Hall Ticket Number:

III/IV B.Tech (Regular) DEGREE EXAMINATION

July/August, 2023 — **Information Technology**

Sixth Semester — **Middleware Technologies**

Time: Three Hours — **Maximum:**70 Marks

Answer question 1 compulsory. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56 Marks)**

			CO	BL	M
1	a)	What is MSIL code?	CO1	L1	1M
	b)	Write syntax of a method to extract substring from a string.	CO1	L2	1M
	c)	What is the use of ref parameter in C#.	CO1	L1	1M
	d)	What is the use of a static method in C#.	CO1	L1	1M
	e)	What is inheritance in C#?	CO2	L1	1M
	f)	What is method overriding in C#?	CO2	L1	1M
	g)	What is a finally key word in exception handling in C#.	CO2	L1	1M
	h)	List three examples of ASP.NET web controls.	CO3	L1	1M
	i)	What is Session state in ASP.NET?	CO3	L1	1M
	j)	Write syntax of RegularExpressionValidation control.	CO3	L2	1M
	k)	What is data binding in ASP.NET?	CO4	L1	1M
	l)	What is the purpose of the DataReader class in ADO.NET?	CO4	L1	1M
	m)	List any four GridView features.	CO4	L1	1M
	n)	What is the difference between DetailsView and FormView controls in ASP.NET	CO4	L1	1M
		Unit-I			
2	a)	Discuss about .NET Frame work with neat diagram.	CO1	L2	7M
	b)	Write a C# program to demonstrate String class in C#	CO1	L3	7M
		(OR)			
3	a)	Write a C# program that takes an array of integers as input and finds the sum and average of all the numbers in the array. Analyze the program and discuss its complexity.	CO1	L3	7M
	b)	Write a program to demonstrate static key word in C#.	CO1	L3	7M
		Unit-II			
4	a)	Explain the concept of inheritance in C# and discuss how it promotes code reuse and extensibility.	CO2	L2	7M
	b)	Discuss about interfaces in C# with examples.	CO2	L2	7M
		(OR)			
5	a)	Discuss about exception handling in C# with examples.	CO2	L2	7M
	b)	How does the use of virtual methods impact the maintainability and extensibility of software systems in C#? Provide examples to support your answer.	CO2	L2	7M
		Unit-III			
6	a)	Explain about ASP.NET Web Server Controls.	CO3	L2	7M
	b)	Explain validation controls used in ASP.NET web applications	CO3	L2	7M
		(OR)			
7	a)	Create an ASP.NET application for converting currency.	CO3	L3	7M
	b)	Discuss about State management in ASP.NET	CO3	L2	7M
		Unit-IV			
8	a)	Discuss about data binding in ASP.NET.	CO4	L2	7M
	b)	Explain about data controls in ASP.NET.	CO4	L3	7M
		(OR)			
9	a)	Describe the purpose and functionality of ADO.NET framework. Apply your knowledge to create a console application that retrieves data from a SQL Server database using ADO.NET's DataReader class. Analyze the benefits and limitations of using DataReader over other ADO.NET components.	CO4	L3	7M
	b)	Create an ASP.NET web application for accessing SQL Server DB for performing CRUD operations.	CO4	L3	7M

Scheme of Evaluation

1 a) What is MSIL code? **CO1 L1 1M**

Ans) Microsoft Intermediate Language (MSIL) is an intermediate language used as the output of a number of compilers (C#, VB, .NET, and so forth) in .NET environment, which is a CPU-independent set of instructions that can be efficiently converted to native code.

b) Write syntax of a method to extract substring from a string. **CO1 L2 1M**

Ans) public string Substring(int startIndex)
public string Substring(int startIndex, int length)

c) What is the use of ref parameter in C#. **CO1 L1 1M**

Ans) The ref parameter modifier causes C# to create a call-by-reference, rather than a call-by-value.

d) What is the use of a static method in C#. **CO1 L1 1M**

Ans) A static method in C# is a method that keeps only one copy of the method at the Type level, not the object level. We can call a static method without using object of its class.

e) What is inheritance in C#? **CO2 L1 1M**

Ans) In C#, inheritance allows us to create a new class from an existing class. It is a key feature of Object-Oriented Programming (OOP).

f) What is method overriding in C#? **CO2 L1 1M**

Ans) Creating a method in the derived class with the same signature as a method in the base class is called as method overriding. Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.

g) What is a finally key word in exception handling in C#. **CO2 L1 1M**

Ans) The finally block will execute when the try/catch block leaves the execution, no matter what condition cause it. It always executes whether the try block terminates normally or terminates due to an exception.

h) List three examples of ASP.NET web controls. **CO3 L1 1M**

Ans) GridView, CompareValidator, CheckBoxList

i) What is Session state in ASP.NET? **CO3 L1 1M**

Ans) It is maintained at session-level and data can be accessed across all pages in the web application. The information is stored within the server and can be accessed by any person that has access to the server where the information is stored.

j) Write syntax of RegularExpressionValidation control. **CO3 L2 1M**

Ans)

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
runat="server"ControlToValidate="username"  ErrorMessage="Please enter valid email"
foreColor="Red"ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

k) What is data binding in ASP.NET? **CO4 L1 1M**

Ans) **Data binding,** in the context of .NET, is the method by which controls on a user interface (UI) of a client application are configured to fetch from, or update data into, a data source, such as a database or XML document.

l) What is the purpose of the DataReader class in ADO.NET? **CO4 L1 1M**

Ans) The DataReader Object in ADO.NET provides a stream-based, forward-only, and read-only retrieval of query results from data sources. It is specifically designed for efficiently accessing and processing large result sets without the need to load the entire set into memory.

m) List any four GridView features. **CO4 L1 1M**

Ans) Improved data source binding capabilities

- Tabular rendering – displays data as a table
- Built-in sorting capability
- Built-in select, edit and delete capabilities
- Built-in paging capability
- Built-in row selection capability
- Multiple key fields

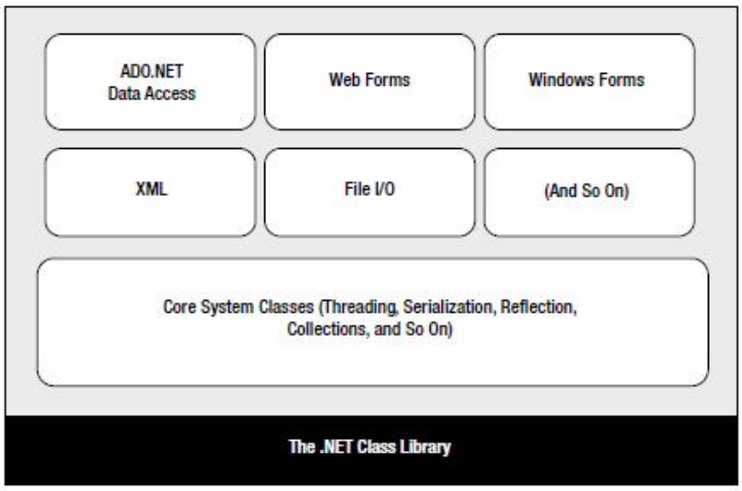
- Programmatic access to the GridView object model to dynamically set properties, handle events and so on
- Richer design-time capabilities

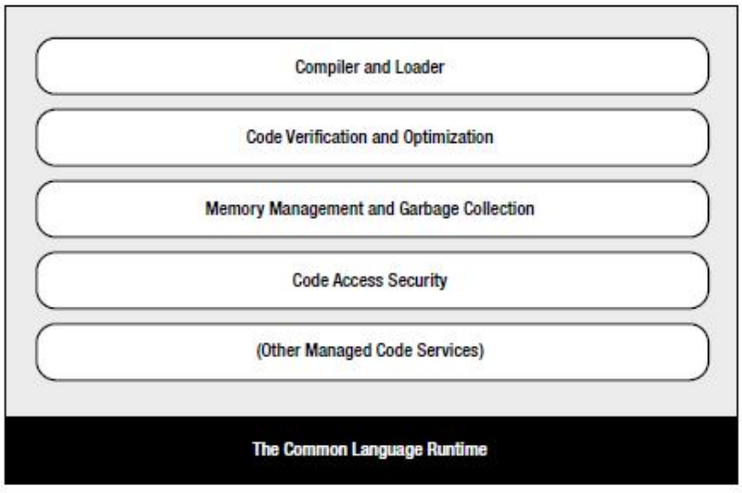
n) What is the difference between DetailsView and FormView controls in ASP.NET **CO4 L1 1M**

Ans) The DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control. The FormView control is used to display a single record from a data source. It is similar to the DetailsView control, except it displays user-defined templates instead of row fields.

Unit-I

2 a) Discuss about .NET Frame work with neat diagram. **CO1 L2 7M**
Ans) **diagram → 3 Marks**





The .NET Framework is really a cluster of several technologies: **explanation → 4 Marks**

The .NET languages: These include Visual Basic, C#, F#, and C++, although third-party developers have created hundreds more.

The Common Language Runtime (CLR): This is the engine that executes all .NET programs and provides automatic services for these applications, such as security checking, memory management, and optimization.

With CLR, the following are possible:

a. **Deep language integration:** VB and C#, like all .NET languages, compile to IL. In other words, the CLR makes no distinction between different languages—in fact, it has no way of knowing what language was used to create an executable. This is far more than mere language compatibility; it's language integration.

b. **Side-by-side execution:** The CLR also has the ability to load more than one version of a component at a time. In other words, you can update a component many times, and the correct version will be loaded and used for each application. As a side effect, multiple versions of the .NET Framework can be installed, meaning that you're able to upgrade to new versions of ASP.NET without replacing the current version or needing to rewrite your applications.

c. **Fewer errors:** Whole categories of errors are impossible with the CLR. For example, the CLR prevents many memory mistakes that are possible with lower-level languages such as C++.

The .NET Framework class library: The class library collects thousands of pieces of prebuilt functionality that you can "snap in" to your applications. These features are sometimes organized into technology sets, such as ADO.NET (the technology for creating database applications) and Windows Presentation Foundation (WPF, the technology for creating desktop user interfaces).

ASP.NET: This is the engine that hosts the web applications you create with .NET, and supports almost any feature from the .NET Framework class library. ASP.NET also includes a set of web-specific services, such as secure authentication and data storage.

Visual Studio: This optional development tool contains a rich set of productivity and debugging features. Visual Studio includes the complete .NET Framework, so you won't need to download it separately.

2. b) Write a C# program to demonstrate String class in C# **CO1 L3 7M**

Ans

Code → 6M

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

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         // create string
         string str = "C# Programming";
         string str1 = "C# language";
         string str2 = "C# Programming";
         string str3 = "C#";
         Console.WriteLine("string: " + str);

         // get length of str
         int length = str.Length;
         Console.WriteLine("Length: " + length);

         str1 = "C#";
         Console.WriteLine("string str1: " + str1);
          str2 = "Programming";
         Console.WriteLine("string str2: " + str2);

         // join two strings
         string joinedString = string.Concat(str1, str2);
         Console.WriteLine("Joined string: " + joinedString);

         // compare str1 and str2
         Boolean result1 = str1.Equals(str2);
         Console.WriteLine("string str1 and str2 are equal: " + result1);

         //compare str1 and str3
         Boolean result2 = str1.Equals(str3);
         Console.WriteLine("string str1 and str3 are equal: " + result2);

         //string copy -- str1 to str2
         str2 = String.Copy(str1);
         Console.WriteLine("Strings after Copy: str1 = " + "'{0}' and str2='{1}'", str1, str2);
```

```
        //Replace method
        const string s = "Sun Rises in the West";
        Console.WriteLine("Sentence Before Replacing : {0} ", s);
        string s1 = s.Replace("West", "East");
        Console.WriteLine("Sentence After Replacing : {0} ", s1);
        Console.ReadLine();
      }
   }
}
```

Output: **output → 1M**

```
string: C# Programming
Length: 14
string str1: C#
string str2: Programming
Joined string: C#Programming
string str1 and str2 are equal: False
string str1 and str3 are equal: True
Strings after Copy: str1 = 'C#' and str2='C#'
Sentence Before Replacing : Sun Rises in the West
Sentence After Replacing : Sun Rises in the East
```

(OR)

3 a) Write a C# program that takes an array of integers as input and finds the sum and average of all the numbers in the array. Analyze the program and discuss its complexity. CO1 L3 7M

Ans) **→7M**

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

namespace arrayDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] arr = { 1, 4, 5 };
         Console.WriteLine("finding sum, average of elements in the array");
         sumAvgArr(arr);
      }
      static void sumAvgArr(int[] arr)
      {
         int sum=0; float avg = 0.0f;
         for (int i = 0; i < arr.Length; i++) sum += arr[i];
         avg = (float)sum / arr.Length;
         Console.WriteLine("sum and average of array elements are: {0} {1}", sum,avg);
         Console.ReadLine();
      }
   }
}
```

3. b) Write a program to demonstrate static key word in C#. CO1 L3 7M

Ans)

static class, static method, static var usage →7M

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

namespace staticDemo
{
      // Creating static class
    static class Author {
   // Static data members of Author
   public static string A_name = "Ankita";
   public static string L_name = "CSharp";
```

```
    public static int T_no = 84;
    // Static method of Author
    public static void details()
    {
        Console.WriteLine("The details of Author is:");
    }

  }  //Author class
    class StaticMV
    {
        // A static variable.
        public static int Val = 100;
        // A static method.
        public static int ValDiv2()
        {
            return Val / 2;
        }
    } //staticMV class

    public class Program
    {
        static void Main(string[] args)
        {
            // Calling static method of Author
            Author.details();
            // Accessing the static data members of Author
            Console.WriteLine("Author name : {0} ", Author.A_name);
            Console.WriteLine("Language : {0} ", Author.L_name);
            Console.WriteLine("Total number of articles : {0} ", Author.T_no);
            //accessing non static class (Program) static members
            Console.WriteLine("Initial value of StaticDemo.Val is " + StaticMV.Val);
            StaticMV.Val = 8;
            Console.WriteLine("StaticDemo.Val is " + StaticMV.Val);
            Console.WriteLine("StaticDemo.ValDiv2(): " +
            StaticMV.ValDiv2());
            Console.Read();
        }
    }
}
```

Output:

```
The details of Author is:
Author name : Ankita
Language : CSharp
Total number of articles : 84
Initial value of StaticDemo.Val is 100
StaticDemo.Val is 8
StaticDemo.ValDiv2(): 4
```

Unit-II

4 a) Explain the concept of inheritance in C# and discuss how it promotes code reuse and extensibility.

CO2 L2 7M

Ans) **7 points → 7M**

C# supports inheritance by allowing one class to incorporate another class into its declaration. This is done by specifying a base class when a derived class is declared.

The general form of a class declaration that inherits a base class is shown here:

```
class derived-class-name : base-class-name {
// body of class
}
```

You can specify only one base class for any derived class that you create. C# does not support the inheritance of multiple base classes into a single derived class. (This differs from C++, in which you can inherit multiple base classes. Be aware of this when converting C++ code to C#.) You can, however, create a hierarchy of inheritance in which a derived class becomes a base class of another derived class. (Of course, no class can be a base class of itself, either directly or indirectly.)

In all cases, a derived class inherits all of the members of its base class. This includes instance variables, methods, properties, and indexers. A major advantage of inheritance is that once you have created a base class that defines the attributes common to a set of objects, it can be used to create any number of more specific derived classes. Each derived class can precisely tailor its own classification.

Member Access: Following table shows the class member accessibility in different places in the program.

	public	protected	internal	protected internal	private	private protected
Entire program	Yes	No	No	No	No	No
Containing class	Yes	Yes	Yes	Yes	Yes	Yes
Current assembly	Yes	No	Yes	Yes	No	No
Derived types	Yes	Yes	No	Yes	No	No
Derived types within current assembly	Yes	Yes	Yes	Yes	No	Yes

The constructor for the base class constructs the base class portion of the object, and the constructor for the derived class constructs the derived class part. This makes sense because the base class has no knowledge of or access to any element in a derived class.

Calling Base Class Constructors:

A derived class can call a constructor defined in its base class by using an expanded form of the derived class' constructor declaration and the base keyword. The general form of this expanded declaration is shown here:

derived-constructor(parameter-list) : base(arg-list) {
// body of constructor
}

Here, arg-list specifies any arguments needed by the constructor in the base class. Notice the placement of the colon.

Inheritance and Name Hiding:

It is possible for a derived class to define a member that has the same name as a member in its base class. When this happens, the member in the base class is hidden within the derived class. While this is not technically an error in C#, the compiler will issue a warning message. This warning alerts you to the fact that a name is being hidden.

Multilevel hierarchy:

it is perfectly acceptable to use a derived class as a base class of another.

Constructors calling:

When a derived class object is created, whose constructor is executed first? The one in the derived class or the one defined by the base class? For example, given a derived class called B and a base class called A, is A's constructor called before B's, or vice versa? The answer is that in a class hierarchy, constructors are called in order of derivation, from base class to derived class

A reference variable for one class type cannot normally refer to an object of another class type.

Preventing Inheritance:

To prevent a class from being inherited, precede its declaration with **sealed**.

Example:

Here is an example of a sealed class:

```
sealed class A {
// ...
}
// The following class is illegal.
class B : A { // ERROR! Can't derive from class A
// ...
}
```

4 b) Discuss about interfaces in C# with examples. CO2 L2 7M

Ans) → 3M

An interface is defined as a syntactical contract that all the classes inheriting the interface should follow. The interface defines the '**what**' part of the syntactical contract and the deriving classes define the '**how**' part of the syntactical contract.

Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

Abstract classes to some extent serve the same purpose, however, they are mostly used when only few methods are to be declared by the base class and the deriving class implements the functionalities.

Declaring Interfaces

Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default. Following is an example of an interface declaration −

```
public interface ITransactions {
  // interface members
  void showTransaction();
  double getAmount();
}
```

Example: → 4M

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

namespace InterfaceApplication {

  public interface ITransactions {
    // interface members
    void showTransaction();
    double getAmount();
  }
  public class Transaction : ITransactions {
    private string tCode;
    private string date;
    private double amount;

    public Transaction() {
      tCode = " ";
      date = " ";
      amount = 0.0;
    }
    public Transaction(string c, string d, double a) {
      tCode = c;
      date = d;
      amount = a;
    }
    public double getAmount() {
      return amount;
    }
    public void showTransaction() {
      Console.WriteLine("Transaction: {0}", tCode);
      Console.WriteLine("Date: {0}", date);
```

```
        Console.WriteLine("Amount: {0}", getAmount());
      }
    }
    class Tester {

      static void Main(string[] args) {
        Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
        Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);

        t1.showTransaction();
        t2.showTransaction();
        Console.ReadKey();
      }
    }
}
```

(OR)

5 a) Discuss about exception handling in C# with examples. CO2 L2 7M

Ans) → 3M

An exception is defined as an event that occurs during the execution of a program that is unexpected by the program code. The actions to be performed in case of occurrence of an exception is not known to the program. In such a case, we create an exception object and call the exception handler code. The execution of an exception handler so that the program code does not crash is called exception handling. Exception handling is important because it gracefully handles an unwanted event, an exception so that the program code still makes sense to the user.

Keyword	**Definition**
try	Used to define a try block. This block holds the code that may throw an exception.
catch	Used to define a catch block. This block catches the exception thrown by the try block.
finally	Used to define the finally block. This block holds the default code.
throw	Used to throw an exception manually.

Exception Handling Using try-catch block:

The code given below shows how we can handle exceptions using the try-catch block. The code that may generate an exception is placed inside the try block. In this case, the access to the 7th element is put inside the try block. When that statement is executed an exception is generated, which is caught by the catch block. The object of the type IndexOutOfRangeException is used to display a message to the user about the exception that has occurred.

Syntax:

```
try
{
  // statements that may cause an exception
}
catch( Exception obj)
{
  // handler code
}
```

Using Multiple try-catch blocks: → 4M

In the code given below, we attempt to generate an exception in the try block and catch it in one of the multiple catch blocks. Multiple catch blocks are used when we are not sure about the exception type that may be generated, so we write different blocks to tackle any type of exception that is encountered.

The finally block is the part of the code that has to be executed irrespective of if the exception was generated or not. In the program given below the elements of the array are displayed in the finally block.

Syntax:

```
try
{
  // statements that may cause an exception
}
catch(Specific_Exception_type obj)
{
  // handler code
}
catch(Specific_Exception_type obj)
{
  // handler code
}
```

.
.
.
finally
{
//default code
}

5 b) How does the use of virtual methods impact the maintainability and extensibility of software systems in C#? Provide examples to support your answer. CO2 L2 7M

Ans) → 7M

In C#, a virtual method is a method that can be overridden in a derived class. When a method is declared as virtual in a base class, it allows a derived class to provide its own implementation of the method.

To declare a method as virtual in C#, the "virtual" keyword is used in the method declaration in the base class. For example

```
public class Animal
{
   public virtual void MakeSound()
   {
      Console.WriteLine("The animal makes a sound");
   }
}
```

In the derived class, the method can be overridden by using the "override" keyword in the method declaration. For example:

```
public class Cat : Animal
{
   public override void MakeSound()
   {
      Console.WriteLine("The cat meows");
   }
}
```

When an instance of the derived class is created and the overridden method is called, the implementation in the derived class will be executed instead of the implementation in the base class. Using virtual methods can be useful in situations where you want to provide a base implementation in a base class, but allow derived classes to modify or extend the behavior of that method.

Unit-III

6 a) Explain about ASP.NET Web Server Controls. CO3 L2 7M

Ans)

Web Server control classes are: → 7M

Control Class	Underlying HTML Element
Label	
Button	<input type="submit"> or <input type="button">
TextBox	<input type="text">, <input type="password">, or <textarea>
CheckBox	<input type="checkbox">
RadioButton	<input type="radio">
Hyperlink	<a>
LinkButton	<a> with a contained <img> tag
ImageButton	<input type="image">
Image	<img>
ListBox	<select size="X"> where X is the number of rows that are visible at once
DropDownList	<select>
CheckBoxList	A list or <table> with multiple <input type="checkbox"> tags
RadioButtonList	A list or <table> with multiple <input type="radio"> tags
BulletedList	An ordered list (numbered) or unordered list (bulleted)
Panel	
Table, TableRow, and TableCell	<table>, <tr>, and <td> or <th>

6. b) Explain validation controls used in ASP.NET web applications **CO3 L2 7M**
Ans)

Validation controls in ASP.NET verifies user input and reporting errors. Each validation control, or *validator*, has its own built-in logic. Some check for missing data, others verify that numbers fall in a predefined range, and so on. In many cases, the validation controls allow you to verify user input without writing a line of code.
The following table shows the Validation controls and their description.

Control Class	Description
RequiredFieldValidator	Validation succeeds as long as the input control doesn't contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that you specify.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression.
CustomValidator	Validation is performed by a user-defined function.

RequiredFieldValidator:
<asp:Label ID="Label1" runat="server" Text="User Name:"></asp:Label>
<asp:TextBox ID="txtUserName" runat="server"></asp:TextBox>
<**asp:RequiredFieldValidator** id="vldUserName" runat="server"
ErrorMessage="You must enter a username." ControlToValidate="txtUserName"
></asp:RequiredFieldValidator>
RangeValidator:
<asp:Label ID="Label5" runat="server" Text="Age:"></asp:Label>
<asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
<**asp:RangeValidator** id="vldAge" runat="server" ErrorMessage="This age is not between 0 and 120."
Type="Integer" MinimumValue="0" MaximumValue="120"
ControlToValidate="txtAge" />
CompareValidator:
<asp:TextBox ID="txtPassword" runat="server" TextMode="Password"></asp:TextBox>
<asp:TextBox ID="txtRetype" runat="server" TextMode="Password"></asp:TextBox>
<**asp:CompareValidator** id="vldRetype" runat="server" ErrorMessage="Your password does not
match." ControlToCompare="txtPassword" ControlToValidate="txtRetype" />
RegularExpressionValidator:
<asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator id="vldEmail" runat="server" ErrorMessage="This email is missing
the @ symbol." ValidationExpression=".+@.+" ControlToValidate="txtEmail" />
CustomValidator:
<asp:TextBox ID="txtCode" runat="server"></asp:TextBox>
<**asp:CustomValidator** id="vldCode" runat="server" ErrorMessage="Try a string that starts with 014."
ValidateEmptyText="False" ControlToValidate="txtCode" />

(OR)

7 a) Create an ASP.NET application for converting currency. **CO3 L3 7M**
Ans)
Default.aspx: **→ 4M**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
Trace="true"%>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Currency converter</title>
</head>
<body>
    <form id="form1" runat="server" enableviewstate="False">
    <div>
Convert:  
<input type = "text" ID = "US" runat = "server" enableviewstate="False" />
```

```
  U.S. dollars to Euros.

<br />
 <input type = "submit" value = "OK" ID = "convert" runat = "server"    onserverclick="Convert_ServerClick"
/>
<p style = "font-weight: bold" ID = "Result" runat = "server" > </p>
</div>
   </form>

</body>
</html>
```

Default.aspx.cs: → **3M**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text;
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {

   }
   protected void Convert_ServerClick(object sender, EventArgs e)
   {
      double USAmount = Double.Parse(US.Value);

      double euroAmount = USAmount * 0.85;
      Result.InnerText = USAmount.ToString() + " U.S. dollars = ";
      string str=Convert.ToString(euroAmount);
      Result.InnerText+=str;
      Result.InnerText += " Euros.";
   }
}
```

b) Discuss about State management in ASP.NET **CO3 L2 7M**

Ans) **Techniques → 3M**

State management is the process by which ASP.NET let the developers maintain state and page information over multiple request for the same or different pages.
There are mainly two types of state management that ASP.NET provides:
1. Client side state management
2. Server side state management
When we use client side state management, the state related information will be stored on client side. This information will travel back and forth with every request and response.

The major benefit of having this kind of state management is that we relieve the server from the burden of keeping the state related information, it saves a lot of server memory. The downside of client side state management is that it takes more bandwidth as considerable amount of data is traveling back and forth. But there is one more problem which is bigger than the bandwidth usage problem. The client side state management makes the information travel back and forth and hence this information can be intercepted by anyone in between. So there is no way we can store the sensitive information like passwords, creditcard number and payable amount on client side, we need server side state management for such things.

Server side state management, in contrast to client side, keeps all the information in user memory. The downside of this is more memory usage on server and the benefit is that users' confidential and sensitive information is secure.

Client side state management techniques

- View State
- Control State
- Hidden fields
- Cookies

· Query Strings

Server side state management techniques

· Application State

· Session State

Techniques Description → 4M

View State: ASP.NET uses this mechanism to track the values of the controls on the web page between page request for same page. We can also add custom values to view state. ASP.NET framework takes care of storing the information of controls in view state and retrieving it back from viewstate before rendering on postback.

Control State: We can disable the View State of the controls. That is why Control State is provided which can not be disabled by control users. Control states lies inside custom controls and work the same as viewstate works.

Hidden fields: Hidden field are the controls provided by the ASP.NET and they let use store some information in them. The only constraint on hidden filed is that it will keep the information when HTTP POST is being done, i.e., button clicks. It will not work with HTTP GET.

Cookies: Cookies are small pieces of information that can be stored in a text file on users' computer. The information can be accessed by the server and can be utilized to store information that is required between page visits and between multiple visits on the same page by the user.

Query Strings: Query strings are commonly used to store variables that identify specific pages, such as search terms or page numbers. A query string is information that is appended to the end of a page URL. They can be used to store/pass information from one page to another to even the same page.

Application State: ASP.NET allows us to save values using application state. A global storage mechanism that is accessible from all pages in the Web application. Application state is stored in the Application key/value dictionary. This information will also be available to all the users of the website. In case we need user specific information, then we better use sessionstate.

Session State: Like Application state, this information is also in a global storage that is accessible from all pages in the Web application. Session state is stored in the Sessionkey/value dictionary. This information will be available to the current user only, i.e., current session only.

Unit-IV

8 a) Discuss about data binding in ASP.NET. CO4 L2 7M

Ans)

Types of ASP.NET Data Binding:

Two types of ASP.NET data binding exist: single-value binding and repeated-value binding. Single-value data binding is by far the simpler of the two, whereas repeated-value binding
provides the foundation for the most advanced ASP.NET data controls.

Single-Value, or "Simple," Data Binding -- 3M

You can use single-value data binding to add information anywhere on an ASP.NET page. You can even place information into a control property or as plain text inside an HTML tag. Single-value data binding doesn't necessarily have anything to do with ADO.NET. Instead, single-value data binding allows you to take a variable, a property, or an expression and insert it dynamically into a page. Single-value binding also helps you create templates for the rich data controls .

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default2.aspx.cs" Inherits="Default2" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" Text=""><%#url %></asp:Label>
<br />
<asp:CheckBox ID="CheckBox1" runat="server" Text="<%#url %>" />
<br />
<asp:HyperLink ID="HyperLink1" runat="server" Text="click here" NavigateUrl="<%# url %>"></asp:HyperLink>
<br />
<asp:Image ID="Image1" runat="server" ImageUrl="<%#url %>" Height="164px" Width="235px" />
</div>
</form>
</body>
</html>
```

Default.aspx.cs:

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

```
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class Default2 : System.Web.UI.Page
{
protected string url;
protected void Page_Load(object sender, EventArgs e)
{
url = “picture.jpg”;
this.DataBind();
}
}
```

Repeated-Value, or “List,” Binding: -- 4M

Repeated-value data binding allows you to display an entire table (or just a single field from a table). Unlike single-value data binding, this type of data binding requires a special control
that supports it. Typically, this will be a list control such as CheckBoxList or ListBox, but it can also be a much more sophisticated control such as the GridView (which is described in Chapter 16). You’ll know that a control supports repeated-value data binding if it provides a DataSource property. As with single-value binding, repeated-value binding doesn’t necessarily need to use data from a database, and it doesn’t have to use the ADO.NET objects. For example, you can use repeated-value binding to bind data from a collection or an array.

```
<%@ Page Language=”C#” AutoEventWireup=”true” CodeFile=”Default4.aspx.cs” Inherits=”Default4” %>
<!DOCTYPE html>
<html xmlns=”http://www.w3.org/1999/xhtml”>
<head runat=”server”>
<title></title>
</head>
<body>
<form id=”form1” runat=”server”>
<div>
<asp:ListBox ID=”ListBox1” runat=”server” AutoPostBack=”True” OnSelectedIndexChanged=”ListBox1_SelectedIndexChanged”></asp:ListBox>
<br />
<br />
<br />
<asp:Label ID=”Label1” runat=”server”></asp:Label>
</div>
</form>
</body>
</html>
```

Default.aspx.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class Default4 : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
if (!this.IsPostBack)
{
Dictionary<int, string> fruit = new Dictionary<int, string>();
fruit.Add(1, “kiwi”);
fruit.Add(2, “Apple”);
fruit.Add(3, “Banana”);
fruit.Add(4, “Mango”);
fruit.Add(5, “blue berry”);
fruit.Add(6, “pine apple”);
fruit.Add(7, “Apriot”);
fruit.Add(8, “pear”);
fruit.Add(9, “peach”);
ListBox1.DataSource = fruit;
ListBox1.DataTextField = “value”;
ListBox1.DataValueField = “key”;
this.DataBind();
}
}
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
```

```
{
Label1.Text = “you picked” + ListBox1.SelectedItem.Text+”<br>”;
Label1.Text += “which as the key value :” + ListBox1.SelectedValue;
}
}
```

8. b) Explain about data controls in ASP.NET. CO4 L3 7M

Ans)

Various data controls available in ASP.NET are:

i) Grid View ii) Form view iii) Details View

GridView control: < 3 Marks >

The GridView is an extremely flexible grid control that displays a multicolumn table. Each record in your datasource becomes a separate row in the grid. Each field in the record becomes a separate column in the grid.The GridView is the most powerful of the rich data controls you’ll learn about in this chapter because itcomes equipped with the most ready-made functionality. This functionality includes features for automaticpaging, sorting, selecting, and editing. The GridView is also the only data control you’ll consider in this chapterthat can show more than one record at a time.

Features:

- Improved data source binding capabilities
- Tabular rendering – displays data as a table
- Built-in sorting capability
- Built-in select, edit and delete capabilities
- Built-in paging capability
- Built-in row selection capability
- Multiple key fields
- Programmatic access to the GridView object model to dynamically set properties, handle events and so on
- Richer design-time capabilities
- Control over Alternate item, Header, Footer, Colors, font, borders, and so on.
- Slow performance as compared to Repeater and DataList control .

FormView control: < 2 Marks >

The FormView provides a template-only control for displaying and editing a single record. FormViewtemplate matches quite closely the model of the TemplateField in the GridView. This means you can work with the following templates:

- ItemTemplate
- EditItemTemplate
- InsertItemTemplate
- FooterTemplate
- HeaderTemplate
- EmptyDataTemplate
- PagerTemplate

Like the DetailsView, the FormView can show a single record at a time. (If the data source has more than one record, you’ll see only the first one.) You can deal with this issue by setting the AllowPaging property to true so that paging links are automatically created. These links allow the user to move from one record to the next.

< 2 Marks >

DetailsView: The DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control. Unlike the GridView control, the DetailsView control displays one row from a data source at a time by rendering an HTML table. The DetailsView supports both declarative and programmatic data binding. By default displays information in two columns.

Features of DetailsView control:

- Tabular rendering
- Supports column layout, by default two columns at a time
- Optional support for paging and navigation.
- Built-in support for data grouping
- Built-in support for edit, insert and delete capabilities

(OR)

9 a) Describe the purpose and functionality of ADO.NET framework. Apply your knowledge to create a console application that retrieves data from a SQL Server database using ADO.NET's DataReader class. Analyze the benefits and limitations of using DataReader over other ADO.NET components.

CO4 L3 7M

Ans)

Console Application to access SQL Server DB using DataReader class:

```
static void HasRows(SqlConnection connection)
{
   using (connection)
   {
      SqlCommand command = new SqlCommand(
        "SELECT CategoryID, CategoryName FROM Categories;",
        connection);
      connection.Open();

      SqlDataReader reader = command.ExecuteReader();

      if (reader.HasRows)
      {
         while (reader.Read())
         {
            Console.WriteLine("{0}\t{1}", reader.GetInt32(0),
               reader.GetString(1));
         }
      }
      else
      {
         Console.WriteLine("No rows found.");
      }
      reader.Close();
   }
}
```

9. b) Create an ASP.NET web application for accessing SQL Server DB for performing CRUD operations. **CO4 L3 7M**

Ans)

Default.aspx: **→ 3M**

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div style="font-size: medium; background-color: #FF9933; width: 821px;" >


      <br />
      <asp:Label ID="Label1" runat="server" Text="Select Author:"></asp:Label>
       
      <asp:DropDownList ID="lstAuthors" runat="server" Height="36px" Width="228px"
AutoPostBack="True" OnSelectedIndexChanged="lstAuthors_SelectedIndexChanged">
      </asp:DropDownList>
            
      <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Update" style="margin-left:
0px" Width="86px" />
         
      <asp:Button ID="Button2" runat="server" OnClick="Button2_Click" Text="Delete" Width="73px" />
          
      <br />
      <br />
```

```
        <asp:Label ID="Label2" runat="server" Text="OR"></asp:Label>

                  
        <asp:Button ID="Button3" runat="server" OnClick="Button3_Click" Text="Create New" />
         
        <asp:Button ID="Button4" runat="server" OnClick="Button4_Click" Text="Insert New" />
        <br />

        <asp:Label ID="lblResults" runat="server" Text=""></asp:Label>
        <br />

        <br />

        <div style="font-size: medium; color: #000000; background-color: #669999; height: 401px; width:
818px;">
        <asp:Label ID="Label3" runat="server" Text="ID:"></asp:Label>

                  
          
            <asp:TextBox ID="IDTxtBox" runat="server" Width="212px" style="text-align: left"></asp:TextBox>
            <br />
            <br />
            <asp:Label ID="Label4" runat="server" Text="AuthorName:"></asp:Label>
          <asp:TextBox ID="AuthorNameTxtBox"
runat="server" Width="211px"></asp:TextBox>
            <br />
            <br />
            <asp:Label ID="Label5" runat="server" Text="Title:"></asp:Label>
                  
       
            <asp:TextBox ID="TitleTxtBox" runat="server" Width="211px"></asp:TextBox>
            <br />
            <br />
            <asp:Label ID="Label7" runat="server" Text="Publisher:"></asp:Label>
               
            <asp:TextBox ID="PublisherTxtBox" runat="server" Width="213px"></asp:TextBox>
            <br />
            <br />
            <asp:Label ID="Label8" runat="server" Text="Edition:"></asp:Label>
                  
  
            <asp:TextBox ID="EditionTxtBox" runat="server" Width="212px"></asp:TextBox>
            <br />
            <br />
  
            <br />
            <br />
                  
     
            <asp:Label ID="ResultstLbl" runat="server" Text="Label"></asp:Label>
            <br />

            <br />
        </div>
    </form>
</body>
</html>
```

Default.aspx.cs file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Data.SqlClient;
```

```
using System.Web.Configuration;
using System.Text;
public partial class _Default : System.Web.UI.Page
{
    string connectionString = null;
    //SqlConnection myConnection = null;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            FillAuthorList();
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        //update
        //UPDATE Authors SET phone='408 496-2222' WHERE au_id='172-32-1176'
        //delete
        connectionString =
       WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
        // Define ADO.NET objects.
        string nameToChange = null;
        string updateSQL;
        nameToChange = lstAuthors.SelectedItem.ToString();
        updateSQL = "UPDATE AuthorTab SET AuthorName= '";
        updateSQL += AuthorNameTxtBox.Text;
        updateSQL += "'";
        updateSQL += " WHERE AuthorName='";
        updateSQL += nameToChange;
        updateSQL += "'";
        SqlConnection con = new SqlConnection(connectionString);
        SqlCommand cmd = new SqlCommand(updateSQL, con);
        // Try to open the database and execute the update cmd.
        int updated = 0;
        try
        {
            con.Open();
            updated = cmd.ExecuteNonQuery();
            ResultstLbl.Text = updated.ToString() + " records updated.";
        }
        catch (Exception err)
        {
            ResultstLbl.Text = "Error updating record. ";
            ResultstLbl.Text += err.Message;
        }
        finally
        {
            con.Close();
        }
        // If the update succeeded, refresh the author list.
        if (updated > 0)
        {
            FillAuthorList();
        }

    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        //delete
        connectionString =
       WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
        // Define ADO.NET objects.
        string nametemp=null;
        string deleteSQL;
        deleteSQL = "DELETE FROM AuthorTab WHERE AuthorName = '";
        deleteSQL += AuthorNameTxtBox.Text;
        deleteSQL += "'";
```

```
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(deleteSQL, con);
    // Try to open the database and execute the delete cmd.
    int deleted = 0;
    try
    {
        con.Open();
        deleted = cmd.ExecuteNonQuery();
        ResultstLbl.Text = deleted.ToString() + " records deleted.";
    }
    catch (Exception err)
    {
        ResultstLbl.Text = "Error deleting record. ";
        ResultstLbl.Text += err.Message;
    }
    finally
    {
        con.Close();
    }
    // If the delete succeeded, refresh the author list.
    if (deleted > 0)
    {
        FillAuthorList();
    }
    //clear current data in the display
    IDTxtBox.Text = "";
    AuthorNameTxtBox.Text = "";
    TitleTxtBox.Text = "";
    PublisherTxtBox.Text = "";
    EditionTxtBox.Text = "";
}
protected void Button3_Click(object sender, EventArgs e)
{
    //create new
    IDTxtBox.Text = "";
    AuthorNameTxtBox.Text = "";
    TitleTxtBox.Text = "";
    PublisherTxtBox.Text = "";
    EditionTxtBox.Text = "";
    lblResults.Text ="Click Insert New to add the completed record.";
}
protected void Button4_Click(object sender, EventArgs e)
{
    //insert new
    // Perform user-defined checks.
    // Alternatively, you could use RequiredFieldValidator controls.
    connectionString =
    WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
    if (IDTxtBox.Text.Equals("") || TitleTxtBox.Text.Equals("")  || EditionTxtBox.Text.Equals("") )
    {
        ResultstLbl.Text = "Required fields missing.";
        return;
    }
    // Define ADO.NET objects.
    string insertSQL;
    insertSQL = "INSERT INTO AuthorTab(Id, AuthorName, Title, Publisher, Edition)";
    insertSQL += " VALUES('";
    insertSQL += IDTxtBox.Text + "', '";
    insertSQL += AuthorNameTxtBox.Text + "', '";
    insertSQL += TitleTxtBox.Text + "', '";
    insertSQL += PublisherTxtBox.Text + "', '";
    insertSQL += EditionTxtBox.Text;
    insertSQL += "')";
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(insertSQL, con);
    // Try to open the database and execute the update.
    int added = 0;
    try
```

```
        {
            con.Open();
            added = cmd.ExecuteNonQuery();
            ResultstLbl.Text = added.ToString() + " records inserted.";
        }
        catch (Exception err)
        {
            ResultstLbl.Text = "Error inserting record. ";
            ResultstLbl.Text += err.Message;
        }
        finally
        {
            con.Close();
        }
        // If the insert succeeded, refresh the author list.
        if (added > 0)
        {
            FillAuthorList();
        }
    }

private void FillAuthorList()
{
    connectionString =
        WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;

    lstAuthors.Items.Clear();
    // Define the Select statement.
    // Three pieces of information are needed: the unique id
    // and the first and last name.
    string selectSQL = "SELECT AuthorName FROM AuthorTab";
    // Define the ADO.NET objects.
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(selectSQL, con);
    SqlDataReader reader;
    // Try to open database and read information.
    try
    {
        con.Open();
        reader = cmd.ExecuteReader();
        // For each item, add the author name to the displayed
        // list box text, and store the unique ID in the Value property.
        while (reader.Read())
        {
        ListItem newItem = new ListItem();
        newItem.Text = reader["AuthorName"].ToString();
        newItem.Value = reader["AuthorName"].ToString();
        lstAuthors.Items.Add(newItem);
        }
        reader.Close();
    }
    catch (Exception err)
    {
        lblResults.Text = "Error reading list of names. ";
        lblResults.Text += err.Message;
    }
    finally
    {
        con.Close();
    }
    }
protected void lstAuthors_SelectedIndexChanged(object sender, EventArgs e)
{
    connectionString =
        WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
    // Create a Select statement that searches for a record
    // matching the specific author ID from the Value property.
    string selectSQL;
```

```
    selectSQL = "SELECT * FROM AuthorTab ";
    selectSQL += "WHERE AuthorName='" + lstAuthors.SelectedItem.Value + "'";
    // Define the ADO.NET objects.
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(selectSQL, con);
    SqlDataReader reader;
    // Try to open database and read information.
    try
    {
    con.Open();
    reader = cmd.ExecuteReader();
    reader.Read();
    // Build a string with the record information,
    // and display that in a label.
    IDTxtBox.Text = reader["Id"].ToString();
    AuthorNameTxtBox.Text= reader["AuthorName"].ToString();
    TitleTxtBox.Text = reader["Title"].ToString();
    PublisherTxtBox.Text=reader["Publisher"].ToString();
    EditionTxtBox.Text=reader["Edition"].ToString();
    reader.Close();
    }
    catch (Exception err)
    {
       lblResults.Text = "Error getting author. ";
       lblResults.Text += err.Message;
    }
    finally
    {
       con.Close();
    }
  }
}
```

Signature of the Internal Examiner	Signature of the HOD	Signature of the External Examiner