USN | | | | | | | | | | |

13MCA53

Fifth Semester MCA Degree Examination, Dec.2016/Jan.2017
Programming using C# and •NET

Time: 3 hrs. Max. Marks:100

Note: *Answer any FIVE full questions.*

1 a. Explain the components and benefits of •Net framework with the help of architecture diagram. **(12 Marks)**

b. What is an assembly? Explain each component of an assembly. **(08 Marks)**

2 a. Explain the different data types in C#. **(10 Marks)**

b. What are different types of type conversions supported by C#. **(06 Marks)**

c. Describe the significance of 'is' and 'as' operator. **(04 Marks)**

3 a. Explain different method parameter modifier with suitable examples. **(08 Marks)**

b. Write a short notes on the following :
i) Partial classes and methods
ii) Indexers. **(08 Marks)**

c. Write a program in C# to read a jagged array and display the sum of all the elements of inner array. **(04 Marks)**

4 a. How method overriding is different from method overloading? Illustrate with examples. **(08 Marks)**

b. Explain sealed classes and sealed methods. **(05 Marks)**

c. How would you enforce encapsulation using accessor and mutators? Explain with suitable code. **(07 Marks)**

5 a. How delegates are used in C#? Discuss single cast and multicast delegates? **(10 Marks)**

b. "Catching one exceptions programmatically is good and necessary mechanism" justify with suitable examples. **(06 Marks)**

c. Write short notes on multiple event handlers. **(04 Marks)**

6 a. Discuss panel control with respect to windows form. **(06 Marks)**

b. Explain the implementation of combobox in C#. **(08 Marks)**

c. Write a program for addition of two numbers, by accepting numbers from text boxes and display the result in the form using button click event. **(06 Marks)**

7 a. Explain the components of ADO·NET entity framework **(06 Marks)**

b. How data adapter is used to build database applications? **(10 Marks)**

c. Discuss the components of dataset. **(04 Marks)**

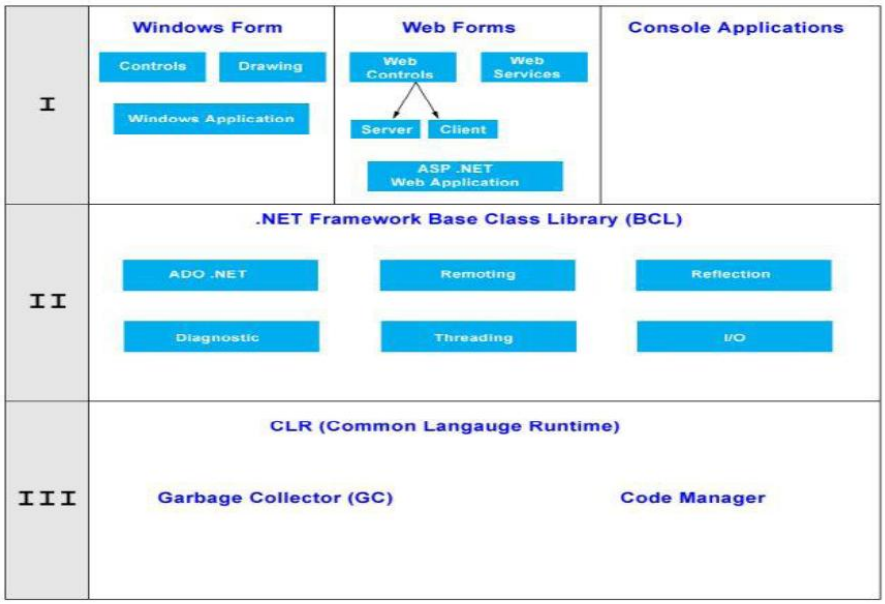
8 a. Explain different validation control with suitable examples supported by ASP·NET. **(08 Marks)**

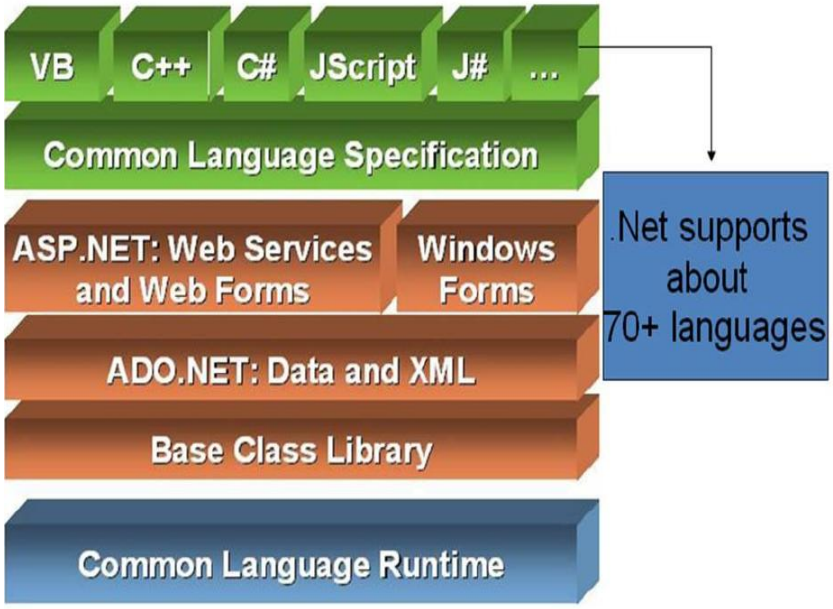
b. Write a web based application to check the entered user name and password are valid or not. Check the entered user name and password of a web form with a database table. **(08 Marks)**

c. Write a short note on sessions in ASP·NET. **(04 Marks)**

* * * * *

1. **a. Explain the components and benefits of .Net framework with the help of architecture diagram.**





CLR: Provides run time environment to run the code and provide various services to develop the application.

CTS: Specify certain guidelines for declaring using and managing types at runtime.

Base Class library: Reusable types. Classes, interfaces, value types helps in speeding-up application development process.

CLS: Common Language Specification.

Windows forms: is the graphical representation of any windows displayed in an application.

Web application: Uses ASP.NET to build application.

ADO.NET: Provides functionality for database communication.

Programming Languages: C#, VB, J#,VC++ and more are supported in .NET environment.

Benefits of .NET framework:

Consistent Programming model
Cross-platform support
Language Interoperability
Automatic management of resources
Ease of Deployment

b. What is an assembly? Explain each component of assembly.

In the .NET framework, an assembly is a compiled code library for use in deployment, Versioning and security.

There are two types: process assemblies (EXE) and library assemblies (DLL).
A process assembly represents a process which will use classes defined in library assemblies.

.NET assemblies contain code in CIL, which is usually generated from a CLI language, and then compiled into machine language at runtime by the CLR just-in-time compiler.

An assembly can consist of one or more files.

Code files are called modules. An assembly can contain more than one code module and since it is possible to use different languages to create code modules it is technically possible to use several different languages to create an assembly.

Visual Studio however does not support using different languages in one assembly.

2. a. **Explain the different types in C#**

i) Value Type:A Value Type stores its contents in memory allocated on the stack. When you created a Value Type, a single space in memory is allocated to store the value and that variable directly holds a value. If you assign it to another variable, the value is copied directly and both variables work independently. Predefined datatypes, structures, enums are also value types, and work in the same way. Value types can be created at compile

time and Stored in stack memory, because of this, Garbage collector can't access the stack.

e.g. int x = 10;
Here the value 10 is stored in an area of memory called the stack.Reference Type:

Reference Types are used by a reference which holds a reference (address) to the object but not the object itself. Because reference types represent the address of the variable rather than the data itself, assigning a reference variable to another doesn't copy the data. Instead it creates a second copy of the reference, which refers to the same location of the heap as the original value. Reference Type variables are stored in a different area of memory called the heap. This means that when a reference type variable is no longer used, it can be marked for garbage collection. Examples of reference types are Classes, Objects, Arrays, Indexers, Interfaces etc.
e.g. int[] iArray = new int[20];

b. What are different types of type conversions supported by C#.

Type conversion is converting one type of data to another type. It is also known as Type Casting. In C#, type casting has two forms:
Implicit type conversion - These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.
Explicit type conversion - These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

c. Describe the significance of 'is' and 'as' operator.

is and as operator

The is operator in C# is used to check the object type and it returns a bool value: **true** if the object is the same type and **false** if not.

```
namespace IsAndAsOperators
{
 // Sample Student Class
class Student
  {
 public int stuNo { get; set; }
public string Name { get; set; }
public int Age { get; set; }

 }
// Sample Employee Class
class Employee
{
public int EmpNo { get; set; }
public string Name { get; set; }
```

```
        public int Age { get; set; }
        public double Salary { get; set; }

        }
        class Program
        {

         static void Main(string[] args)
        {
        Student stuObj = new Student();
        stuObj.stuNo = 1;
        stuObj.Name = "Siva";
        stuObj.Age = 15;

        Employee EMPobj=new Employee();
        EMPobj.EmpNo=20;
        EMPobj.Name="Rajesh";
        EMPobj.Salary=100000;
        EMPobj.Age=25;

        // Is operator

         // Check Employee EMPobj is Student Type

         bool isStudent = (EMPobj is Student);
        System.Console.WriteLine("Empobj is a Student ?: {0}", isStudent.ToString());

         // Check Student stiObj is Student Typoe
        isStudent = (stuObj is Student);
        System.Console.WriteLine("Stuobj is a Student ?: {0}", isStudent.ToString());

         stuObj = null;
        // Check  null object Type
         isStudent = (stuObj is Student);
        System.Console.WriteLine("Stuobj(null) is a Student ?: {0}", isStudent.ToString());
        System.Console.ReadLine();
    }
}
```

3. **a. Explain different method parameter modifier with suitable examples**.

   By using the `params` keyword, we can specify a method parameter that takes a variable number of arguments.
   We can send a comma-separated list of arguments of the type specified in the parameter declaration or an array of arguments of the specified type. You also can send no arguments. If you send no arguments, the length of the `params` list is zero.

No additional parameters are permitted after the `params` keyword in a method declaration, and only one `params` keyword is permitted in a method declaration.

```
    public class MyClass
    {
    public static void UseParams(params int[] list)
    {
        for (int i = 0; i < list.Length; i++)
        {
            Console.Write(list[i] + " ");
        }
        Console.WriteLine();
    }

    public static void UseParams2(params object[] list)
    {
        for (int i = 0; i < list.Length; i++)
        {
            Console.Write(list[i] + " ");
        }
        Console.WriteLine();
    }

    static void Main()
    {

        UseParams(1, 2, 3, 4);
        UseParams2(1, 'a', "test");

        UseParams2();


        int[] myIntArray = { 5, 6, 7, 8, 9 };
        UseParams(myIntArray);

        object[] myObjArray = { 2, 'b', "test", "again" };
        UseParams2(myObjArray);


        UseParams2(myIntArray);
    }
}
```

b. Write short notes on the following:

i) Partial classes and methods

It is possible to split the definition of a class or a struct, an interface or a method over two or more source files. Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

Partial Classes

There are several situations when splitting a class definition is desirable:

- When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
- When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.
- To split a class definition, use the partial keyword modifier, as shown here:

```
public partial class CoOrds
{
    private int x;
    private int y;

    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

public partial class CoOrds
{
    public void PrintCoOrds()
    {
        Console.WriteLine("CoOrds: {0},{1}", x, y);
    }

}

class TestCoOrds
{
    static void Main()
    {
        CoOrds myCoOrds = new CoOrds(10, 15);
        myCoOrds.PrintCoOrds();

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
```

Partial Methods:

Partial methods enable the implementer of one part of a class to define a method, similar to an event. The implementer of the other part of the class can decide whether to implement the method or not. If the method is not implemented, then the compiler removes the method signature and all calls to the method. The calls to the method, including any results that would occur from evaluation of arguments in the calls,

have no effect at run time. Therefore, any code in the partial class can freely use a partial method, even if the implementation is not supplied. No compile-time or run-time errors will result if the method is called but not implemented.

```
// Definition in file1.cs
partial void onNameChanged();

// Implementation in file2.cs
partial void onNameChanged()
{
  // method body
}
```

ii) Indexers

Declaration of behavior of an indexer is to some extent similar to a property. similar to the properties, you use get and set accessors for defining an indexer. However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instance. In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part.

Defining a property involves providing a property name. Indexers are not defined with names, but with the this keyword, which refers to the object instance. The following example demonstrates the concept:

```
using System;
namespace IndexerApplication
{
   class IndexedNames
   {
      private string[] namelist = new string[size];
      static public int size = 10;
      public IndexedNames()
      {
         for (int i = 0; i < size; i++)
         namelist[i] = "N. A.";
      }

      public string this[int index]
      {
         get
         {
            string tmp;

            if( index >= 0 && index <= size-1 )
            {
               tmp = namelist[index];
            }
            else
```

```
            {
               tmp = "";
            }

            return ( tmp );
         }
         set
         {
            if( index >= 0 && index <= size-1 )
            {
               namelist[index] = value;
            }
         }
      }

      static void Main(string[] args)
      {
         IndexedNames names = new IndexedNames();
         names[0] = "Zara";
         names[1] = "Riz";
         names[2] = "Nuha";
         names[3] = "Asif";
         names[4] = "Davinder";
         names[5] = "Sunil";
         names[6] = "Rubic";
         for ( int i = 0; i < IndexedNames.size; i++ )
         {
            Console.WriteLine(names[i]);
         }

         Console.ReadKey();
      }
   }
}
```

4a. How method overriding is different from method overloading? Illustrate with examples.

What is Method Overloading ?

Creating a multiple methods in a class with same name but different parameters and types is called as method overloading.method overloading is the example of Compile time polymorphism which is done at compile time.

Method overloading can be achieved by using following things :

- By changing the number of parameters used.
- By changing the order of parameters.
- By using different data types for the parameters.

What is Method overriding ?

Creating the method in a derived class with same name, same parameters and same return type as in base class is called as method overriding.
Method overriding is the example of run time polymorphism,
Some Key Points of Method overriding

- Method overriding is only possible in derived class not within the same class where the method is declared.
- Only those methods are overrides in the derived class which is declared in the base class with the help of virtual keyword or abstract keyword.

```
namespace ConsoleApplication1
{
   class Shape
   {
      //Method to be overridden in derive class
      public virtual void Draw()
      {
      }
   }
   class Ractangel : Shape
   {
      public override void Draw()
      {
         Console.WriteLine("Rectangle Drawn ");
      }
   }
   class Circle : Shape
   {
      public override void Draw()
      {
         Console.WriteLine("Circle Drawn ");
      }
   }
```

```
    class Program
    {
        static void Main(string[] args)
        {
            Shape[] s = new Shape[2];
            /* creating Array with Different types of Objects */
            s[0] = new Circle();
            s[1] = new Ractangel();
            Console.WriteLine("\n\nRuntime polymorphism test\n\n");
            for (int i = 0; i < 2; i++)
            {
                s[i].Draw();
            }

            Console.ReadKey();
        }
    }
}
```

b. Explain sealed classes and sealed methods.

Sealed Class

Sealed class is used to define the inheritance level of a class.

The sealed modifier is used to prevent derivation from a class. An error occurs if a sealed class is specified as the base class of another class.

Some points to remember:

1. A class, which restricts inheritance for security reason is declared, sealed class.
2. Sealed class is the last class in the hierarchy.
3. Sealed class can be a derived class but can't be a base class.
4. A sealed class cannot also be an abstract class. Because abstract class has to provide functionality and here we are
   restricting it to inherit.

Practical demonstration of sealed class

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

namespace  sealed_class
{
    class  Program
    {
```

```
    public sealed class BaseClass
    {
      public void Display()
    {
      Console.WriteLine("This is a sealed class which can;t be further inherited");
    }
  }

    public class Derived : BaseClass
    {
    }

    static void Main(string[] args)
    {
      BaseClass obj = new BaseClass();

      obj.Display();

      Console.ReadLine();
    }
  }
}
```

Sealed Methods

Sealed method is used to define the overriding level of a virtual method.

Sealed keyword is always used with override keyword.

Practical demonstration of sealed method

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

namespace sealed_method
{
  class Program
  {
    public class BaseClass
    {

      public virtual void Display()
      {
        Console.WriteLine("Virtual method");
      }
    }
```

```
    public class DerivedClass : BaseClass
    {
        public override sealed void Display()
        {
            Console.WriteLine("Sealed method");
        }
    }



    static void Main(string[] args)
    {

        DerivedClass ob1 = new DerivedClass();
        ob1.Display();

        Console.ReadLine();
    }
  }
}
```

c. How would you enforce encapsulation using accessor and mutators?

Encapsulation provides a way to protect data from accidental corruption. Rather than defining the data in the form of public, we can declare those fields as private. The Private data are manipulated indirectly by two ways. Let us see some example programs in C# to demonstrate Encapsulation by those two methods. The first method is using a pair of conventional accessor and mutator methods. Another one method is using a named property. Whatever be the method our aim is to use the data with out any damage or change.

ENCAPSULATION USING ACCESSORS AND MUTATORS:

Let us see an example of Department class. To manipulate the data in that class (String departname) we define an accessor (get method) and mutator (set method).

```
using system;
public class Department
{
private string departname;
.....
// Accessor.
public string GetDepartname()
{
return departname;
}
// Mutator.
```

```
public void SetDepartname( string a)
{
departname=a;
}
}
```

Like the above way we can protect the private data from the outside world. Here we use two separate methods to assign and get the required data.

```
public static int Main(string[] args)
{
Department d = new Department();
d.SetDepartname("ELECTRONICS");
Console.WriteLine("The Department is :"+d.GetDepartname());
return 0;
}
```

In the above example we can't access the private data departname from an object instance. We manipulate the data only using those two methods.

5**a.How delegates are used in C#? Discuss single cast and multicast delegates?**

Delegate: A delegate is a special type of object that contains the details of a method rather than data.

In C# delegate is a class type object, which is used to invoke the method that has been encapsulated into it at the time of its creation. A delegate can be used to hold the reference to a method of any class.

Delegate contains 3 important piece of information

1. The name of the method on which it makes calls
2. Argument of this method
3. Return value of this method

Creating and using delegate:

1. Declaring a delegate
2. Defining delegate methods
3. Creating delegate objects
4. Invoking delegate objects

Declaring a delegate

Access-modifier delegate return-type delegate-name (parameter-list);

Public delegate void compute(int x, int y);

Defining Delegate Methods

Public static void Add(int a, int b)
{
Console.WriteLine("Sum={0}",a +b);
}

Creating Delegate Objects:

Delegate-name object-name=new delegate-name(expression);

Invoking Delegate object

Delegate-object(argument-list)

Cmp1(30,20);

```
using System;
public delegate double Conversion(double from);
class DelegateDemo
{
    public static double FeetToInches(double feet)
    {
        return feet * 12;
    }

           static void Main()
           {
               Conversion doConversion = new Conversion(FeetToInches);
               Console.Write("Enter Feet:  ");
               double feet = Double.Parse(Console.ReadLine());
               double inches = doConversion(feet);
               Console.WriteLine("\n{0} Feet = {1} Inches.\n", feet, inches);
               Console.ReadLine();
           }
       }
```

Multicasting with delegates:

A delegate object can hold reference of and invoke multiple methods.

```
using System;
delegate void CustomDel(string s);
class TestClass
{
    static void Hello(string s)
    {
        System.Console.WriteLine("  Hello, {0}!", s);
    }
    static void Goodbye(string s)
    {
        System.Console.WriteLine("  Goodbye, {0}!", s);
```

```
    }
    static void Main()
    {
        CustomDel hiDel, byeDel, multiDel, multiMinusHiDel;
        hiDel = Hello;
        byeDel = Goodbye;
        multiDel = hiDel + byeDel;
        multiMinusHiDel = multiDel - hiDel;
        Console.WriteLine("Invoking delegate hiDel:");
        hiDel("A");
        Console.WriteLine("Invoking delegate byeDel:");
        byeDel("B");
        Console.WriteLine("Invoking delegate multiDel:");
        multiDel("C");
        Console.WriteLine("Invoking delegate multiMinusHiDel:");
        multiMinusHiDel("D");
        Console.ReadLine();
    }
}
```

b. "**Catching on exceptions programmatically is good and necessary mechanism" justify with suitable examples.**

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

namespace GMT_TCF
{
    class Program
    {

        static void Main(string[] args)
        {
            Console.WriteLine(" Enter the dividend");
            int m= Convert.ToInt32(Console.ReadLine());

            Console.WriteLine(" Enter the divisor");
            int n = Convert.ToInt32(Console.ReadLine());

            try
            {
                int k = m / n;
                Console.WriteLine("Output is:" + k.ToString());
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Exception Caught:" + e.Message);
            }
            finally
            {
                 Console.ReadLine();
            }
        }
```

```
    }
}
```

6a. Discuss panel control with respect to windows form.

The Panel control is similar to the GroupBox control; however, only the Panel control can have scroll bars, and only the GroupBox control displays a caption.

How to use Panel Control
Drag and drop Panel control from toolbox on the window Form.
Collection of control can be placed in side Panel.

Transparent Panel

First set BackColor of Panel suppose you set red then set Form's TransparencyKey property to the same color as Panel's background color –red in this case.
Example:

```
private void frmPanel_Load(object sender, EventArgs e)
{
  //change back color of Panel
  panel1.BackColor = Color.Red;
  //set Form's TransparencyKey to the same color as Panel's back color
  this.TransparencyKey = Color.Red;
}
```

Now panel will be transparent when application run.

Panel Properties

BackColor: Panel BackColor can be changed through BackColor property.
Example:

```
private void frmPanel_Load(object sender, EventArgs e)
{
   //change back color of Panel
   panel1.BackColor = Color.CadetBlue;
}
```

BorderStyle: Get or set BorderStyle of Panel.
Example:

```
private void frmPanel_Load(object sender, EventArgs e)
{
      //Set Border style of Panel
      panel1.BorderStyle = BorderStyle.Fixed3D;
```

}

b. Explain the implementation of combobox in C#

C# controls are located in the Toolbox of the development environment, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions. A ComboBox displays a text box combined with a ListBox, which enables the user to select items from the list or enter a new value.

How add a item to combobox

```
comboBox1.Items.Add("Sunday");
comboBox1.Items.Add("Monday");
comboBox1.Items.Add("Tuesday");
ComboBox SelectedItem
```

retrieve the displayed item to a string variable ,

```
string var;
var = comboBox1.Text;
                    Or
var item = this.comboBox1.GetItemText(this.comboBox1.SelectedItem);
MessageBox.Show(item);
```

How to remove an item from ComboBox

```
comboBox1.Items.RemoveAt(1);
```

The above code will remove the second item from the combobox.

```
comboBox1.Items.Remove("Friday");
```

c.Write a program for addition of two numbers, by accepting numbers from text boxes and display the result in the form using button lick event.

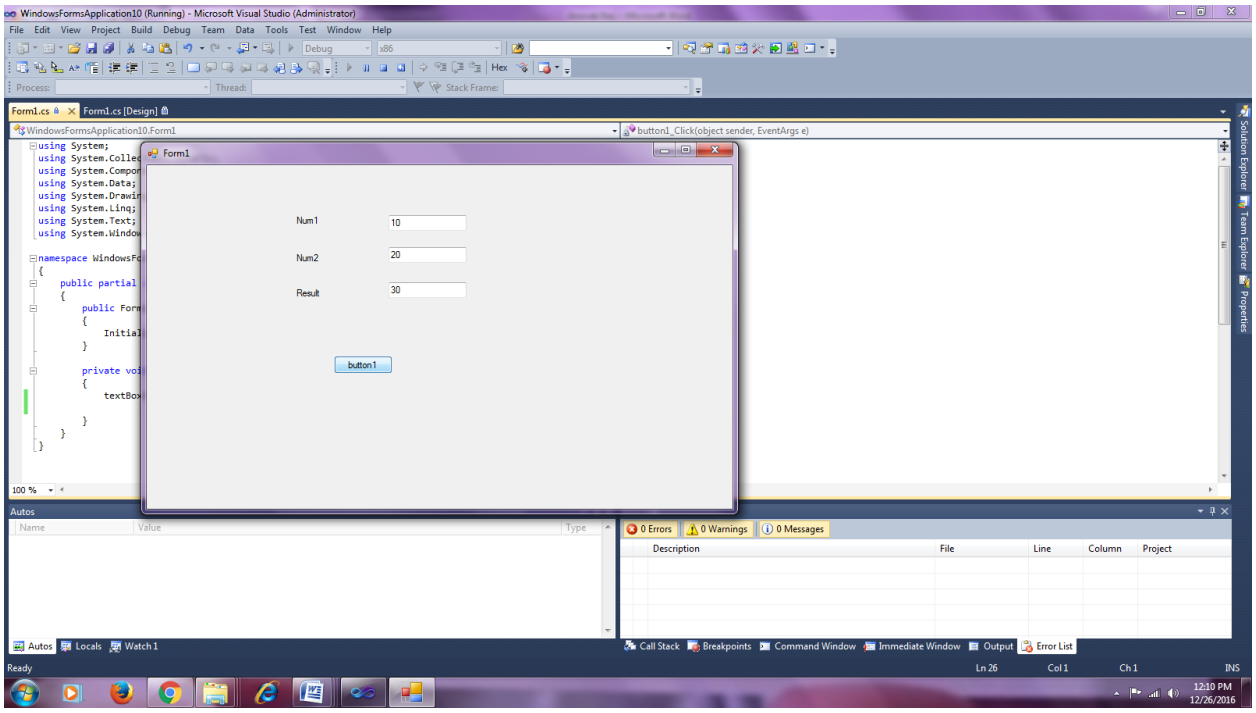
```
using System;

namespace WindowsFormsApplication10
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

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

```
textBox3.Text = (Convert.ToInt32(textBox1.Text) +
Convert.ToInt32(textBox2.Text)).ToString();

        }
    }
}
```



6a.Explain the components of ADO.NET entity framework

Entity Data Model: Defines the conceptual entities that can be read in serialized form using a DataReader

Entity SQL: Defines a common SQL based query language that is extended to express queries in terms of EDM concepts.

Entity Client: Provides a gateway for the queries of the entity level queries, which is queried through a common Entity SQL language.

Object Services: Allows us to

b.**How data adapter is used to build database applications?**

Data adapters are an integral part of ADO.NET managed providers, which are the set of objects used to communicate between a datasource and a dataset. (In addition to adapters, managed providers include connection objects, data reader objects, and command objects.)

To create a data adapter manually

1. Make sure a connection object is available to the form or component that you are working with. For details about adding a standalone connection, see Establishing a Connection.
2. From the Data tab of the Toolbox, drag an OleDbDataAdapter, SqlDataAdapter, OdbcDataAdapter, OracleDataAdapter object onto the design surface.
   The designer adds an instance of the adapter to the form or component and launches the Data Adapter Configuration Wizard.
3. Close the wizard.

c.Discuss the components of dataset.

Datatable: Consists of DataRow and Data Column and stores data in the table row format.

DataView: Represents a customized view of DataTable for sorting, filtering, searching, editing and navigating.

DataColumn: Consists of a number of columns that comprise a Datatable. A DataColumn is the essential building block of the DataTable.

DataRow: Represents a row in the DataTable.

DataRelation: Allows us to specify relations between various table**.**

5a.Explain different validation control with suitable examples supported by ASP.NET.

Validation Controls in ASP.NET

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET

1. RequiredFieldValidation Control
2. CompareValidator Control
3. RangeValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary

The below table describes the controls and their work:

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page

All validation controls are rendered in form as (label are referred as on client by server)

Important points for validation controls

- ControlToValidate property is mandatory to all validate controls.
- One validation control will validate only one input control but multiple validate control can be assigned to a input control.

A mutator, in the context of C#, is a method, with a public level of accessibility, used to modify and control the value of a private member variable of a class. The mutator is used to assign a new value to the private field of a type. It forms a tool to implement encapsulation by only controlling access to the internal field values that must be modified.

The benefits of using a mutator include:

- Prevents the user from directly accessing the private data of an object instance and allows access only through public methods to prevent data corruption.
- Provides flexibility in modifying the internal representation of the fields of an object that represents the internal state without breaking the interface used by the object's clients.
- Ability to include additional processing logic like validation of a values set, triggering of events, etc., during the modification of the field in the mutator.
- Provides the synchronization that is necessary for multithreading scenarios.
- Includes a provision to override the mutator declared in a base class with the code in the derived class.

b.Write a web based application to check the entered username and password are valid or not.Check the entered username and password of a web form with a database table.

```
private void btnLogin_Click(object sender, EventArgs e)
    {
       if (string.IsNullOrEmpty(this.txtUsername.Text) | string.IsNullOrEmpty(this.txtPassword.Text))
       {
          MessageBox.Show("provide User Name and Password");
       }

       if (string.IsNullOrEmpty(cboUsertype.Text))
       {
          MessageBox.Show("Select User Type");
       }

       SqlConnection conn = new SqlConnection();
       conn.ConnectionString = "Data Source=pc101;Initial Catalog=SMS;User ID=sa;Password=mike";
       conn.Open();
       string UserName = txtUsername.Text;
       string Password = txtPassword.Text;
       string UserType = cboUsertype.Text;

       SqlCommand cmd = new SqlCommand("SELECT * FROM tbluser WHERE username = '" +
txtUsername.Text + "' and usertype = '" + cboUsertype.Text + "' and mypassword = '" + txtPassword.Text
+ "'", conn);

       SqlDataAdapter da = new SqlDataAdapter(cmd);
       DataTable dt = new DataTable();
       da.Fill(dt);

       System.Data.SqlClient.SqlDataReader dr = null;
       dr = cmd.ExecuteReader();

       if (dr.Read())
       {
          SqlConnection con = new
SqlConnection(ConfigurationSettings.AppSettings["ConnectionString"]);
          con.ConnectionString = "Data Source=pc101;Initial Catalog=SMS;User ID=sa;Password=mike";
          con.Open();

          if (this.cboUsertype.Text == dr["UserType"].ToString() & this.txtUsername.Text ==
dr["UserName"].ToString() & this.txtPassword.Text == dr["mypassword"].ToString() &
this.cboUsertype.Text == "Data Entry Clerk")
           {
             MessageBox.Show("*** Login Successful ***");
             frmMain f = new frmMain();
             f.Show();
```

```
            // f.CreateUserAccountToolStripMenuItem.Enabled = false;
             this.Hide();
           }

            else if (this.cboUsertype.Text == dr["UserType"].ToString() & this.txtUsername.Text ==
dr["UserName"].ToString() & this.txtPassword.Text == dr["mypassword"].ToString())
           {
             MessageBox.Show("*** Login Successful ***");
             frmMain g = new frmMain();
             g.Show();
             this.Hide();
           }

           else
           {
             MessageBox.Show("Invalid UserName or Password", "Login", MessageBoxButtons.OK,
MessageBoxIcon.Information);
             MessageBox.Show("Access Denied!!");

           }
       }
   }
```

c. Write a short note on sessions in ASP.NET.

Session state include the following:

- Application state, which stores variables that can be accessed by all users of an ASP.NET application.
- Profile properties, which persists user values in a data store without expiring them.
- ASP.NET caching, which stores values in memory that is available to all ASP.NET applications.
- View state, which persists values in a page.
- Cookies.
- The query string and fields on an HTML form that are available from an HTTP request.

Session variables are stored in a SessionStateItemCollection object that is exposed through the HttpContext.Session property. In an ASP.NET page, the current session variables are exposed through the Session property of the Page object.

The collection of session variables is indexed by the name of the variable or by an integer index. Session variables are created by referring to the session variable by name. You do not have to declare a session variable or explicitly add it to the collection. The following example shows how to create session variables in an ASP.NET page for the first and last name of a user, and set them to values retrieved from TextBox controls.