambiguate the term. We can invoke Complete Word by choosing Edit > IntelliSense > Complete Word, by pressing Ctrl + Space, or by choosing the Complete Word button on the editor toolbar.

c) Explain important classes used in Windows.

The Form class is the fundamental class for developing Windows application. The base classes concerned with the Form class are defined in the System namespace.

Following are the base classes form an object class,

1. Component Class

Every class in C#.Net inherits from Object class. This class defines a number of methods that allow a control, such as button to interact with the hosting container, such as s Form.

This class also provides an implementation of Dispose( ) method, which is called by the system when a component is no longer required .

2. Control Class

Control Class encapsulates the common behavior exposed by every control, such as Button control and ListBox control. The core properties of this class help us to configure thesize and position of a control. The core methods of this class help us to capture and handle the events.

3. ScrollableControl Class

The ScrollableControl class has only few members, which allows the container to support vertical and horizontal scrollbars. The two important properties in this class are AutoScroll and AutoScrollMinSize.

4. ContainerControl Class

The members of the ContainerControl class are useful when a Form object contains a number of child controls and we wish to allow the user to alternate focus among these controls with the help of the Tab key. Some members of the ContainerControl class are ActiveControl, ParentForm, ProcessTabKey( ), etc.

5. Form Class

The main class of every Windows application inherits from the Form class. In addition to the list of properties and methods that the Form class inherits from its base classes, such as the Control class, ScrollableControl, and ContainerControl class it possesses some additional properties (like BorderStyle, CancelButton, etc.), methods (like Activate( ), Close( ), etc.), and events (like Closing, Closed, etc.).

d) Define Event. What are mouse events in C#.Net?

Events are user actions such as key press, clicks, mouse movements etc. There are several kinds of mouse events:

1. **Click**
   This Event raised when the control is clicked.The handler for this event receives an argument of type EventArgs. Handle this event when you only need to determine when a click occurs.

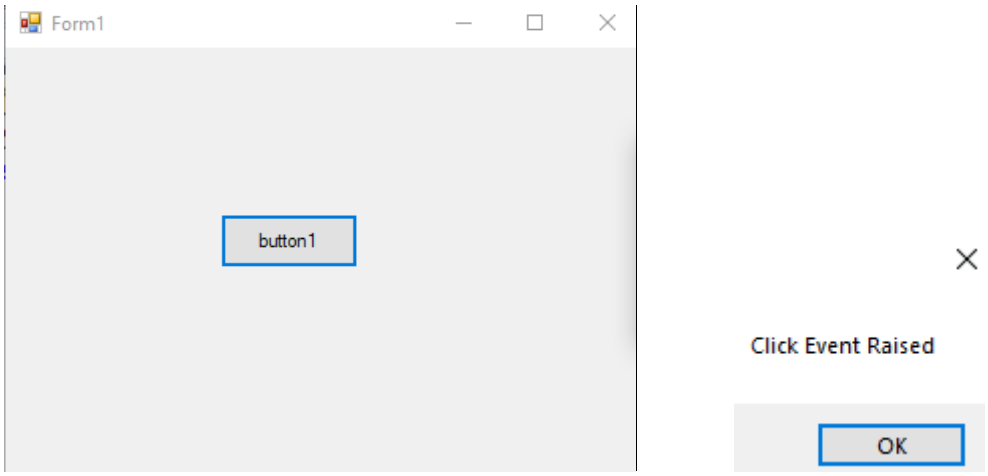
2. **DoubleClick**
   This event occurs when the control is double-clicked. The handler for this event receives an argument of type EventArgs. Handle this event when you only need to determine when a double-click occurs.

3. **MouseDown**
   This event occurs when the mouse pointer is over the control and the user presses a mouse button. The handler for this event receives an argument of type MouseEventArgs.

4. **MouseEnter**
   Event raised when the mouse enters the visible part of the control.The handler for this event receives an argument of type EventArgs.

5. **MouseLeave**
   Event raised when the mouse leaves the visible part of the control. The handler for this event receives an argument of type EventArgs.

Example to demonstrate Mouse Click Event.



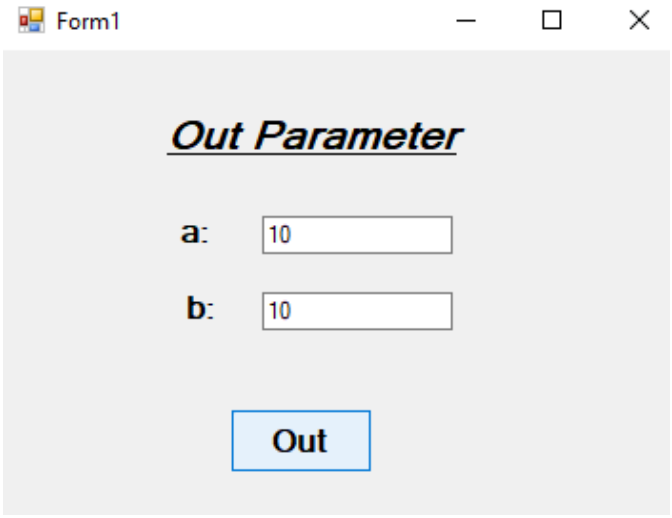

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         MessageBox.Show("Click Event Raised");
      }
   }
}
```

e) Explain out parameter in detail.

Function is a block of code that perform specified task. Function is used to execute statements specified in the code block. To use a method, you need to Define the method andCall the method.

C# provides out keyword to pass arguments as out-type. It is like reference-type, except that it does not require variable to initialize before passing. We must use out keyword to pass argument as out-type. It is useful when we want a function to return multiple values.

Example




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

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void btn_out_Click(object sender, EventArgs e)
      {
         int a;
         disp_out(out a);
         txt_a.Text = a.ToString();
      }
```

```
    public void disp_out(out int b)
    {
       b = 10;
       txt_b.Text = b.ToString();
    }
  }
}
```

f) Explain try, catch and finally block.

a) try block

A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

b) catch block

Exception raised within try block can be handled using the catch block. Code in the catch block will only execute when an exception occurs.

c) Finally Block

The finally block must come after a try or catch block. The finally block will always be executed whether or not an exception is thrown. The finally block is generally used for cleaning-up code e.g. for disposing an unmanaged objects etc.

Syntax –

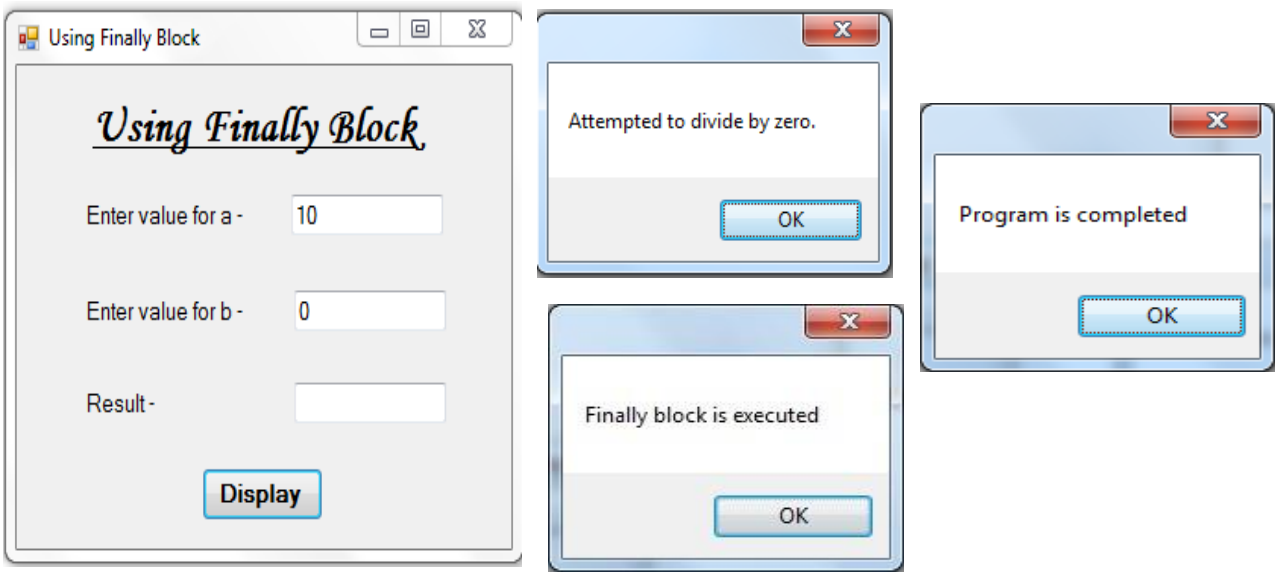
```
try
{
        //Code that may raise exceptions
}
Catch
{
        // handle exception
}
finally
{
        // statements to be executed at any cost
}
```

Example-

```
namespace finallyDemo
{
        public partial class Form1 : Form
        {
                public Form1( )
```

```
        {
            InitializeComponent( );
        }
        private void btn_display_Click(object sender, EventArgs e)
        {
            try
            {
                int a = int.Parse(txt_a.Text);
                int b = int.Parse(txt_b.Text);
                int c = a / b;
                txt_result.Text = c.ToString( );
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                MessageBox.Show("Finally block is executed");
            }
            MessageBox.Show("Program is completed");
        }
    }
}
```

Output:



g) Explain project types in C#.Net.

The Following are some of the different projects that can be created with C#.NET,

1. Console applications
2. Windows Applications
3. Web applications
4. Web services
5. Class library
6. Windows Control Library
7. Web Control Library

1) Console Application

A console application is an application that runs in a console window same as a C and C++ program. It doesn't have any graphical user interface. Console Applications will have character based interface.

2) Windows Application

It is a form based standard desktop application for common day to day tasks such as inventory management system, college management system and many more.

3) Web Application

It is a client–server computer program which the client runs in a web browser. Common web applications include webmail, online retail sales, online auctions, and many more.

4) Web services

Web services are web applications that provide services to other applications over the internet. Web service is a language independent way of communication.

5) Class Library

Class Library contains classes, interfaces, and value types to be used inside other applications. A Class library cannot be executed and thus it does not have any entry point.

6) Windows Control Library

Windows Control Library contains user defined windows controls to be used by Windows applications.

7) Web Control Library

Web Control Library contains user defined web controls to be used by web applications.

Q. 2 Attempt any three of the following (5 Marks each) 15

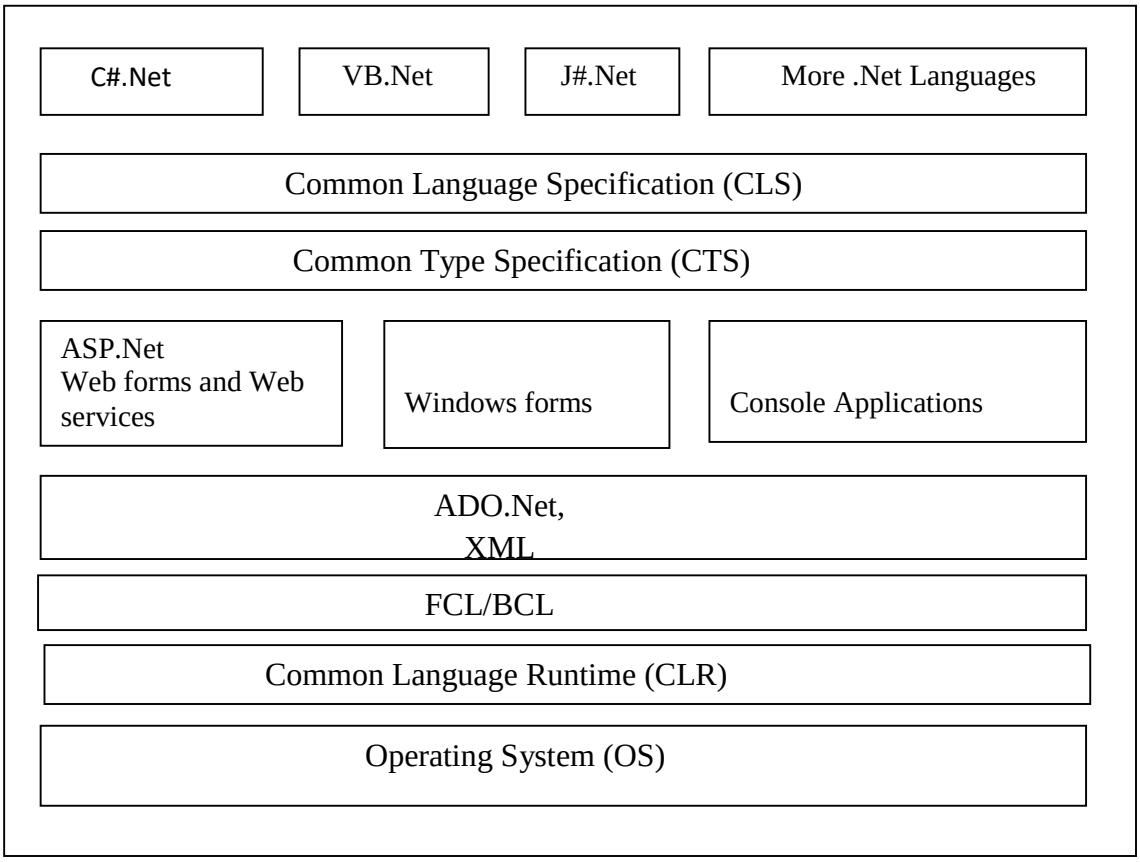
a) Explain difference between Java and C#.

Java	C# (Sharp)
1) Java is an object-oriented programming language developed by James Gosling for Sun-Microsystem in 1995.	1) C# is an object-oriented programming language developed by Anders Hejlsberg for Microsoft around 2000.
2) Java does not support concept of operator overloading.	2) C# supports concept of operator overloading.
3) Java does not support concept of delegates.	3) C# supports concept of delegates.
4) Java uses JVM (Java Virtual Machine).	4) C# uses CLR (Common Language Runtime).
5) Java supports only signed integer type.	5) C# supports both unsigned and signed integer types.
6) Java does not support concept of Cross - language interoperability.	6) C# supports concept of Cross - language interoperability.
7) Java has no preprocessor directives.	7) C# supports few preprocessor directives.
8) In Java, the switch statement can have only integer expression.	8) In C#, the switch statement can have both integer and string expression.
9) Java provides less data types as compared to C#.	9) C# provides more data types as compared to Java.
10) Java doesn't have enumerations, but can specify a class to emulate them.	10) C# supports enumerations.
11) Java does not support the strut type.	11) C# supports the strut type.
12) Java allows parameters to be passed by value.	12) C# allows parameters to be passed by reference by using the ref keyword.
13) Java does not support goto statement.	13) C# supports goto statement.
14) Java does not support events and delegates.	14) C# supports events and delegates.
15) Java has no language support for a decimal type.	15) C# has language support for a decimal type.

b) Explain .Net Architecture in detail.

"Dot Net (.Net) is not a programming language, it is a platform introduce by Microsoft todevelop various kinds of applications."

Following model shows .Net architecture,



1) .Net Languages

.Net technology supports different types of languages such as C#, VB, etc. to develop various types of applications.

CTS and CLS are parts of .NET CLR and are responsible for type safety within the code.

2) CLS

CLS stands for Common Language Specification and it is a subset of CTS. It defines aset of rules and restrictions that every language must follow which runs under .NET framework.

3) CTS

It stands for Common Type Specification. CTS defines how data types are declared, used and managed in the runtime. It facilitates cross-language integration. The rules defined in CTS can be used to define your own classes and values.

4) a) ASP.NET

ASP.NET is used to create dynamic Web applications and is the successor to ASP.

1) Web Forms are the forms generator for Web applications in ASP.NET andreplaces Visual Interdev.
2) Web services are part of ASP.NET and beneficial for open standards, such asHTTP and XML, to publish public functions to the Internet.

b) Windows Forms

Windows Forms is the form generator for client-side applications and it is similar to the visual basic forms. Here, we perform GUI based activities.

c) Console Based Application

Console applications have been available to C base programmers and useful for text based activities.

5) a) ADO.NET

It provides support for data access in Microsoft .NET. It stands for ActiveX Data Object.

b) XML

It stands for Extensible Markup Language. XML is the temporary data storage of the database used for web applications.

6) Class library

The .NET Framework includes a set of standard class libraries. These class libraries implement a large number of common functions, such as file reading and writing, graphic rendering, database interaction, and etc.

The class library is organized in a hierarchy of namespaces. Most of the built in classes are part of either System.* or Microsoft.* namespaces.

The class library is divided into two parts: the Framework Class Library and the Base Class Library.

a) Framework Class Library (FCL)

It is a superset of the BCL classes and refers to the entire class library. It includes an expanded set of libraries, including the Windows Forms, ASP.NET, and Windows Presentation Foundation (WPF), Language Integrated Query (LINQ), Windows Communication Foundation (WCF), and Workflow Foundation(WF).

b) Base Class Library (BCL)

It includes a small subset of the entire class library.

For .NET Framework most classes considered being part of BCL reside in mscorlib.dll, System.dll and System.Core.dll

7) Common Language Runtime (CLR)

It is the heart of the .NET framework. It is the responsibility of the CLR to take care of the code execution of the program. The CLR is a runtime engine that handles memory allocation, security, code verification, type verification, exception handling, and garbage collection.

8) Operating System

It is a system software, which acts as an interface between computer user and computer hardware. Operating system supports to CLR to execute .Net applications.

c) Explain Jagged Array in detail.

A jagged array is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an "array of arrays."

Syntax –

```
Data_Type[ ][ ] Array_Name = new Data_Type[size][dimension];
```

A jagged array is initialized with two square brackets [ ][ ]. The first bracket specifies the size of an array and the second bracket specifies the dimension of the array which is going to be stored as values. (Remember, jagged array always store an array.

Example –

The following is a declaration of a single-dimensional array that has two elements, each of which is a single-dimensional array of integers:

```
int[ ][ ] jaggedArray = new int[2][ ];
```

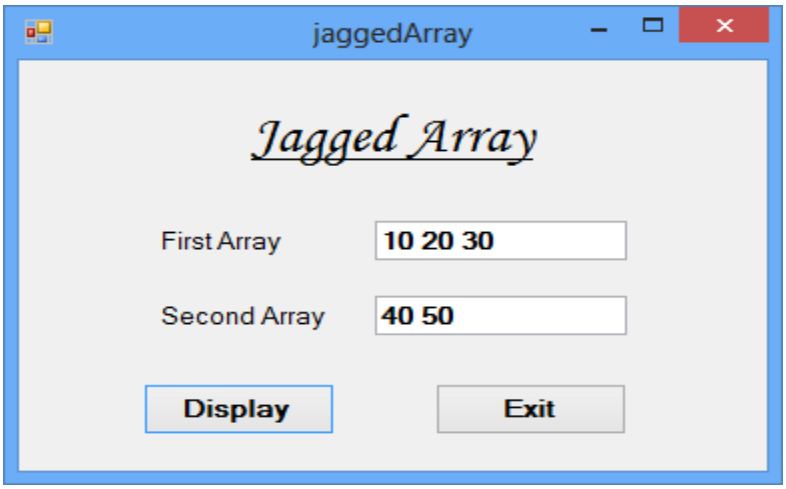
Before you can use jaggedArray, its elements must be initialized. You can initialize the elements like this:

```
jaggedArray[0] = new int[3];
jaggedArray[1] = new int[2];
```

You can assign individual array elements like following examples:

```
jaggedArray[0][0]  = 10;    // Assign 10 to the first element ([0]) of the first array ([0])
jaggedArray[0][1]  = 20;    // Assign 20 to the second element ([1]) of the first array ([0])
jaggedArray[0][2]  = 30;    // Assign 30 to the third element ([2]) of the first array ([0]):
jaggedArray[1][0] = 40;    // Assign 40 to the first element ([0]) of the second array ([1]):
jaggedArray[1][1] =50; // Assign 50 to the second element ([1]) of the second array ([1]):
```

Example– Program to demonstrate the concept of Jagged Array in C#.NET.




```
namespace JaggedArrayExample
{
   public partial class Form1 : Form
   {
      public Form1( )
      {
         InitializeComponent( );
      }
      private void btn_display_Click(object sender, EventArgs e)
      {
         // Declare the array of two elements:
             int[][] jaggedArray = new int[2][];

         // Initialize the elements:
             jaggedArray[0] = new int[3];
             jaggedArray[1] = new int[2];

             jaggedArray[0][0] = 10;
             jaggedArray[0][1] = 20;
             jaggedArray[0][2] = 30;

             jaggedArray[1][0] = 40;
             jaggedArray[1][1] = 50;
```

```
        // Display the array elements:
        for (int i = 0; i < jaggedArray.Length; i++)
        {
            for (int j = 0; j < jaggedArray[i].Length; j++)
            {
                if (i == 0)
                {
                    txt_first.Text = txt_first.Text + jaggedArray[i][j] + " ";
                }
                else if(i == 1)
                {
                    txt_second.Text = txt_second.Text + jaggedArray[i][j] + " ";
                }
            }
        }
    }
    private void btn_exit_Click(object sender, EventArgs e)
    {
        Application.Exit( );
    }
  }
}
```

d) Explain textbox and button control with properties.

a) TextBox Control

A TextBox control accepts user input on a Form. We can create a TextBox control using a Forms designer at design-time. To create a TextBox control at design-time, you simply drag anddrop a TextBox control from Toolbox to a Form in Visual Studio.

a) Button Control

A Button control is a child control placed on a Form and used to process click event and can be clicked by a mouse click or by pressing ENTER or ESC keys.

We can create a Button control using a Forms designer at design-time. To create a Button control at design-time, we simply need to drag and drop a Button control from Toolbox to a Form in Visual Studio.

Common Properties of TextBox and Button Controls–

1) **Name –** It represents a unique name of a control. It is used to access the control in the code.
2) **BackColor –** It is used to set background color of a control.
3) **ForeColor –**This property is used to set foreground color of a control.
4) **Size –** The Size property specifies the size of the control in pixels.
5) **Text –** Text property of a control represents the current text of a control.
6) **TextAlign –** It represents text alignment that can be Left, Center, or Right.
7) **Visible –** Determines whether the control is visible or hidden.

Unique Properties of TextBox Control

1) **CharacterCasing –** It sets the case of text in a TextBox,It has three values Upper, Lower,and Normal.

2) **MaxLength –** We can restrict number of characters in a TextBox control by settingMaxLength property.

3) **Multiline –** By default, a TextBox control accepts input in a single line only. To make it multi-line, we need to set Multiline property to true.

4) **PasswordChar –** It is used to apply masking on a TextBox when you need to use it for a password input. For example, you can place a star (*) for passwordcharacters.

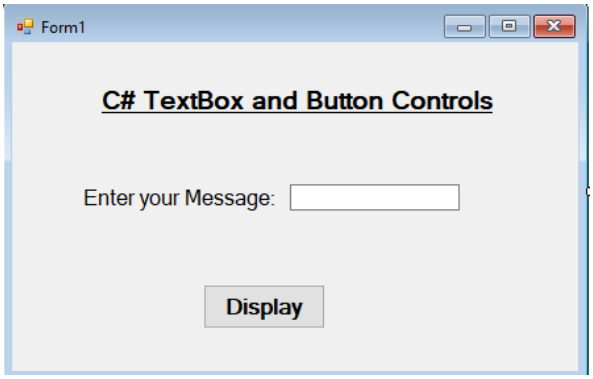
5) **ScrollBars –** A Multiline TextBox control can have scrollbars. The ScrollBars property of TextBox control is used to show scrollbars on a control. The ScrollBars property is represented by a ScrollBars enumeration that has four values Both, Vertical, Horizontal, and None.

6) **ReadOnly –** We can make a TextBox control read-only (non-editable) by setting theReadOnlyproperty to true.

Unique Properties of Button Control

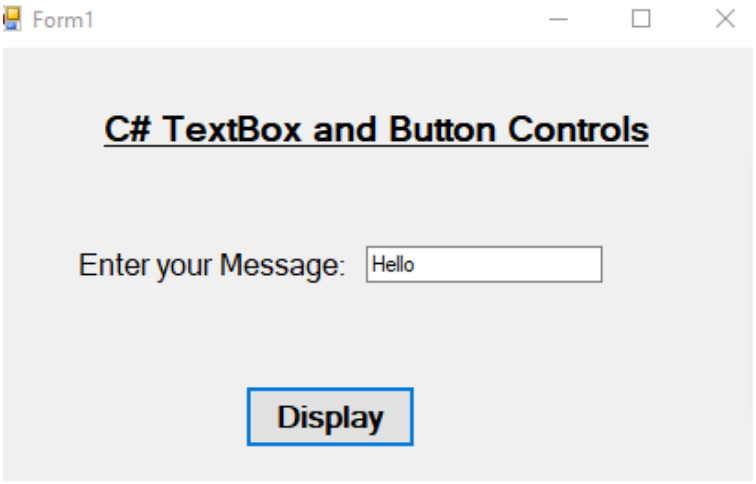
1) **BackgroundImage –** The background image used for the Button control.
1) **Image –** The image that will be displayed on a control.
2) **ImageAlign –** The alignment of the image that will be displayed on the control.

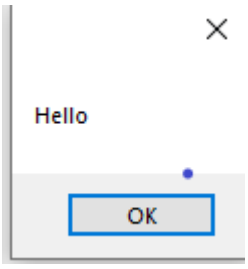
Example –Program to demonstrate TextBox and Button control.




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

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         MessageBox.Show(textBox1.Text);
      }
   }
}
```

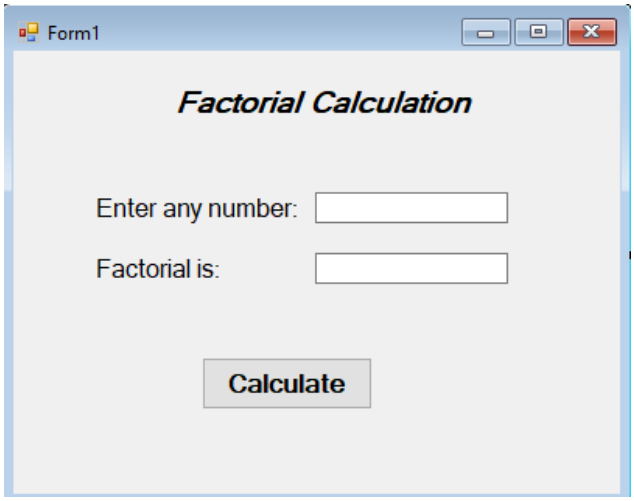
Output:





e) Write a program to calculate factorial of a number using windows form.

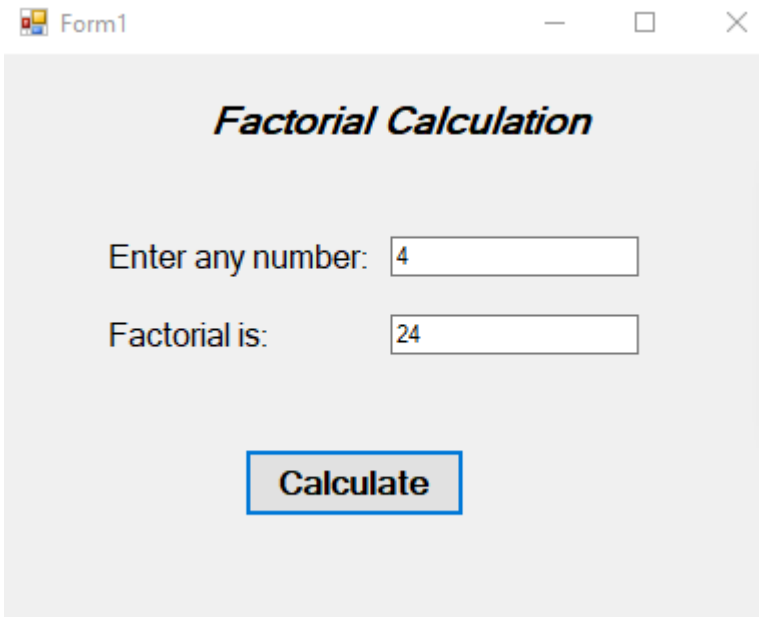
Design View-

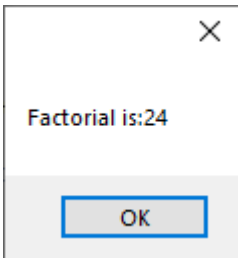


Coding-

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

namespace WindowsFormsApplication2
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         int n, i, fact = 1;
         n = int.Parse(textBox1.Text);
         for (i = 1; i <= n; i++)
         {
            fact = fact * i;
         }
         textBox2.Text = fact.ToString();
         MessageBox.Show("Factorial is:" + fact);
      }
   }
}
```

Output-





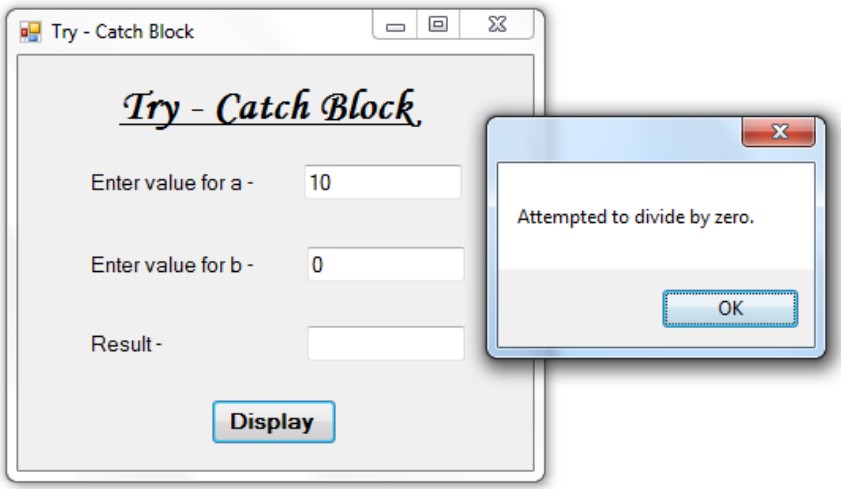
Q. 3 Attempt any three of the following (5 Marks each) 15

a) Write a Program for demonstration of handling exception.

```
namespace tryCatch
{
    public partial class Form1 : Form
    {
        public Form1( )
        {
            InitializeComponent( );
        }
        private void btn_display_Click(object sender, EventArgs e)
        {
            try
            {
                int a = int.Parse(txt_a.Text);
                int b = int.Parse(txt_b.Text);
                int c = a / b;
                txt_result.Text = c.ToString( );
            }
```

```
                catch (DivideByZeroException ex)
                {
                        MessageBox.Show(ex.Message);
                }
                MessageBox.Show("Program is completed");
        }
    }
}
```

Output:



b) Explain ListBox control in detail.

A ListBox control provides a user interface to display a list of items. Users can select one or more items from the list. We can create a ListBox control using a Forms designer at design-time by simply drag and drop a ListBox control from Toolbox to a Form in Visual Studio.

Properties –

1) **Name –** It represents a unique name of a control. It is used to access the control in the code.
2) **BackColor –** BackColor property is used to set background color of a control.
3) **ForeColor –**It is used to set foreground color of a control.
4) **Items –** The Items property is used to add and work with items in a ListBox.
5) **SelectionMode –** It defines how items are selected in a ListBox. It has four values None, One, MultiSimple,and MultiExtended.
6) **Size –** It specifies the size of the control in pixels.
7) **Sorted –** If the Sorted property is set to true then the ListBox items are sorted.

8) **Text –** Text property of a control represents the current text of a control.
9) **Visible –** Determines whether the control is visible or hidden.

Events –

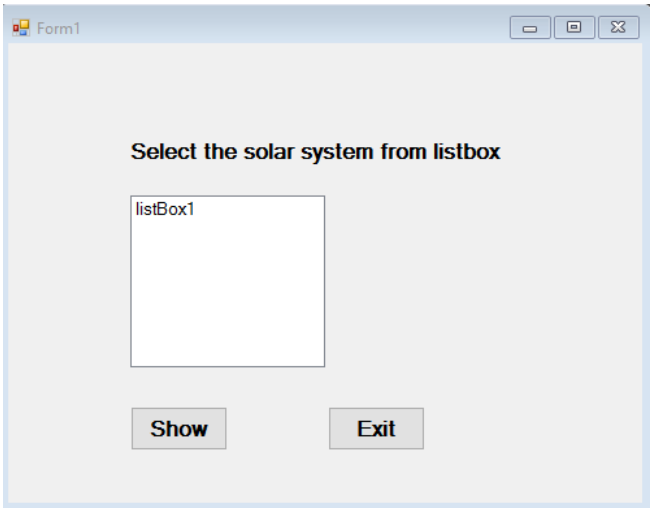
1) **BackColorChanged –** Event raised when the value of the BackColor property changes.
2) **Click –** Event raised when the control is clicked.
3) **MouseEnter –** Event raised when the mouse enters the visible part of the control.
4) **MouseLeave –** Event raised when the mouse leaves the visible part of the control.
5) **SelectedIndexChanged –** Event raised when the item selection is changed in a ListBox.

Runtime Methods –

1) **Add( ) –** Add the new item to the list
   Syntax – ListBoxName.Items.Add(string or object);
1) **Insert( ) –** Insert the new item at the desired point in the list.
   Syntax – ListBoxName.Items.Insert(IndexNumber, string or object);
2) **Remove( ) –** It removes the specified the item from list.
   Syntax – ListBoxName.Items.Remove(string or object);
3) **RemoveAt( ) –** It removes the item with the specified index number.
   Syntax –ListBoxName.Items.RemoveAt(IndexNumber);
4) **Clear ( ) –** It removes all items from the collection.
   Syntax –ListBoxName.Items.Clear( );

Example- Program to demonstrate Listbox Control in C#.NET.

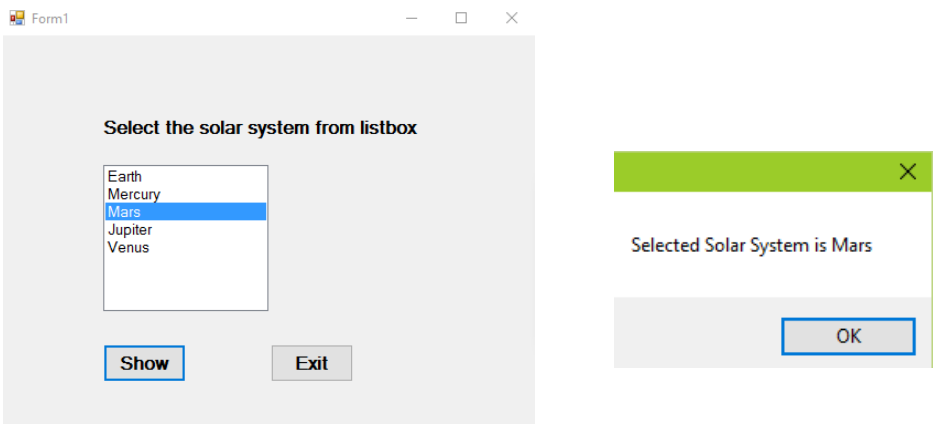
Designing-



Coding-

```
using System;
using System.Windows.Forms;
namespace ListBoxEx
{
    public partial class Form1 : Form
      {
            public Form1()
            {
                  InitializeComponent();
            }
            private void Form1_Load(object sender, EventArgs e)
            {
                  //Add the items into the
                  ListBox
                  listBox1.Items.Add("Earth");
                  listBox1.Items.Add("Mercur);
                  listBox1.Items.Add("Mars");
                  listBox1.Items.Add("Jupiter";
                  listBox1.Items.Add("Venus");
            }
            private void button1_Click(object sender, EventArgs e)
            {
                string n;
                n = listBox1.Text;
                MessageBox.Show(" Selected Solar System is " + n);
            }
      }
}
```

Output:



c) Explain C# function.

Function is a block of code that perform specified task. Function is used to execute statements specified in the code block. To use a method, you need to −

1) Define the method
2) Call the method

1) Defining Method in C# -

The syntax for defining a method in C# is as follows –

Syntax –

```
<Access_Specifier> <Return_Type> Function_Name (<Parameter_List>)
{
    // function body
    // return statement
}
```

Where,

Access_Specifier: It is used to specify function accessibility in the application.

Return_Type: It is used to specify the data type of function return value.

Function_Name: It is a unique name that is used to make Function call.

Parameter_List: It is a list of arguments that we can pass to the function during call.

Function_Body: It is a block that contains executable statements.

Access_Specifier, Parameter_List and return statement are optional.

2) Calling Method in C# -

The syntax for calling method in C# is as follows –

Syntax –

Function_Name (<Parameter_List >);

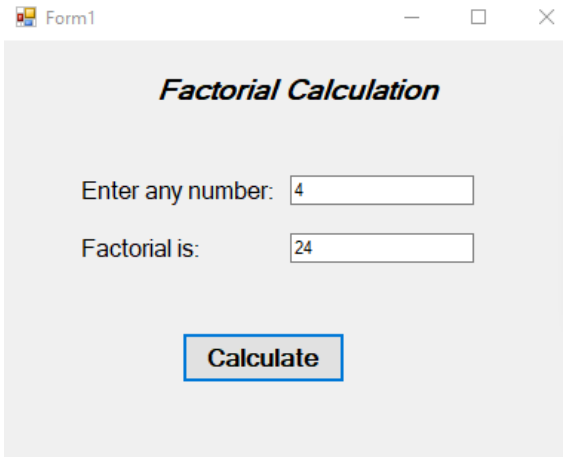
Where,

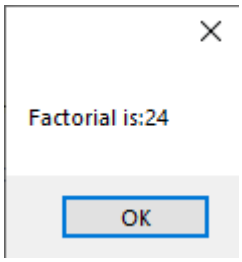
Function_Name: It is a name through which function is call.

Parameter_List: It is a list of arguments that we can receive through the function call.

In C#, value-type parameters are that pass a copy of original value to the function rather than reference. It is the default approach to pass parameters to the functions. C# provides out keyword to pass arguments as out-type where as ref keyword to pass argument as reference-type.

Example- Program to demonstrate function in C#.






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

namespace WindowsFormsApplication2
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
        int n= int.Parse(textBox1.Text);
        int res = fact(n);
        textBox2.Text = res.ToString();
        MessageBox.Show("Factorial is:" + res);
      }
      public int fact(int n)
      {
         int f = 1, i;
         for (i = 1; i <= n; i++)
         {
            f = f * i;
         }
         return f;
      }
   }
}
```

d) Explain concept of delegates.

Delegate is a type safe function. C# delegates are similar to pointers to functions, in C or C++. A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

Delegates can be of two types in C#-

1. **Singlecast delegate**

   A delegate that holds single method's reference at a time known as Singlecast delegate.

2. **Multicast Delegate**

   The delegate can points to multiple methods. A delegate that points multiple methods is called a multicast delegate. Delegate objects can be composed using the "+" operator.

   A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed. The "-" operator can be used to remove a component delegate from a composed delegate.

Syntax to define delegate-

Step 1: Define a Delegate

<access_modifier> delegate <return type> <delegate_name> (<parameters>);

Step 2: Create an Instance of a Delegate

<delegate_name> <del_object_name> = new <delegate_name> (<method_name>);

Step 3: Invoking a Delegate

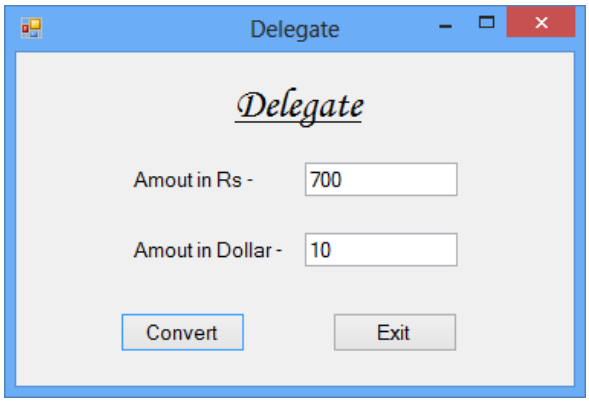
<del_object_name> (<parameters>);

Example –program demonstrates singlecasting of a delegate.

```
namespace delegateExample
{
   public delegate double dollarDelegate(double n);              // Step 1 - define a delegate
   public partial class Form1 : Form
   {
      public Form1( )
      {
         InitializeComponent( );
      }
      private void btn_convert_Click(object sender, EventArgs e)
      {
       double amount = double.Parse(txt_rs.Text);
```

```
        Conversion obj = new Conversion( );
        dollarDelegate dd = new dollarDelegate(obj.dollar);  // Step 2 - Delegate instantiation
         txt_dollar.Text = (dd(amount)).ToString( );                    // Step 3 - Delegate Invocation
       }
      private void btn_exit_Click(object sender, EventArgs e)
      {
        Application.Exit( );
       }
   }
   public class Conversion
   {
     public double dollar(double x)
      {
         return x / 70;
       }
   }
}
```

Output:



e) Explain Dialog Boxes in C#.

A dialog box is a type of window, which is used to enable common communication or dialog between a computer and its user. It performs the common tasks like saving a file, choosing a font etc. The examples are – FontDialog, ColorDialog, OpenDialog, and SaveDialog.

a) OpenFileDialog – The OpenFileDialog allows us to choose a file to be opened in an Application.

Properties

1) **Name –** It represents a unique name of a control. It is used to access the control in the code.
2) **FileName –** The file first shown in the dialog box or the last one selected by the user.
3) **Filter –** The file filters to display in the dialog box.
4) **Multiselect –** Control whether multiple files can be selected in the dialog.
5) **Title –** The string to display in the title bar of the dialog box.

Event –

1) **FileOk –** Event raised when the user clicks the open or save button in the dialog box.

b) SaveFileDialog – The SaveFileDialog box is used to allow the user to select the destination and name of the file to be saved.

Properties –

1) **Name –** It represents a unique name of a control. It is used to access the control in the code.
2) **FileName –** The file first shown in the dialog box or the last one selected by the user.
3) **Filter –** The file filters to display in the dialog box.
4) **Title –** The string to display in the title bar of the dialog box.

Event –

1) **FileOk –** Event raised when the user clicks the open or save button in the dialog box.

c) FontDialogBox – The FontDialogBox is used to allow the user to select font settings.

Properties –

1) **Name –** It represents a unique name of a control. It is used to access the controlin the code.
2) **Font –** The font selected in the dialog box.

Event –

1) **Apply –** Event raised when the user clicks the apply button in the dialog box.

d) ColorDialogBox –

The ColorDialogBox is used to allow the user to select a color.

Properties –

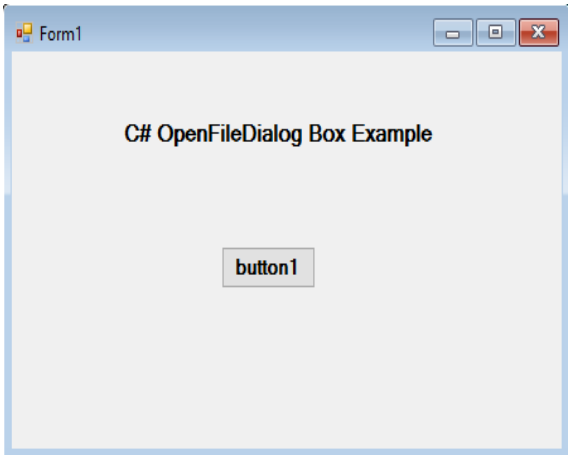
1) **Name –** It represents a unique name of a control. It is used to access the control in the code.
2) **Color –** The color selected in the dialog box.

Event –

1) **HelpRequest –** Event raised when the user clicks the Help button.

Example- Program to demonstrate OpenFileDialog Box in C#.Net.

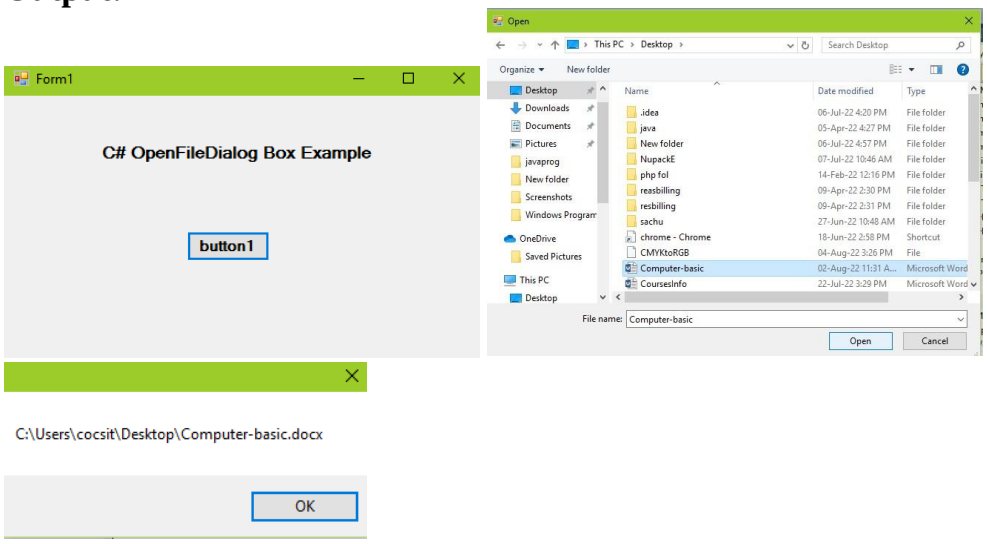
Design View-



Coding:

```
using System;
using
System.Windows.Form
s;namespace
Openfiledialog
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            OpenFileDialog dlg = new
            OpenFileDialog();dlg.ShowDialog();

            if (dlg.ShowDialog() == DialogResult.OK)
            {
                string fileName;
                fileName =
                dlg.FileName;
                MessageBox.Show(fileName);
            }
        }
    }
}
```

Output:



Q. 4 Attempt any three of the following (5 Marks each) 15

a) Explain Arraylist class with example.

ArrayList is a non-generic type of collection in C#. It can contain elements of any data types. It is similar to an array, except that it grows automatically as you add items in it. Unlike an array, you don't need to specify the size of ArrayList.

Note – Need to add System.Collections namespace to use ArrayList class in the project.

Syntax –

ArrayList array_list_name = new ArrayList( );

Example –

ArrayList myList = new ArrayList( );

You can also add items when you initialize it using object initializer syntax.

ArrayList myList = new ArrayList( ) { 10, "AMOL" };

Property –

1) Count –

It gets the number of elements actually contained in the ArrayList.

Methods –

1) Add(Object value) –

This method is used to add a single element at the end of ArrayList.

2) Insert(int index, Object value) –

This method is used to insert a single element at the specified index.

3) Remove(Object value) –

This method is used to remove a specified element from an ArrayList.

4) RemoveAt(int index) –

This method is used to remove an element from the specified index location.

5) Contains(Object value) –

This method checks whether specified element exists in the ArrayList or not. Returns true if exists otherwise false.

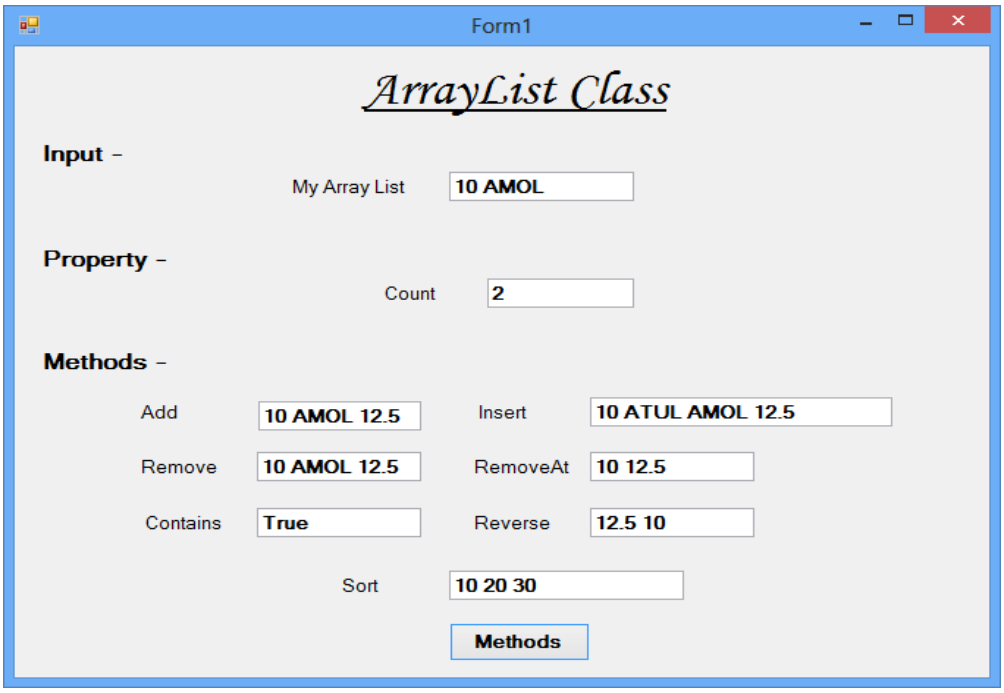
6) Reverse( ) –

This method arranges elements in reverse order. Last element at zero index and so on.

7) Sort ( ) –

This method arranges elements in ascending order. However, all the elements should have same data type so that it can compare with default comparer otherwise it will throw runtime exception.

Program – Write a program to demonstrate ArrayList class in C#.NET.




```
using System.Collections;
namespace arrayListClass
{
   public partial class Form1 : Form
   {
      ArrayList myList = new ArrayList( ) { 10, "AMOL" };
       public Form1( )
       {
         InitializeComponent( );
       }
       private void Form1_Load(object sender, EventArgs e)
       {
            for (int i = 0; i < myList.Count; i++)
            {
               txt_mylist.Text = txt_mylist.Text + myList[i] + " ";
            }
       }
       private void btn_methods_Click(object sender, EventArgs e)
        {
            //count property
            txt_count.Text = (myList.Count).ToString( );
             //add method
             myList.Add(12.5);
```

```
            for (int i = 0; i < myList.Count; i++)
            {
                    txt_add.Text = txt_add.Text + myList[i] + " ";
             }
            //insert method
           myList.Insert(1, "ATUL");
           for (int i = 0; i < myList.Count; i++)
           {
               txt_insert.Text = txt_insert.Text + myList[i] + " ";
            }
           //remove method
           myList.Remove("ATUL");
           for (int i = 0; i < myList.Count; i++)
           {
              txt_remove.Text = txt_remove.Text + myList[i] + " ";
            }
           //removeat method
           myList.RemoveAt(1);
           for (int i = 0; i < myList.Count; i++)
           {
             txt_removeat.Text = txt_removeat.Text + myList[i] + " ";
           }
           //contains method
           txt_contains.Text = (myList.Contains(12.5)).ToString( );
           //reverse method
           myList.Reverse( );
           for (int i = 0; i < myList.Count; i++)
           {
              txt_reverse.Text = txt_reverse .Text + myList[i] + " ";
            }
           //sort method
           ArrayList newList = new ArrayList( ) { 30, 10, 20 };
           newList.Sort( );
           for (int i = 0; i < newList.Count; i++)
           {
              txt_sort.Text = txt_sort.Text + newList[i] + " ";
           }
      }
   }
}
```

b) Explain creating and using interfaces.

It is also a user defined data type like a class but can contain only abstract methods. The default scope of the members of an interface is public where as its private in case of a class.

Every abstract method of an interface should be implemented by the child class of an interface. Interface cannot contain any data member. By default, every method of an interface is abstract so we don't require using abstract keyword just like in case of abstract class. If required an interface can inherit from another interface.

Syntax –

```
<Access_Specifier> Interface <Interface_Name>
{
        // Declaration of abstract methods
}
```

Program – Design a windows application to demonstrate concept of an Interface.

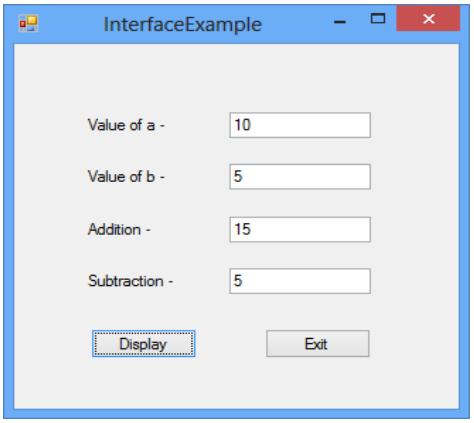
Step 1 – Create an Interface file named InterfaceDemo.cs, (Project Menu - Add New Item - Interface).

```
namespace InterfaceDemo
{
   interface iInterface1
   {
      int add(int x, int y);
   }
   interface iInterface2 : iInterface1
   {
      int sub(int x, int y);
   }
   public class Sample : iInterface2
   {
      public int add(int x, int y)
      {
         return x + y;
      }
      public int sub(int x, int y)
      {
         return x - y;
      }
   }
}
```

Step 2 – Create a new Windows Form named InterfaceExample.cs, (Project Menu - Add Windows Form)

```
namespace InterfaceDemo
{
   public partial class InterfaceExample : Form
   {
      public InterfaceExample( )
      {
         InitializeComponent( );
      }
      private void btn_display_Click(object sender, EventArgs e)
      {
          int a = int.Parse(txt_a.Text);
          int b = int.Parse(txt_b.Text);
          Sample s = new Sample( );
          txt_add.Text = (s.add(a, b)).ToString();
          txt_sub.Text = (s.sub(a, b)).ToString( );
      }
      private void btn_exit_Click(object sender, EventArgs e)
      {
          Application.Exit( );
      }
   }
}
```

Output:



c) Explain connected data access in C#.

The ADO.NET Framework supports two models of Data Access Architecture, connected and disconnected modes. A connected mode of operation in ADO.Net is one in which the connection to theunderlying database is alive throughout the lifetime of the operation.

In Connection Oriented Data Access, when you read data from a database by using a DataReader object, an open connection must be maintained between your application and the

Data Source.

Example- a program for demonstration of connected data access.

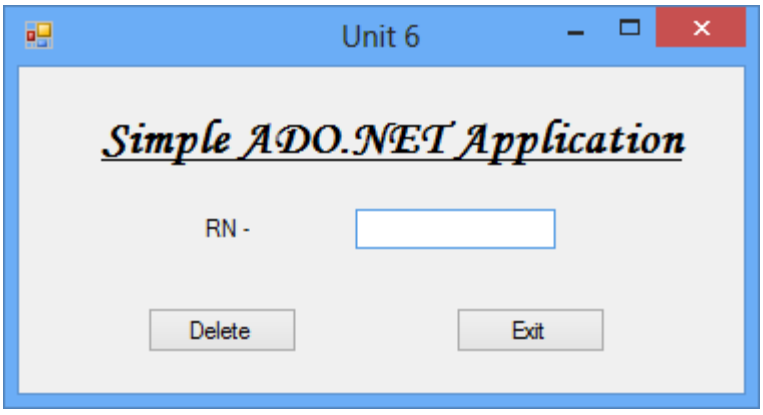
Consider college.mdb database is created in MS-Access with Student table,

Field	Data Type
RN	Auto Number
Name	Text
Fees	Number

RN	Name	Fees
1	Amol	1000
2	Balaji	2000
3	Chetan	3000

Table – Student (Design and Data Sheet View)

Program – Develop an application that will remove record from Student table depending upon roll number.



Property Table –

Control	Property	
	Name	Text
Label1	-	RN
textBox1	txt_rn	-
button1	btn_delete	Delete

button2	btn_exit	Exit

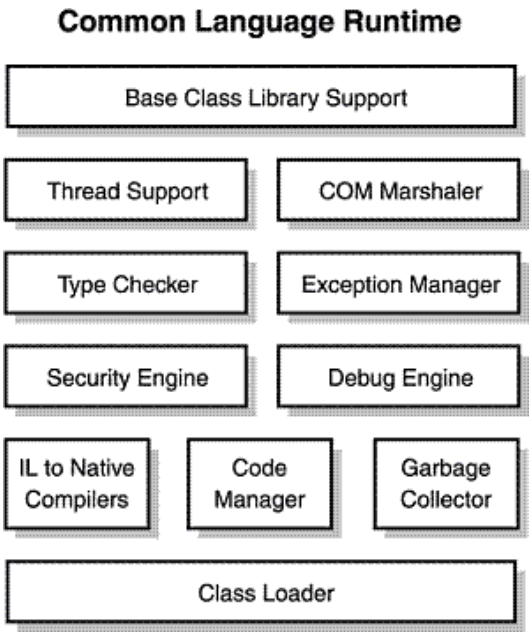
```
using System.Data.OleDb;
namespace simpleADO
{
   public partial class Form1 : Form
   {
      public Form1( )
      {
         InitializeComponent( );
      }
      private void btn_delete_Click(object sender, EventArgs e)
      {
                  OleDbConnection con = new OleDbConnection ("Provider = Microsoft.Jet.oledb.4.0;
                  Data Source = d:\\college.mdb");
                  con.Open();
                  string query = "delete from student where rn = " + txt_rn.Text ;
                  OleDbCommand cmd = new OleDbCommand(query, con);
                  cmd.ExecuteNonQuery( );
                  MessageBox.Show("Record is deleted");
                  con.Close( );
      }
      private void btn_exit_Click(object sender, EventArgs e)
      {
         Application.Exit( );
      }
   }

}
```

d) Explain CLR architecture.

Common Language Runtime (CLR)

It is the heart of the .NET framework. It is the responsibility of the CLR to take care of the code execution of the program. The CLR is a runtime engine that handles memory allocation, security, code verification, type verification, exception handling, garbage collection and many more.

Following are the component structure of Common Language Runtime –



1. Class Loader

It is used to load all the classes at runtime for execution of .Net application.

2. MSIL to native Compiler

It is a JIT (Just-In-Time) compiler; it will convert MSIL code to native code.

3. Code manager

It manages the code during execution.

4. Garbage Collector

Garbage Collector handles automatic memory management and it will release memory of unused objects in an application.

5. Security Engine

It enforces security permissions at code level security, folder level security, and machinelevel security using Dot Net Framework setting and tools provided by Dot Net.

6. Debug Engine

CLR allows us to perform debugging an application during runtime.

7. Type Checker

Type checker will verify data types used in the application with CTSstandards supported by CLR, this provides type safety.

8. Exception Manager

Exception Manager will handle exceptions thrown by application by while executingTry-catch block provided by an exception.

9. Thread support

Threads are basically light weight processes responsible for multi-tasking within a singleapplication. It provides multithreading support to our application.

10. Com Marshaler

It allows the communication between the application and COM objects.

11. Base class library support

Which provides all types of classes that application need at runtime.

e) WAP for demonstration of accessing data from database.

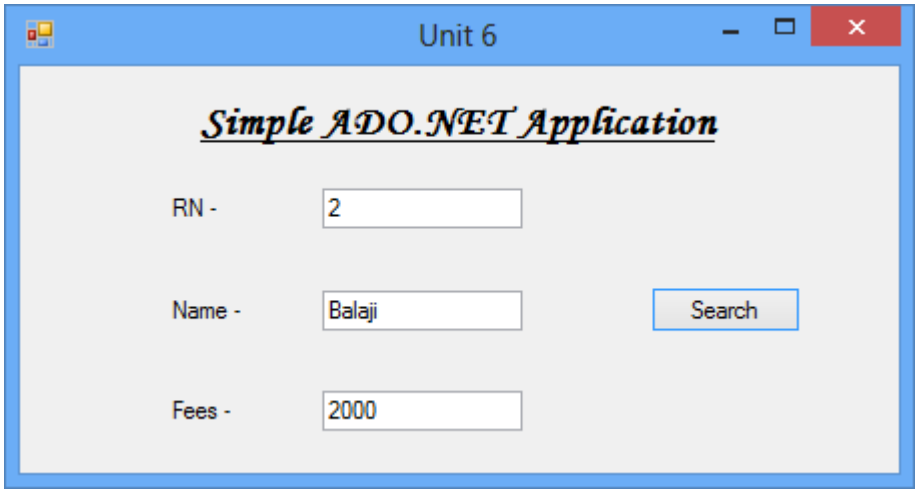
The SQL SELECT statement returns a result set of records from one or more tables. Consider college.mdb database is created in MS-Access with Student table,

Field	Data Type
RN	Auto Number
Name	Text
Fees	Number

RN	Name	Fees
1	Amol	1000
2	Balaji	2000
3	Chetan	3000

Table – Student (Design and Data Sheet View)

Program -



Property Table –

Control	Property	
	Name	**Text**
Label1	-	RN
Label1	-	Name
Label1	-	Fees
textBox1	txt_rn	-
textBox2	txt_name	-
textBox3	txt_fees	-
button1	btn_search	Search

```
using System.Data.OleDb;

public partial class Form1 : Form
{
   public Form1( )
   {
      InitializeComponent( );
   }
   private void btn_search_Click(object sender, EventArgs e)
   {
         OleDbConnection con = new OleDbConnection ("Provider =
         Microsoft.Jet.oledb.4.0;Data Source = d:\\college.mdb");
         con.Open();
         string query = "select * from student where rn =" +
         txt_rn.Text;OleDbCommand cmd = new
         OleDbCommand(query, con); OleDbDataReader dr =
         cmd.ExecuteReader( );
         if (dr.Read( ))
         {
            txt_name.Text =
            dr["name"].ToString( );
            txt_fees.Text =
            dr["fees"].ToString( );
         }
```

```
            else
            {
               MessageBox.Show("Record not
               found");txt_clear( );
            }
            con.Close( );
        }
    }
}
```

Q .5 Short notes on any three of the following (5 Marks each) 15

a) .Net and IDE

Dot Net (.Net) is not a programming language, it is a platform introduced by Microsoft todevelop various kinds of applications. Microsoft began developing .Net framework in the late 1990, under the name of NGWS (Next Generation Windows Services). By late 2000, the first beta version 1.0 was released.

.Net framework includes a large class library called as a FCL (Framework Class Library). Program written for .Net framework executes in an environment called as CLR (Common Language Runtime). FCL and CLR together constitute .Net framework.

Microsoft produces an IDE (Integrated Development Environment) for .Net software called as Visual Studio. .Net framework lead to a family of .Net platforms targeting mobile computing, embedded devices, operating systems and so on.

When we start a new project in Visual Studio .NET, we will see a group of windows opened within the development environment.

1. Menu Bar
2. Standard Toolbar
3. ToolBox
4. Forms Designer
5. Output Window
6. Solution Explorer
7. Properties Window

1. Menu Bar

It is a horizontal bar, typically located at the top of the screen below the title bar, containing drop-down menus.

2. Standard Toolbar

A standard toolbar is a horizontal or vertical strip that contains buttons that are bound to commands.

3. ToolBox

The Toolbox window contains a list of controls or components that we can drag and drop onto the form design. To view the Toolbox window, select View, Toolbox from the Visual Studio .NET menu bar.

4. Forms Designer

When we create a new project that is a Windows application, Visual Studio .NET will open the project in Designer view. The Form Designer is usually the windows in the middle of the Visual Studio .NET environment that contains the design-time representation of the application's form object. We can also select the Designer option from the View menu to put the project in Designer view

5. Output Window

The Output window can display status messages for various features in the integrated development environment (IDE). To open the Output window, on the menu bar, choose View/Output (or click CTRL + ALT + O).

6. Solution Explorer

Solution Explorer is a tool window in the Visual Studio integrated development environment (IDE) that displays the contents of a solution, which includes the solution's projects and each project's items.

In Visual Studio .NET, a solution is a set of one or more projects that are part of the same application. The Solution Explorer window shows us an expandable list of projects, each project's references, and each project's components. If this window is closed, we can open it by selecting the View, Solution Explorer menu item.

7) Properties Window

Use this window to view and change the design-time properties and events of selected objects that are located in editors and designers. We can find Properties Window on the View menu. We can also open it by pressing F4. Once this window is visible, we can either view the list alphabetically or categorized by attribute.

b) ADO.NET Architecture

ADO stands for ActiveX Data Objects. ADO.NET is a database technology of .NETFramework used to connect application system and database server. ADO.NET is a part of the .NET Framework. ADO.NET consists of a set of classes used to handle data access.

The following figure shows the ADO.NET objects at a glance:

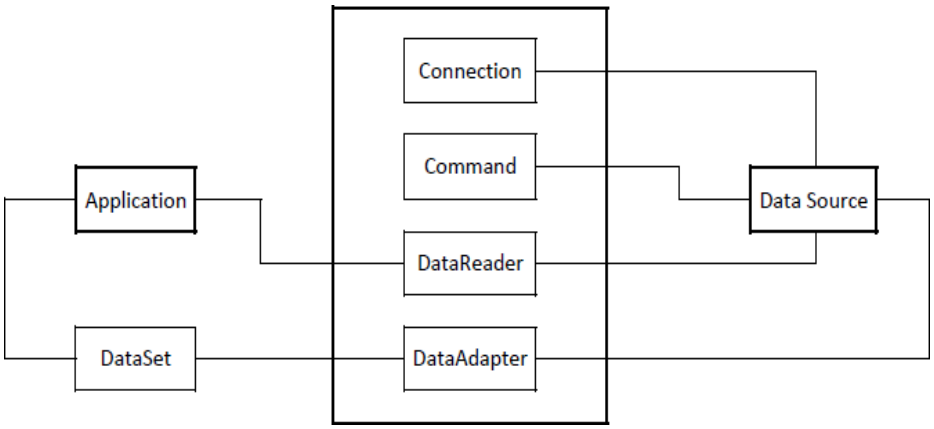


Figure – ADO.Net Architecture

System.Data namespace is the core of ADO.NET and it contains classes used by all data providers.

1) Data Providers –

A key component of an ADO.NET is Data Provider. The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases.

The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider, OLEDB Data Provider and ODBC Data Provider. SQL Server uses the SqlConnection object, OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

2) Connection –

The Connection Object provides physical connection to the Data Source. Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

3) Command –

The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

Different execute methods of ADO.NET command object are ExecuteScalar ( ), ExecuteReader ( ), ExecuteNonQuery ( ). ExecuteScalar ( ) fetches only a single object. ExecuteReader ( ) fetches result set with multiple rows and loads to DataReader. ExecuteNonquery ( ) executes SQL statements for insert, update and delete.

4) DataReader –

The DataReader Object is a stream-based, forward-only, read-only retrieval of query results from the Data Source, which do not update the data. DataReader requires a live connection with the database.

5) DataAdapter –

DataAdapter Object populates a Dataset Object with results from a Data Source. It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

6) DataSet –

DataSet class provides mechanisms for managing data when it is disconnected from the data source. It is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

c) String Class

In C#, string is an object of System.String class that represents sequence of characters. We can perform many operations on strings such as concatenation, comparison, getting substring, search, trim, replacement etc.

string vs String –

In C#, string is keyword which is an alias for System.String class. That is why string and String are equivalent. We are free to use any naming convention.

```
string s1 = “abc”;
String s2 = “def”;
```

a) C# String Property –

1) Length – Gets the number of characters in the current String object.

Syntax – s1.Length // s1 is string object

C# String methods –

1) Compare(string s1, string s2) –

The C# Compare( ) method is used to compare first string with second string. It returns an integer value. If both strings are equal, it returns 0. If first string is greater than second string, it returns 1 else it returns -1.

2) Concat(string s1, string s2) –

It is used to concatenate two specified instances of String. It returns a string object.

3) Contains(string s1) –

It is used to return a value indicating whether the specified substring occurs within this string or not. If the specified substring is found in this string, it returns true otherwise false.

4) Copy(string s1) –

It is used to create a new instance of String with the same value as a specified String. Its return type is string.

5) Equals(string s1, string s2) –

It is used to check whether two specified String objects have the same value or not. If both strings have same value, it return true otherwise false.

6) Remove(Int beginIndex, Int length) –

It is used to get a new string after removing all the characters from specified beginIndex till given length. If length is not specified, it removes all the characters after beginIndex.

7) Replace(string s1, string s2) –

It is used to get a new string in which all occurrences of a specified Unicode character in this string are replaced with another specified Unicode character. It returns a string.

8) Substring(Int startIndex, Int length) –

The C# SubString( ) method is used to get a substring from a String. The substring starts at a specified character position and continues to the end of the string. It returns a string.

9) ToLower( ) –

The C# ToLower( ) method is used to convert a string into lowercase. It returns a string.

10) ToString( ) –

The C# ToString( ) method is used to return instance of String. It returns a string object.

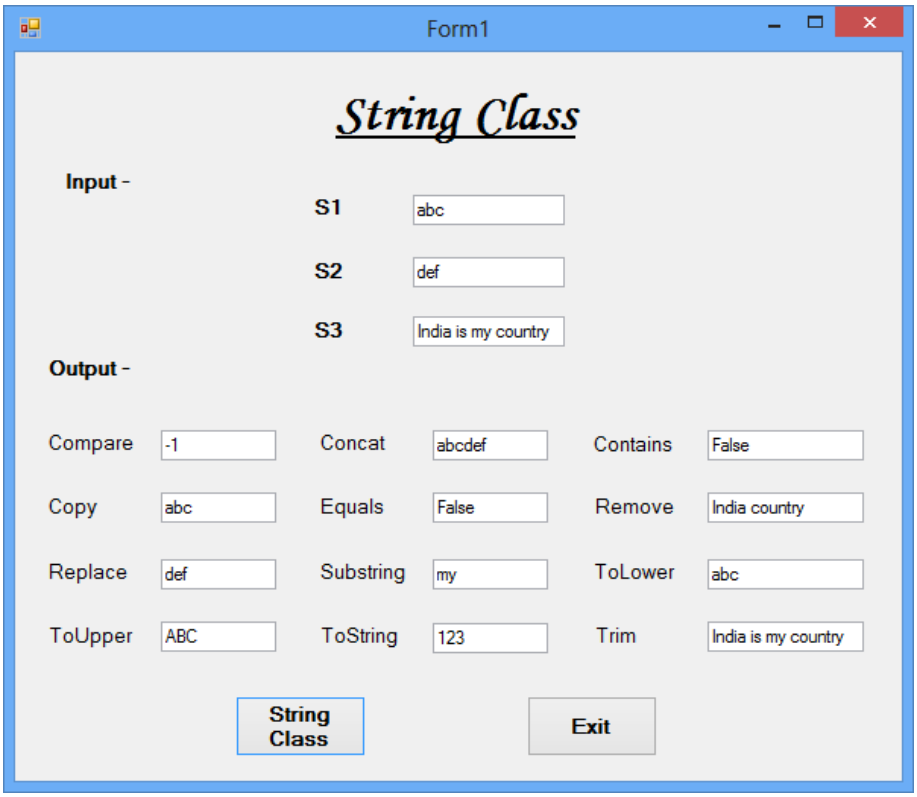
11) ToUpper( ) –

The C# ToUpper( ) method is used to convert string into uppercase. It returns a string.

12) Trim( ) –

It is used to remove all white-space characters from the current String object.

Example – Program to demonstrate string methods in C#.NET.




```
namespace stringExample
{
   public partial class Form1 : Form
   {
      public Form1( )
      {
         InitializeComponent( );
      }
      private void btn_stringclass_Click(object sender, EventArgs e)
      {
            string s1 = textBox1.Text;
            String s2 = textBox2.Text;
            string s3 = textBox3.Text;
            txt_compare.Text = (string.Compare(s1, s2)).ToString( );          //Compare Method
```

```
            txt_concat.Text = string.Concat(s1, s2);                    //Concat Method
            txt_contains.Text = (s1.Contains(s2)).ToString( );          //Contains
            txt_copy.Text = string.Copy(s1);                            //Copy
            txt_equals.Text = (string.Equals(s1, s2)).ToString( );      //Equals
            //replace
            txt_replace.Text = s1.Replace(s1, s2);
            //Tolower
            txt_tolower.Text = s1.ToLower( );
            //Toupper
            txt_toupper.Text = s1.ToUpper( );
            //remove
            txt_remove.Text = s3.Remove(5, 6);
            //substring
            txt_substring.Text = s3.Substring(9,2);
            //Tostring
            int n = 123;
            txt_tostring.Text = n.ToString( );
            //Trim
            string s4 = " India is my country ";
            txt_trim.Text = s4.Trim( );
        }
        private void btn_exit_Click(object sender, EventArgs e)
        {
            Application.Exit( );
        }
    }
}
```

e) Namespace (DLL Library)

Namespace are used to group classes, methods and other code into a package for use in many different files.

A namespace shall be viewed as a container for classes and interface. As we know, a folder on our file system contains files. A namespace is like a folder. By placing the classes into namespaces, we can group related classes together. We can also avoid the risk of name collisions.

To access the class of a namespace, we need to use namespacename.classname. We can use using keyword so that we don't have to use complete name all the time.

Syntax –

```
namespace namespace_name
{
        // Classes, Methods, etc.
}
```

Program- Design a windows application to demonstrate creating and using custom namespace using DLL.

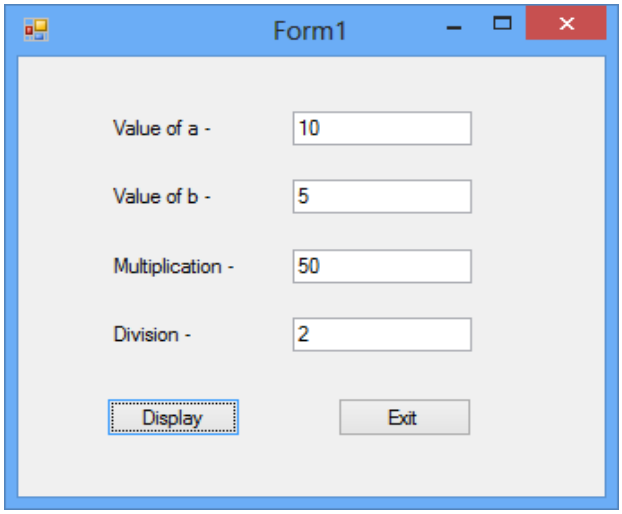
Step 1 – Create a new project of type Class Library named myNameSpace.cs and build it.

```
namespace myNameSpace
{
   public class operation1
   {
      public int mul(int a, int b)
      {
         return a * b;
      }
   }
   public class operation2
   {
      public int div(int a, int b)
      {
         return a / b;
      }
    }
}
```

Step 2 – Create a new project of type Windows Form Application named Custom_Namespace and add reference of myNameSpace.dll file that already created (Project – Add Reference – Browse – Project Name – bin – debug – myNameSpace.dll).

```
using myNameSpace;
   namespace Custom_Namespace
   {
   public partial class Form1 : Form
   {
      public Form1( )
      {
         InitializeComponent( );
      }
      private void btn_display_Click(object sender, EventArgs e)
      {
       int a=int.Parse(txt_a.Text);
       int b=int.Parse(txt_b.Text);
       operation1 obj1 = new operation1( );
       operation2 obj2 = new operation2( );

        txt_mul.Text=(obj1.mul(a,b)).ToString();
        txt_div.Text=(obj2.div(a,b)).ToString();
      }
      private void btn_exit_Click(object sender, EventArgs e)
      {
         Application.Exit( )
      }
   }
}
```

Output:



e) Properties in C#

Properties are data members of a class using which we can expose values associated with a class outside the class environment.

A Property in C# is used to set and get data from a data field i.e. variable of a class. It is never used to store any data, it just acts an interface to transfer the data outside the class.

We use the properties as public data members of a class but they are actually contains special methods called as accessors. The Accessors are nothing but the special methods which are used to set and get the values from the data member of a class.

Property can be following types:

1. Read/Write Properties: A property with both get and set accessor is a Read-Writeproperty.
2. Read Only Properties: A property with only get accessor is a Read only property.
3. Write Only Properties: A property with only set accessor is a Write only property.

In C#, we use get and set accessors to implement properties.

Syntax –

```
<Access_Specifier> <type> <Property_Name>
{
        get { // get statements }          // Only get property
            is read only
        set { // set statements }          // Only set property
            is write only
        // when get and set then property is read-write
}
```

Example – Program to demonstrate concept of properties in C#.

```
namespace propertiesExample
{
   public partial class Form1 : Form
   {
      public Form1( )
      {
```

```
        InitializeComponent( );
      }
      private void btn_set_Click(object sender, EventArgs e)
      {
          number c = new number( );
          txt_first.Text=(c.noProperty).ToString( );
          c.noProperty = 10;
          txt_second.Text = (c.noProperty).ToString( );
       }
       private void btn_exit_Click(object sender, EventArgs e)
       {
          Application.Exit( );
       }
}
public class number
{
  int no = 5;
  public int noProperty                    // Property definition
  {
     get { return no; }
     set{no=value;}
   }
  }
}
```

Output: