Visual Studio

Practical No: Date:

Aim: Introduction to visual c# using visual studio environment.

Theory:

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

C# - PROGRAM STRUCTURE

A C# program consists of the following parts:

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

Let us look at a simple code that prints the words "Hello World":

```
using System;
namespace HelloWorldApplication
{
  class HelloWorld
  {
    static void Main(string[] args)
    {
      /* my first program in C# */
      Console.WriteLine("Hello World");
      Console.ReadKey();
    }
  }
}
```

When this code is compiled and executed, it produces the following result:

```
Hello World
```

Let us look at the various parts of the given program:

1. The first line of the program **using System;** - the **using** keyword is used to include the **System** namespace in the program. A program generally has multiple **using** statements.

2. The next line has the **namespace** declaration. A **namespace** is a collection of classes. The *HelloWorldApplication* namespace contains the class *HelloWorld*.

3. The next line has a **class** declaration, the class *HelloWorld* contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the *HelloWorld* class has only one method **Main**.

4. The next line defines the **Main** method, which is the **entry point** for all C# programs. The **Main** method states what the class does when executed.

5. The next line /*...*/ is ignored by the compiler and it is put to add**comments** in the program.

6. The Main method specifies its behavior with the statement**Console.WriteLine("Hello World");**

   WriteLine is a method of the *Console* class defined in the *System*namespace. This statement causes the message "Hello, World!" to be displayed on the screen.

7. The last line **Console.ReadKey();** is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

It is worth to note the following points:

1. C# is case sensitive.
2. All statements and expression must end with a semicolon (;).
3. The program execution starts at the Main method.
4. Unlike Java, program file name could be different from the class name.

- **Steps to create A Simple Console Application Which Displays Hello World**

  Step 1: Open MVS

  From the *start =>All Programs* menu, select "Microsoft Visual Studio 2010".(MVS). Within MVS, there is both an **editor** within which computer code can be written, and a **compiler** that creates the application from the code.

  Step 2: Create a new project

  In MVS, select *File => New => Project*. A "New Project" window will pop up. In the left part of the window there is a list of Project types. From the "Installed Templates" directory, select "Visual C#". In the right part of the window is a list of Templates. For this lab we will select "Console Application".Enter a name and location and press the "OK". Fig. 1 illustrates this selection.

Your project has several files which you can view/edit by double clicking them from the "Solution Explorer", (see Fig. 2). The main file to edit is "Program.cs", which has several components. Notice the first 4 lines of the files start with the command using. This command allows your program to access various packages of code, each called a namespace, already written. The namespace keyword is used to declare a scope. This namespace scope lets you organize code and gives you a way to create globally unique types.

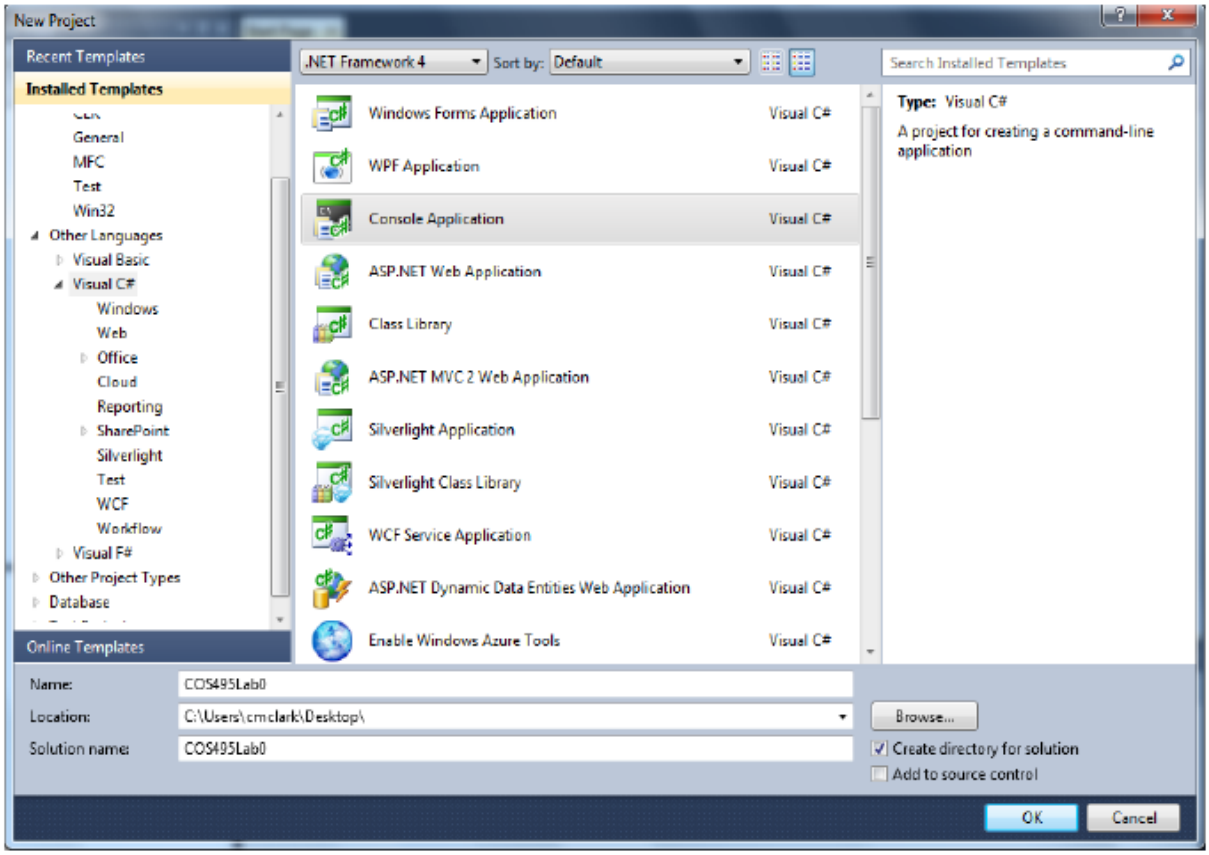


Figure 1: Creating a new console application

The next line “class Program” says to create a new class called Program. A class is an object that has various variables and methods associated with it. In the class Program, there is only one method called “Main”. The Main method of a program is what the program calls when it starts running.

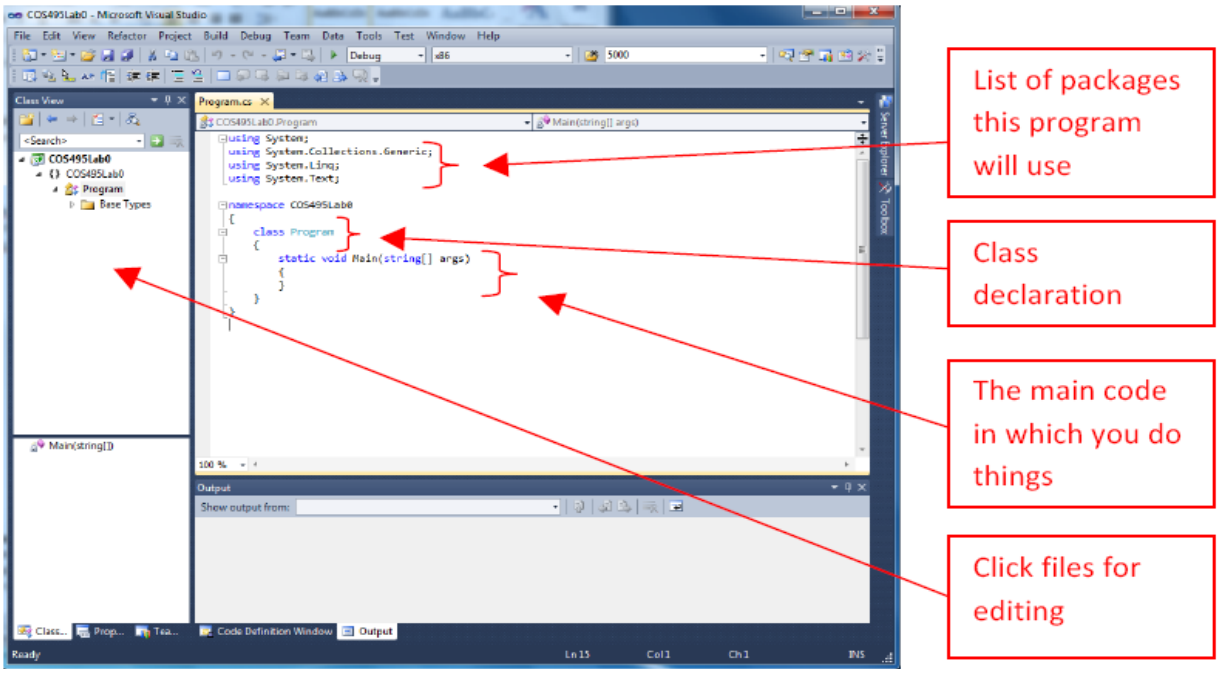


Figure 2: The template for a console application

Step 3: Add some commands

Within your Main() method of the class Program, add the following commands:

```
Console.WriteLine("Hello World!");
Console.ReadLine();
```

Your file Program.cs should look like the one shown in Figure 3. Note that Console is a class from the namespace System that the program says we want to use. Two methods that are part of the Console class are WriteLine() and ReadLine().

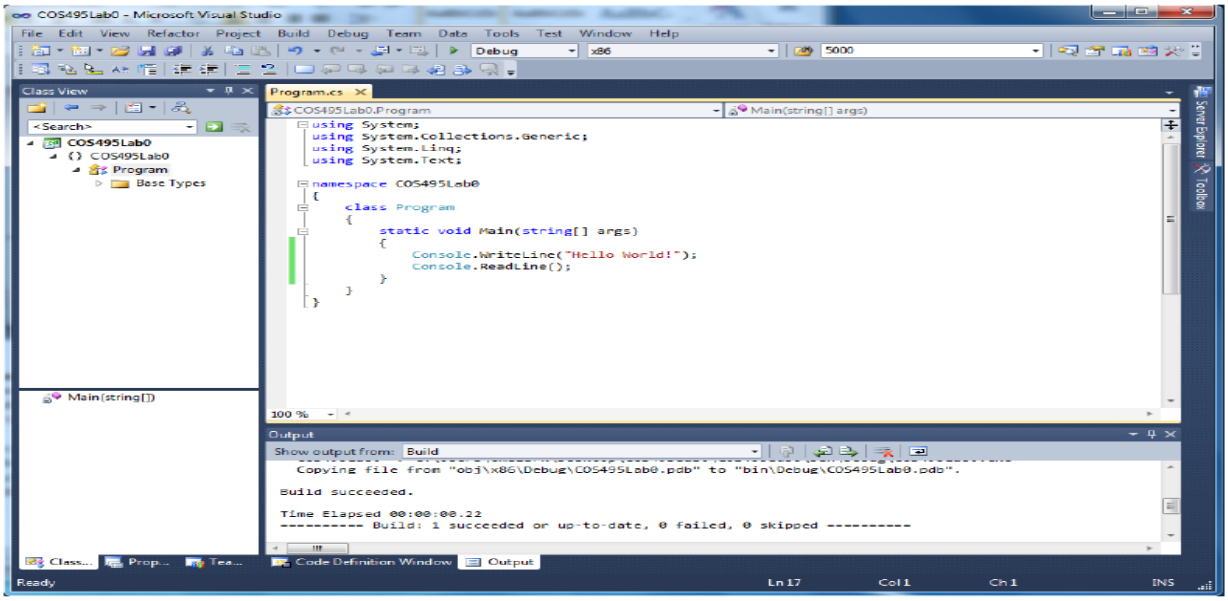


Figure 3: Adding commands to the Main method of your class.

Step 4: Compile your program

You have now created a very simple program that should open up a Console window with the message "Hello World!" shown. Pressing enter should close the program, (the ReadLine() command waits for the <Enter> key to be pressed before continuing).

To compile your program, i.e. creating an executable program from the code you have written, select *Build => Build Solution* from the MVS drop-down menu. If you did everything correctly, the Output fields at the bottom of your MVS window should read "========= Build: 1 succeeded" (see Figure 3). Otherwise, errors will be reported.

Step 5: Run your program

To run your program, click the green triangle OR select *Debug => Start Debugging* from the MVS dropdown menu. A console window should pop up with your "Hello World!" message. Hit the <Enter> key to close the program.

You have created, compiled, and run your first C# program!

TAKING NON-NUMERIC DATA FROM KEYBOARD INTO CONSOLE APPLICATION

- ReadLine () function works similar to scanf () function. Waits for the user until an input is given from the keyboard
- Write () and WriteLine () functions work similar to printf () function.

TAKING NUMERIC DATA IN CONSOLE APPLICATION

- ReadLine () function returns a 'string', so in case we want to work with an integer number we have to convert the string to integer by using **Convert.ToInt16 (**string**).** According to the integer's size you can also use **ToInt32, ToInt64** etc**.**

Exercise:

1. Write a program in which accepts user's details as name, city, age and pin number. Then store all the values in the appropriate variable and then print all the information in correct format.
2. Write a program in which accept two numbers from the user and returns four output value as add, subtract, multiplication and division.

Review Question:

1. Explain .net framework with figure.
2. Explain CLR and CTS.
3. What is namespace? Create C# program to demonstrate it Explain user defined namespace and show how to import it in other application.

BASICS OF C#

Practical No: Date:

Aim: To study about basics of c# (condition, loops, and arrays).

Theory:

C# DATA TYPES

The varibles in C#, are categorized into the following types:

- Value types
- Reference types
- Pointer types

VALUE TYPE

- Value type variables can be assigned a value directly. They are derived from the class **System.ValueType**.
- The value types directly contain data. Some examples are **int, char, and float**, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an **int** type, the system allocates memory to store the value.

REFERENCE TYPE

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.

In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value. Example of built-in reference types are: **object**, **dynamic,** and **string**.

OBJECT TYPE

The Object Type is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System. Object class. The object types can be assigned values of any other types, value types, reference types, predefined or user-defined types. However, before assigning values, it needs type conversion.

When a value type is converted to object type, it is called **boxing** and on the other hand, when an object type is converted to a value type, it is called **unboxing**.

```
object obj;
obj = 100; // this is boxing
```

DYNAMIC TYPE

You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

Syntax for declaring a dynamic type is:

```
dynamic <variable_name> = value;
```

For example,

```
dynamic d = 20;
```

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

STRING TYPE

The **String Type** allows you to assign any string values to a variable. The string type is an alias for the System. String class. It is derived from object type. The value for a string type can be assigned using string literals in two forms: quoted and @quoted.

For example,

```
String str = "Hello";
```

A @quoted string literal looks as follows:

```
@"Hello";
```

POINTER TYPE

Pointer type variables store the memory address of another type. Pointers in C# have the same capabilities as the pointers in C or C++.

Syntax for declaring a pointer type is:

```
type* identifier;
```

For example,

```
char* cptr;
int* iptr;
```

C# - DECISION MAKING

Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

C# provides following types of decision making statements. Click the following links to check their detail.

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside anotherswitch statement(s).

There may be a situation, when you need to execute a block of code several number of times. In general, the statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
Programming languages provide various control structures that allow for more complicated execution paths.
A loop statement allows us to execute a statement or a group of statements multiple times and following is the general from of a loop statement in most of the programming languages:

C# provides following types of loop to handle looping requirements. Click the following links to check their detail.

Loop Type	Description
while loop	It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	It is similar to a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

LOOP CONTROL STATEMENTS

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C# provides the following control statements. Click the following links to check their details.

Control Statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

INFINITE LOOP

A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

Example

```
using System;
namespace Loops
{
  class Program
  {
    static void Main(string[] args)
    {
      for (; ; )
      {
        Console.WriteLine("Hey! I am Trapped");
      } } }}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but programmers more commonly use the for(;;) construct to signify an infinite loop.

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

DECLARING ARRAYS

To declare an array in C#, you can use the following syntax:

```
datatype[] arrayName;
```

where,

- d*atatype* is used to specify the type of elements in the array.
- *[ ]* specifies the rank of the array. The rank specifies the size of the array.
- *arrayName* specifies the name of the array.

For example,

```
double[] balance;
```

INITIALIZING AN ARRAY

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array.

Array is a reference type, so you need to use the **new** keyword to create an instance of the array. For example,

```
double[] balance = new double[10];
```

Assigning Values to an Array

You can assign values to individual array elements, by using the index number, like:

```
double[] balance = new double[10];

balance[0] = 4500.0;
```

You can assign values to the array at the time of declaration, as shown:

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

You can also create and initialize an array, as shown:

```
int [] marks = new int[5]  { 99,  98, 92, 97, 95};
```

You may also omit the size of the array, as shown:

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
```

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location:

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
```

```
int[] score = marks;
```

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

ACCESSING ARRAY ELEMENTS

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example,

```
double salary = balance[9];
```

The following example, demonstrates the above-mentioned concepts declaration, assignment, and accessing arrays:

```
using System;
namespace ArrayApplication
{
  class MyArray
  {
    static void Main(string[] args)
    {
      int []  n = new int[10]; /* n is an array of 10 integers */
      int i,j;

      /* initialize elements of array n */
      for ( i = 0; i < 10; i++ )
      {
        n[ i ] = i + 100;
      }
      /* output each array element's value */
      for (j = 0; j < 10; j++ )
      {
        Console.WriteLine("Element[{0}] = {1}", j, n[j]);
      }
      Console.ReadKey();
    }
  }
}
```

C# ARRAYS

There are following few important concepts related to array which should be clear to a C# programmer:

Concept	Description
Multi-dimensional arrays	C# supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
Jagged arrays	C# supports multidimensional arrays, which are arrays of arrays.
Passing arrays to functions	You can pass to the function a pointer to an array by specifying the array's name without an index.
Param arrays	This is used for passing unknown number of parameters to a function.
The Array Class	Defined in System namespace, it is the base class to all arrays, and provides various properties and methods for working with arrays.

SINGLE-DIMENSIONAL ARRAYS:

int[] array = new int[5];

This array contains the elements from array[0] to array[4]. The new operator is used to create the array and initialize the array elements to their default values. In this example, all the array elements are initialized to zero.An array that stores string elements can be declared in the same way.

For example:

string[] stringArray = new string[6];

ARRAY INITIALIZATION:

It is possible to initialize an array upon declaration, in which case, the rank specifier is not needed because it is already supplied by the number of elements in the initialization list.

For example:

int[] array1 = new int[5] { 1, 3, 5, 7, 9 };

A string array can be initialized in the same way. The following is a declaration of a string array where each array element is initialized by a name of a day:

string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

C# - MULTIDIMENSIONAL ARRAYS

C# allows multidimensional arrays. Multi-dimensional arrays are also called rectangular array. You can declare a 2-dimensional array of strings as:

```
string [,] names;
```

or, a 3-dimensional array of int variables as:

```
int [ , , ] m;
```

TWO-DIMENSIONAL ARRAYS

The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays.

A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns:

Thus, every element in the array a is identified by an element name of the form a[i , j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in array.

INITIALIZING TWO-DIMENSIONAL ARRAYS

Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.

```
int [,] a = new int [3,4] {
  {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

C# JAGGED ARRAYS

A Jagged array is an array of arrays. You can declare a jagged array named*scores* of type **int** as:

```
int [ ][ ] scores;
```

Declaring an array, does not create the array in memory. To create the above array:

```
int[ ][ ] scores = new int[5][ ];
for (int i = 0; i < scores.Length; i++)
{   scores[i] = new int[4];
}
```

You can initialize a jagged array as:

```
int[ ][ ] scores = new int[2][ ]{new int[ ]{92,93,94},new int[]{85,66,87,88}};
```

Where, scores is an array of two arrays of integers - scores[0] is an array of 3 integers and scores[1] is an array of 4 integers.

Exercise:

1. Write C# code to display the asterisk pattern as shown below:

```
*****
*****
*****
*****
*****
```

2. Write a C# code to convert following conversion.
    a. Write C# code to convert infix notation to postfix notation.
    b. Write a C# code to Perform Celsius to Fahrenheit Conversion and Fahrenheit to Celsius conversion.
    c. Convert binary to decimal
3. Write a Program in C# to check whether a number is Palindrome or not.
4. Design a simple calculator using Switch Statement in C#.
5. Write a Program in C# to find the second largest element in a single dimensional array.
6. Write a Program in C# to multiply to matrices using Rectangular arrays.

Review Question:

1. Differentiate VB.Net and C#.Net.
2. Explain MSIL.
3. Explain C# arrays.

Function Overloading in C#

Practical No: Date:

Aim: To study about inheritance, function overloading, constructor and destructor in c#.

Theory:

INHERITANCE

Inheritance is the process of creating a new class from one or more existing class. In other words, a new class acquires the qualities of existing class.
Example Lets say that we need to model a bank account. Let's say that the bank has only two types of account – Savings Account and Current Account. Both of these accounts have some charateristics in common. So, we define a class based on common charateristics called BankAccount. Now to create a class called for SavingsAccount / CurrentAccount, we can derive/inherit the fetaures of BankAccount. This is known as inhertitance**.** BankAccount is called base class/ super class. SavingsAccount/CurrentAccount is called as derived class / sub class

- Inheritance can be single (only one base class) or multiple (two or more base class)). C# supports only single Inheritance. Multiple inheritance is not support in C#.
- To achieve multiple inheritance in C# we need to use Interfaces.

Inheritance in C# is indicated by the following syntax

<access modifier><classname> **:** <baseClassName>

Example : public class SavingsAccount : BankAccount { // declare the additional info needed for savings account}

In Inheritance, what ever is true for BaseClass is also true for derived class. For example, all the attributes and member functions in baseclass – BankAccount is also true for derived class - SavingsAccount .

Since derived classes are special category of base class, its called as **specialization**. In other words, specialization is the process of defining a new class based on definition of existing class**.** **Generalization** is the term used to refer the baseclass (i.e. recognizing the common features of several existing classes and creating a common base class for all of them) .

Inheritance is based on accessibility of the baseclass. All public members are inherited to derived class. But there are some complexities in inheriting other accessibility types. We will discuss them later.

Inheritance is also called as relationship between classes. It is different from that of Aggregation and Association. It differs at the object level. While Association and aggregation is between individual objects, Inheritance on the other hand is the way of describing a single object. ie with Inheritance, an object is simultaneously an instance of derived class and all of its base classes.

EXAMPLE OF INHERITANCE

```
using System;
namespace ProgramCall
{
   class BaseClass
   {
      //Method to find sum of give  2 numbers
      public int FindSum(int x, int y)
      {
         return (x + y);
      }
       //method to print given 2 numbers
      //When declared protected , can be accessed only from inside the derived class
      //cannot access  with the instance of  derived class
      protected void Print(int x, int y)
      {
         Console.WriteLine("First Number: " + x);
         Console.WriteLine("Second Number: " + y);
      }
   }
   class Derivedclass : BaseClass
   {
     public void Print3numbers(int x, int y, int z)
      {
         Print(x, y); //We can directly call baseclass members
         Console.WriteLine("Third Number: " + z);
      }
   }
   class MainClass
   {
      static void Main(string[] args)
      {
         //Create instance for derived class, so that base class members  can also  be accessed
         //This is possible because  derivedclass is inheriting base class
         Derivedclass instance = new Derivedclass();
         instance.Print3numbers(30, 40, 50); //Derived class internally calls base class method.
         int sum = instance.FindSum(30, 40); //calling base class method with derived class
         instance
         Console.WriteLine("Sum : " + sum);
         Console.Read();
      } } }
```

FUNCTION/METHOD OVERLOADING IN C#.NET

- The process of creating more than one method in a class with same name or creating a method in derived class with same name as a method in base class is called as method overloading.
- In VB.net when you are overloading a method of the base class in derived class, then you must use the keyword "Overloads".
- But in C# no need to use any keyword while overloading a method either in same class or in derived class.
- While overloading methods, a rule to follow is the overloaded methods must differ either in number of arguments they take or the data type of at least one argument.
- Method overloading provides more than one form for a method. Hence it is an example for polymorphism.
- In case of method overloading, compiler identifies which overloaded method to execute based on number of arguments and their data types during compilation it self. Hence method overloading is an example for compile time polymorphism**.**

EXAMPLE FOR METHOD OVERLOADING

```
using System;
namespace ProgramCall
{
   class Class1
   {
      public int Sum(int A, int B)
      {
         return A + B;
      }
      public float Sum(int A, float B)
      {
         return A + B;
      }
   }
   class Class2 : Class1
   {
      public int Sum(int A, int B, int C)
      {
         return A + B + C;
      }
   }

   class MainClass
   {
```

```
        static void Main()
        {
            Class2 obj = new Class2();
            Console.WriteLine(obj.Sum(10, 20));
            Console.WriteLine(obj.Sum(10, 15.70f));
            Console.WriteLine(obj.Sum(10, 20, 30));
            Console.Read();
        }
    }
}
```

CONSTRUCTOR

A special method of the class that will be automatically invoked when an instance of the class is created is called a constructor. The main use of constructors is to initialize private fields of the class while creating an instance for the class. When you have not created a constructor in the class, the compiler will automatically create a default constructor in the class. The default constructor initializes all numeric fields in the class to zero and all string and object fields to null. Some of the key points regarding the Constructor are:

- Constructor will not return anything.
- Constructor name is same as class name.
- By default, C# will create default constructor internally.
- Constructor with no arguments and nobody is called default constructor.
- Constructor with arguments is called parameterized constructor.
- Constructor by default public.
- We can create private constructors.
- A method with same name as class name is called constructor there is no separate keyword.

Constructors can be divided into 5 types:

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor
4. Static Constructor
5. Private Constructor

Now let us see each constructor type with example as below

Default Constructor

A constructor without any parameters is called a default constructor; in other words this type of constructor does not take parameters. The drawback of a default constructor is that every instance of the class will be initialized to the same values and it is not possible to initialize each instance of the class to different values. The default constructor initializes:

1. All numeric fields in the class to zero.
2. All string and object fields to null.

Parameterized Constructor

A constructor with at least one parameter is called a parameterized constructor. The advantage of a parameterized constructor is that you can initialize each instance of the class to different values.

Copy Constructor

The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance to the values of an existing instance.

Static Constructor

When a constructor is created as static, it will be invoked only once for all of instances of the class and it is invoked during the creation of the first instance of the class or the first reference to a static member in the class. A static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.

Some key points of a static constructor is:

1. A static constructor does not take access modifiers or have parameters.
2. A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
3. A static constructor cannot be called directly.
4. The user has no control on when the static constructor is executed in the program.
5. A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.

Private Constructor

When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class. They are usually used in classes that contain static members only. Some key points of a private constructor are:

1. One use of a private constructor is when we have only static members.
2. It provides an implementation of a singleton class pattern
3. Once we provide a constructor that is either private or public or any, the compiler will not add theparameter-less public constructor to the class.

Sample Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace BRK.ConstructorExample
{
```

```
   class Welcome
   {
      // Default constructor
      public Welcome()
      {
         Console.WriteLine("Welcome message from Default Constructor...");
      }
      // Parametarized constructor
      public Welcome(string name)
      {
         Console.WriteLine("\n\nThis message from parametarized constructor");
         Console.WriteLine("Welcome to Constructor sample, by {0}", name);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         // Creating object for Welcome class
         // This will called default constructor
         Welcome obj = new Welcome();
         // Creating object for welcome class by passing parameter
         // This will called parametarized constructor which matches
         Welcome pObj = new Welcome("Ramakrishna Basgalla");
         Console.Read();
      }
   }
}
```

DESTRUCTORS

.Net will clean up the un-used objects by using garbage collection process. It internally uses the destruction method to clean up the un-used objects. Some times the programmer needs to do manual cleanup.

Syntax:

```
~<ClassName>
{}
```

Sample Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace BRK.ConstructorExample
```

```
{
   class Welcome
   {
      // Default constructor
      public Welcome()
      {
         Console.WriteLine("Welcome message from Default Constructor...");
      }
      // Destructor
      ~Welcome()
      {
         Console.WriteLine("Destructor called");
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         // Creating object for Welcome class
         // This will called default constructor
         Welcome obj = new Welcome();
         Console.Read();
      }
   }
}
```

Note: Destructor will be called after execution of the program.

Exercise:

1. Write a program using function overloading to swap two numbers .
2. Define a class ‘salary’ which will contain member variable Basic, TA, DA, HRA. Write a program using Constructor with default values for DA and HRA and calculate the salary of employee.
3. Write a Program to demonstrate multiple inheritance.

Review Question:

1. In C#, Constructor can return the value. Justify.
2. Explain Constructor with example.
3. Explain

Windows Form Application

Practical No: Date:

Aim: To study about windows forms, events and dialogs.

Theory:

Until now all we have done is created console applications which have no good representation of Graphical User Interface (GUI). For doing this, we need to create windows application that can have single or multiple forms which are the means of GUI representation.

CREATING WINDOWS FORM APPLICATION

- Open Visual C# Express and go to **File > New Project**.
- Then from the list of templates, **choose Windows Forms Application**. A Windows Forms Application is a type of application that has a graphical user interface.
- **Name the project** MyFirstWindowsApplication.

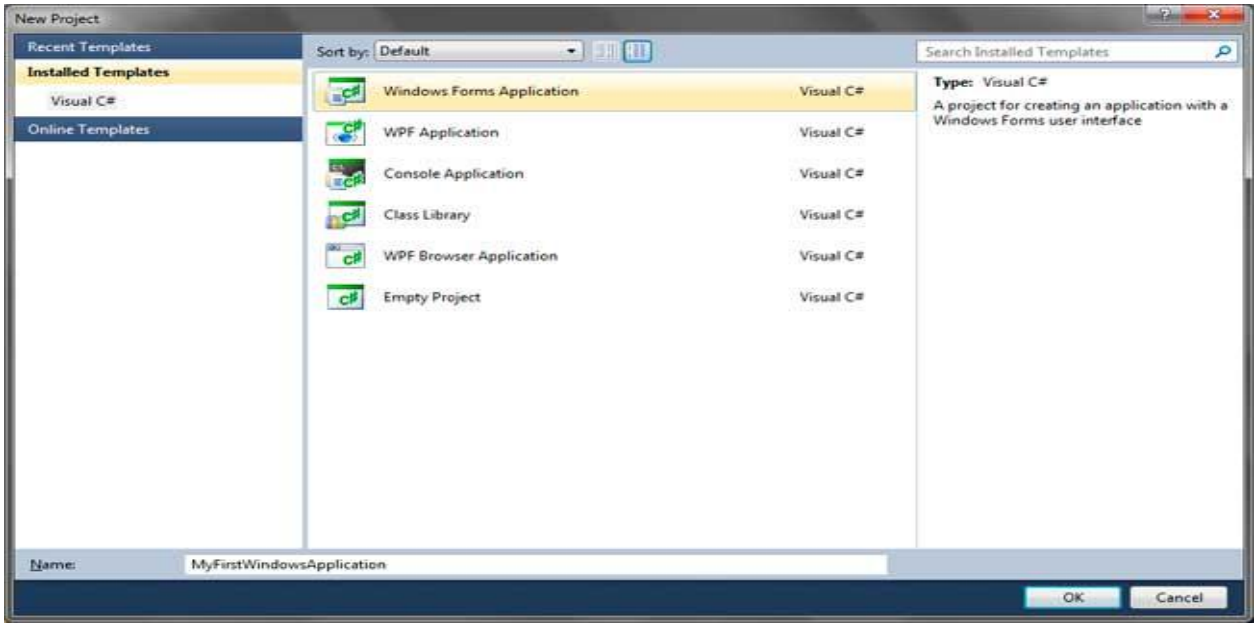


Figure 1 - New Project Window

You will be presented with a blank form. The selected tab tells that you are viewing the Form1.cs file in Designer View.

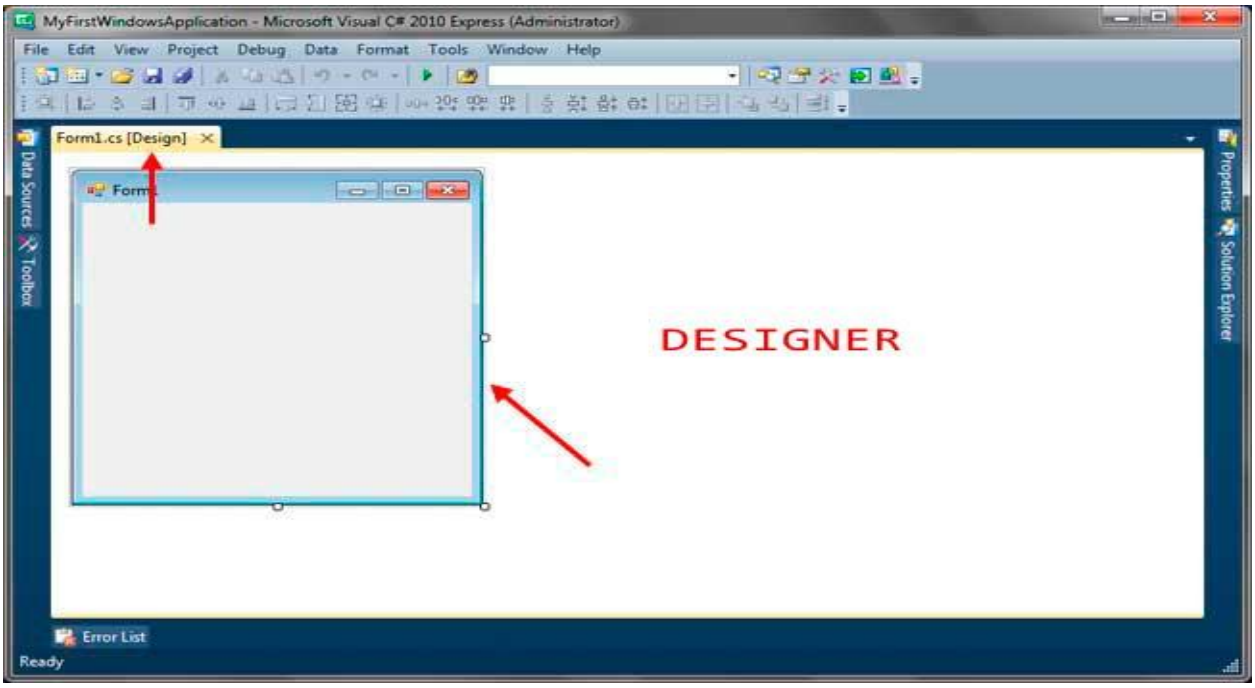


Figure 2 - Newly Created Blank Form

THE WINDOWS FORM

Windows Forms (or simply forms) are the windows you see in a Windows Application. You can create multiple forms in a single application. Each form inherits the properties and methods of the System.Windows.Forms.Form class. The namespace System.Windows.Forms contains components you will need for creating forms and controls.

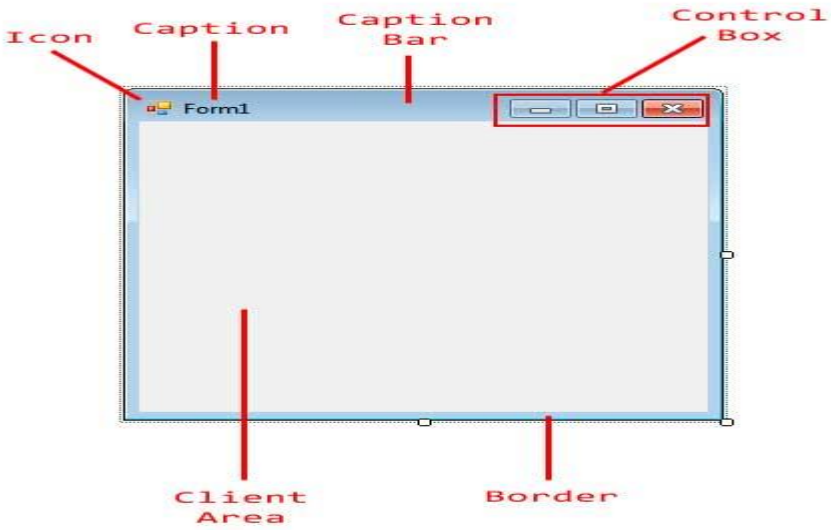


Figure 3. Parts of a typical windows form.

The figure shows the parts of a typical windows form.

At the top, you will find the Caption Bar. The Caption Bar is composed of the icon, the caption, and the control box. The control box contains buttons such as minimizing, maximizing, closing, or a help button. The Client Area is where we add the controls. The border or frame, which includes the caption bar,encloses the client area and allows you to resize the form.

Figure 4-properties of the Form base class.

Property	Description
AcceptButton	The button on the form that is pressed when you hit the Enter key.
CancelButton	The button on the form that is pressed when you hit the Esc key.
ClientSize	Gets or sets the client area of the form. The client area is the portion of the form inside the frame borders.
ControlBox	Specifies whether to show the control box at the top right portion of the form. The control box contains the buttons minimize, maximize, and close.
Controls	A collection of Control objects contained inside the form.
DesktopBounds	The size and location of the form in the Window's desktop.
Font	The font that the form will use. Controls inside the form will inherit this property.
FormBorderStyle	The border style of the form.
HelpButton	Shows a help button right before the close button of the form. (minimize and maximize buttons should be disabled)
Icon	The icon that will be used by the form.
Location	The coordinates of the form in the screen.
MainMenuStrip	Specifies the main menu to be used by the form.
MaximizeBox	Tells whether the maximize box located at the top right is displayed.
MinimizeBox	Tells whether the minimize box located at the top right is displayed.
Modal	Tells whether the form is modal.
Name	The name of the form that is used to reference it in the code.
OwnedForms	A collection of forms that this form owns.
Owner	The form that owns this form.
ShowIcon	Tells whether the icon is displayed at the left side of the caption bar.
Size	The size of the form.
StartPosition	The starting position of the form when it is initially shown.
Text	The text that is shown in the caption bar of the form.

Method	Description
Activate	Gives the focus to this form and activates it.
AddOwnedForm	Adds a new form that this form owns.
CenterToScreen	Centers the position of the form in the screen.
Close	Closes the form.
Hide	Hides this form.
OnLoad	Raises the Load event.
Show	Shows the form.

Figure 5-Methods of the Form base class

ADDING CONTROLS TO THE FORM

All the controls are located in the Toolbox. The Toolbox can be accessed via the Toolbox tab located by default at the left of the IDE. If it is not shown, you can go to View > Other Windows > Toolbox. Hover your mouse over or click the Toolbox tab to show the actual Toolbox.

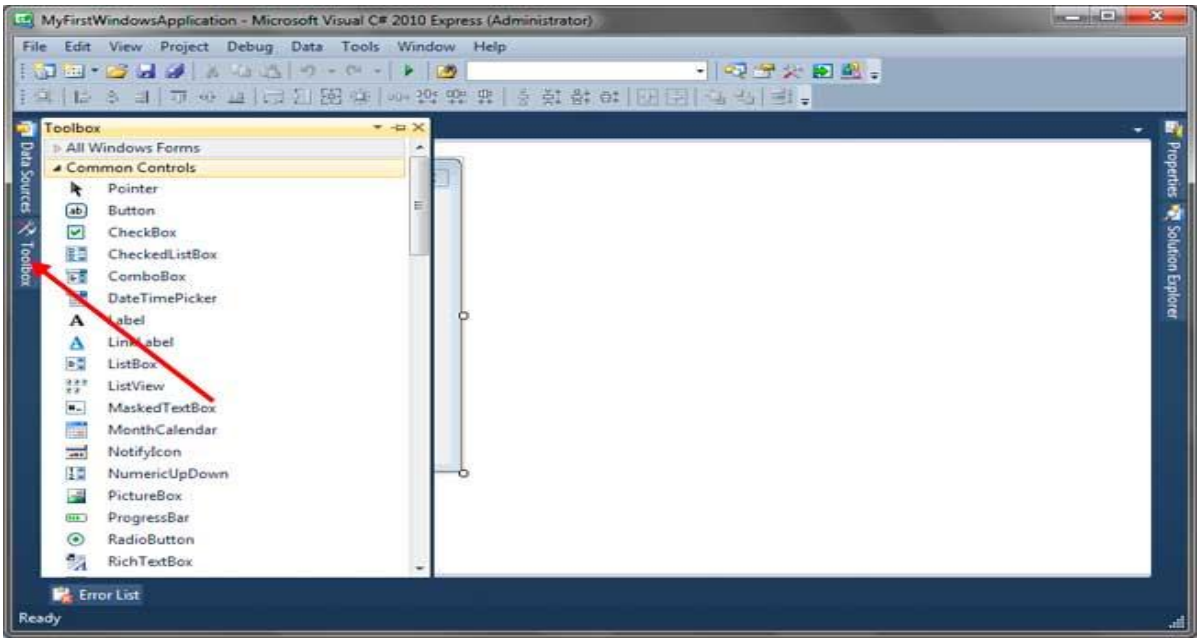


Figure 6 - Toolbox Tab

The Toolbox is divided into categories and the most commonly used controls are located in the Common Controls category. To open a category and expose its control, simply click the category. The Toolbox will auto-hide by default. If you don't want that behavior, then you can click the pin icon beside the close button of the Toolbox.

To add controls to the form, choose a control in the Toolbox and double click it. Alternatively, you can drag the control from the Toolbox to the form. Please note that you can only add controls to the client area of the form. The client area is the blank area of the form. You can delete a control or controls by selecting it in the designer and hitting delete in your keyboard.

MODIFYING THE CONTROL BOX

We use the ControlBox property to hide or show the Control Box. This is useful when you are planning to disable minimizing or maximizing of control or you want to only close the form through the code.

ENABLE AND DISABLE WINDOWS FORM CONTROLS

We can set the properties to enable or disable the windows form. to disable a windows form, simply set the Enabled property to False in the properties window of the form. Similarly, you can enable a disabled Windows Form by setting the Enabled property to True. By default Enabled property of a Windows Forms is True.

DIALOGS

Dialogs are windows that have a certain specific function such as saving or opening a file, choosing a color, or printing a document. The .NET framework offers the dialog controls which allows you to call dialogs to do a specific task.

THE ColorDialog CONTROL

The ColorDialog (System.Windows.Forms.ColorDialog) is used when you want to pick different colors. For example, when you want to pick a color of the font or a background color for the form, you can use the ColorDialog control.

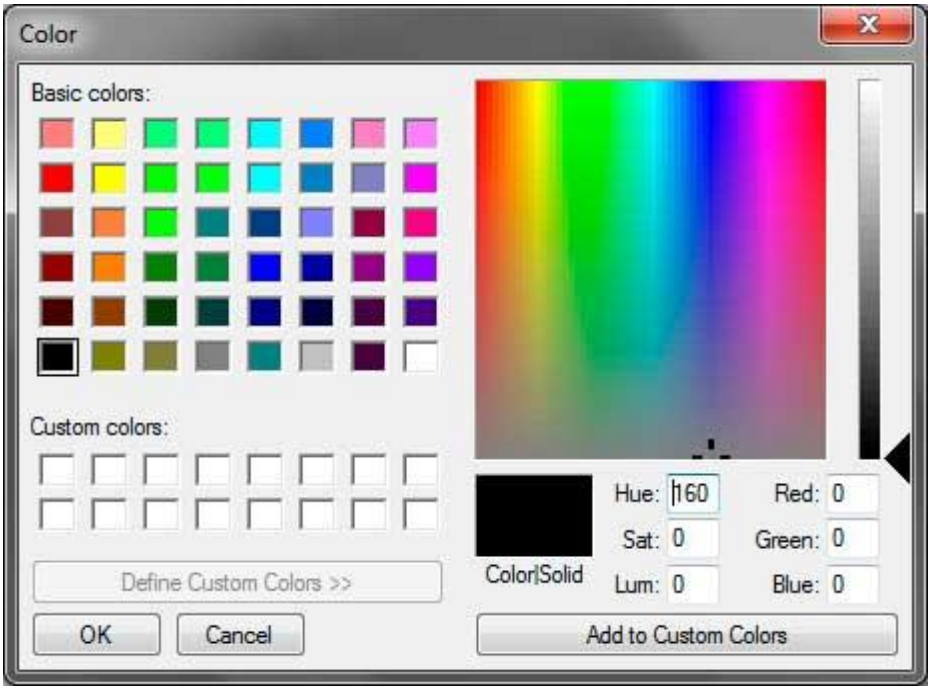


Figure 8-ColorDialog control

The following are some of the useful properties of the ColorDialog control.

Properties	Description
AllowFullOpen	Specifies whether the user can choose custom colors.
Color	The color that the user selected.
CustomColors	A collection of custom colors picked by the user.
FullOpen	Specifies whether the part used to pick custom colors are automatically open.

Figure 9-Properties of ColorDialog control

The window of the initially composed of predefined color palettes. You can click the Define Custom Colors button to reveal a more colors where you can pick every color you can think of. You can even provide the Hue, Saturation, Luminance values or RGB values. The main window allows you to choose colors while the right slider adjusts the brightness of the color. You can

click the Add Custom Colors button to put the selected color to the Custom Colors palette so you can reuse it later.

The FontDialog Control

The FontDialog control (System.Windows.Forms.FontDialog) is a handy control for selecting different kinds of font and font-related properties.

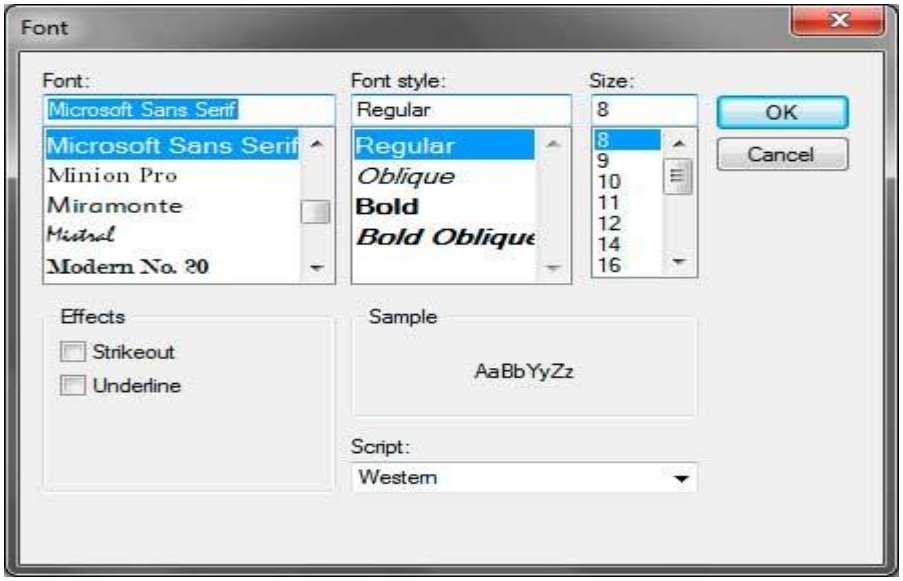


Figure 10- FontDialog control

Using the FontDialog, you can change the font type, style, size, color, add effects and specify character sets for the font. You can also preview the actual font. The following are some of the useful properties of the FontDialog control.

Properties	**Description**
Color	The selected color of the user.
Font	The resulting font constructed using the font dialog.
MaxSize	The maximum size the dialog can provide.
MinSize	The minimum size the dialog can provide.
ShowApply	Indicates whether to show the Apply button.
ShowColor	Indicates whether to show the Color option.
ShowEffects	Indicates whether to show the Effects option.

Figure 10- Properties of FontDialog control

Note that the FontDialog hides the option to choose color by default. To allow the user to choose a color, set the value of the ShowColorproperty to true. You can then access the selected color via the Color property. The ShowApply property allows you to show the Applybutton on the FontDialog. You can then handle the Apply event to apply the changes to the target font while not closing the dialog itself.

The FolderBrowserDialog Control

The FolderBrowserDialog control (System.Windows.Forms.FolderBrowserDialog) allows you to browse for a directory in your system. The control uses a tree view to show all the folders. The Browser shows the Desktop (by default) as the top level root folder and below it is its content. You can click the arrows or each folder to show other folders/subdirectories inside it. You can also create a new folder inside the selected folder or directory.

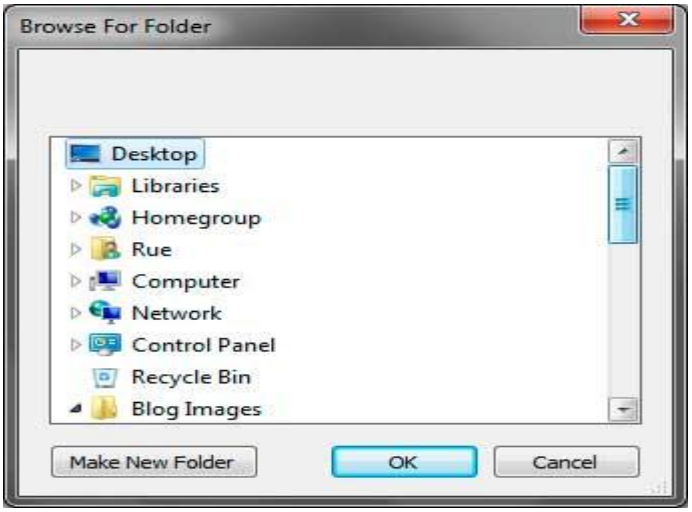


Figure 10- FolderBrowserDialog control.

The following are some of the useful properties of the FolderBrowserDialog control.

Property	**Description**
Description	Allows you to add a descriptive text above the tree view of the FolderBrowserDialog.
RootFolder	Gets or sets the folder that the FolderBrowserDialog will consider as the root or the top level folder.
SelectedPath	The folder or path that the user has selected.
ShowNewFolderButton	Shows or hides the Make New Folder button.

Figure 11-Properties of the FolderBrowserDialog control

The OpenFileDialog Control

The OpenFileDialog control (System.Windows.Forms.OpenFileDialog) allows you to open and read file contents such as texts from text files. The dialog allows you to browse your file system and pick a directory. You can type the name of the file that exist in the current directory or you can type the whole path of a file or directory in the File Name text box located at the bottom of the dialog.

The following table shows some useful properties of the OpenFileDialog control.

Property	**Description**
AddExtention	Specifies whether to automatically add an extension when the user omits the extension of the file he or she chooses.
CheckFileExists	Specifies whether to initiate a warning if the user types a file that does not exist.
CheckPathExists	Specifies whether to initiate a warning if the user types a path that does not exist.
DefaultExt	The default extension to add when the user does not indicate a file extension.
FileName	The file selected by the user. This can also be the default selected the file when the dialog shows up.
FileNames	A collection of files that the user picked.
Filter	Allows you to add a filter which are a special string indicating which types or files are only allowed to be opened by the user.
FilterIndex	If multiple filters are present, this indicates which filter shows as the default starting with index 1.
InitialDirectory	The initial directory that the OpenFileDialog will show.
Multiselect	Tells whether the user can select multiple files.
Title	The title of the dialog.

Figure 12-Properties of the OpenFileDialog control.

The AddExtention property automatically adds an extension when the user does not indicate the file extension of the file name. The extension to add is specified by the DefaultExt property. The CheckFileExists and CheckPathExists methods are recommended to be set to true so the dialog will issue a warning message if it cannot find the specified file or directory. The InitialDirectory property specifies the initial directory that the dialog will show when you open it up. The Title property is the title of the dialog located at the title bar. The FileName property is the selected file selected or specified by the user. You can allow a user to select multiple files by setting theMultiselect property to true. You can then use the FileNames property to get the collection of selected file names.

The SaveFileDialog Control

The SaveFileDialog control (System.Windows.Forms.SaveFileDialog) allows you to save or write data to a specified file. The dialog allows you to browse your file system and pick a directory, then type a filename you want to be the name of the file that will contain the data to be written. You can type the name of the file that exist in the current directory and the dialog can prompt you if you want to overwrite it. You can also type the whole path of a file or directory in the File Name text box located at the bottom of the dialog.

Exercise

1. Create a form using C# that takes name and message from the user and choose a color by radio button, select a style for ex.-bold, italic, underline from the checkbox and display in label control, when you clicked on display button. And clear the information when you clicked on clear button.
2. WAP to change background color for the form using color dialog and change Font styles of a text using Font styles Dialog.
3. WAP to save a text file using saving dialog box and browse that saved file using browsing dialog

Review Question:

1. What is difference between a CheckBox control and a RadioButton control?
2. List out various windows form controls. Explain any three of it.
3. Write a C# program which creates windows form with one Button,Textbox.When button is clicked it should display “Hello World” message and display red backcolor in the textbox

Windows Form and Control

Practical No: Date:

Aim: To study about windows forms and controls.

Theory:

The Button control

The button control (System.Windows.Forms.Button) is commonly used to execute commands when it is clicked. When a button is clicked, you specify codes that will be used. Buttons are typically used to confirm or cancel an action, to perform different actions, and to open some more dialogs. The Button control has several properties that you can use.

Property	Description
AutoEllipsis	Specifies whether to append dots (...) when the text in the button is too long and can't fit the button.
AutoSize	Specifies whether the button will automatically resize to fit its content.
FlatStyle	Determines the style of the button. Popup makes the button flat, and when you hover on the button, the button will pop out. Flat makes the button flat and when you move point your mouse inside the button, the background color of the button changes.
Enabled	If set to false, the button cannot be clicked or receive focus.
Image	An optional image that you can place inside the control.
ImageAlign	The alignment of the image in the button.
Text	The caption inside the button.
Visible	Tells whether the button is visible or not.

Figure 1 - Button Properties

A button is still useless by just editing its properties. It needs to react to events to do some work. The following are the most common events available for the Button control.

Event	Description
Click	Occurs when you click the button.
Enter	Occurs when the control becomes theactive control of the form.
Leave	Occurs when the control becomesinactive anymore.
LocationChanged	Occurs when the location of the button is changed.
MouseDown	Occurs when the mouse pointer is in the button and the mouse button is pressed down.
MouseEnter	Occurs when the mouse enters the button.

MouseHover	Occurs when the mouse stays stationary in the button for an amount of time.
MouseUp	Occurs when you pressed the button and you let go of the mouse button.
MouseLeave	Occurs when the mouse pointer leaves the button.

Figure 2 – Button Events

The Label Control

The Label control (System.Windows.Forms.Label) is used to add text to a form that can be used to show messages, or add labels to identify what other controls' functionality is. Drag a label control from the toolbox to the form. By default, it will have an initial text. The following properties are the most common ones that you will modify.

Property	**Description**
AutoSize	If true, the size of the borders of the label control in the designer will be resized automatically depending on the text inside it.
BorderStyle	Specifies the type of border around the label.
Font	Used to change the font properties of the text inside the label control.
Text	The text of the label.
TextAlign	The alignment of the text inside the Label control

Figure 3 - Lable Properties

The TextBox Control

The TextBox control (System.Windows.Forms.TextBox) is the most basic means of input in a windows forms. You give the input data by typing it inside the text box. The text you type can be accessed by using the Text property of the control. The following table shows some useful properties that you can use for the TextBox control.

Property	**Description**
AcceptsReturn	Used with Multiline. Tells if whether the return key is included in the input. The return will be converted into a \n escape sequence.
AcceptsTab	Indicates whether to accept tab as an input. By default, hitting tab will bring the focus to the control with the next TabIndex in the form.
Enabled	Set to false to make the text box read-only therefore making the text box act like a label.
Font	The font properties that will be used by the textbox.
Lines	The lines of text in a multiline text box.
Multiline	Set to true to allow multiple lines in a text box.
Text	The text inside the text box.
PasswordChar	Accepts a character that will be used to mask each character typed by the user.

ReadOnly	Tells whether the text in the form can be edited.
Visible	Tells whether the text box is visible inside the form.
WordWrap	Used with Multiline. Set to true to enable automatic wrapping of words.

Figure 4 - TextBox Properties

The RichTextBox Control

The RichTextBox control (System.Windows.Forms.RichTextBox) is similar to a TextBox control, but it allows you to format different parts of the text inside it. The TextBox control is typically used to accept text input from the user while the RichTextBox control is used to show formatted text and save it in Rich Text Format (RTF).

The RichTextBox control allows you to format parts of the whole text of the control. You can also use a rich text box control to accept input from the user as both TextBox and RichTextBox are derived fromTextBoxBase class, therefore, they share the most common properties such as the Text property. The following are some common properties available when using the RichTextBox control.

Property	**Description**
CanRedo	Specifies a whether there are actions that have occurred within the RichTextBox that can be reapplied.
CanUndo	Specifies whether the user can undo the operations done in a RichTextBox control.
DetectUrls	Specifies whether to automatically detect and add the target link to URLs inside the RichTextBox control.
Lines	A collection of individual lines in a RichTextBox control.
Rtf	Contains the text of the RichTextBox control including the rich text format (RTF) codes.
Scrollbars	Specifies the type of scrollbars to display.
SelectedRtf	Gets or sets the currently selected rich text format (RTF) formatted text in the control.
SelectedText	Gets or sets the selected text within the RichTextBox.
SelectionAlignment	Specifies the alignment to apply to the current selection or insertion point.
SelectionBackColor	Specifies the background color of the selected text.
SelectionBullet	Specifies whether the bullet style is applied to the current selection or insertion point.
SelectionCharOffset	Specifies whether the selected text in the control appears on the baseline, as a superscript, or as a subscript below the baseline.
SelectionColor	Specifies the color of the selected text.
SelectionFont	Specifies the font that the selected text will use.
SelectionLength	Specifies the number of characters of the selected text.
SelectionProtected	Tells whether the selected text or all the text after the insertion point is protected.

Figure 5 - RichTextBox Properties

The RadioButton Control

The RadioButton control (System.Windows.Forms.RadioButton) is a button that can be either turned on or off. The radio button is a circular button with a label. You simply click a radio button to change it from off to on or vice versa. When the radio button is turned on, a dot can be seen and when it is turned off, the circle appears empty. Radio buttons are usually used when a user must select one choice from a set of choices. For example, if you want to determine the gender of the user, then you can use two radio buttons labeled Male and Female. Since you used a radio button, you can only choose one of the two. The following are some properties of the radio button control.

Property	**Description**
Appearance	The radio button can be displayed as a normal button,or a circular button with a label beside it.
CheckAlign	Determines the alignment of the button. The default is MiddleLeft, which will show the button to the left of the label.
Checked	When set to true, the radio button will be in its "ON" state and a dot will be seen inside the button.
Text	Sets the text inside the label of the radio button.

Figure 6 - RadioButton Properties

The CheckBox Control

The CheckBox control (System.Windows.Forms.CheckBox) is also a type of button and appears as an empty box with a label beside it. By default, when the empty box is clicked, a check will show up inside the box telling that the checkbox control is in its "checked" state. Unlike a radio button, you can check multiple or even all of the checkboxes. The CheckBox control contains similar properties as the radio button control. The following are some properties that are exclusive to the CheckBox control.

Property	**Description**
Checked	Determines if the checkbox is checked.
CheckState	Tells whether the checkbox is Checked, Unchecked.
ThreeState	If true, the checkbox can accept an Intermediate state.

Figure 7 - CheckBox Properties

Unlike the radio button, the CheckBox control can have three states by setting the ThreeState property to true. Those three states areChecked, Unchecked or Intermediate. Intermediate indicates that value of the checkbox is invalid or cannot be determined

The ComboBox Control

The ComboBox control is another way of allowing a user to choose from a set of options. The ComboBox control looks like a text box with a button in its right side. When the button is clicked, the ComboBox show a drop-down box containing the list of options/items available. The user can

then choose from these options by clicking one of them. The selected option will then be the text inside the ComboBox. The following are some of the properties of the ComboBox control.

Property	Description
DataSource	A list of data that the control will use to get its items.
DropDownHeight	The height in pixels of the dropdown box in a combo box.
FormatString	The format specifier characters that indicate how a value is to be displayed.
Items	The items in the combo box.
Sorted	Specifies whether the items on the combo box should be sorted.
Text	The default text when there is no item selected.
SelectedIndex	Gets or sets the selected index. Every item has an index starting from 0 to (number of items - 1). A value of -1 indicates that no item is selected.
SelectedItem	Gets or sets the item of the currently selected item.

Figure 9 - ComboBox Properties

The ListBox Control

The ListBox control is used to show a list of strings which you can select. By default, you can only select one item. The ListBox control is best used if you are trying to display a large number of items. The following are the commonly used properties of the ListBox control.

Property	Description
ColumnWidth	Specifies the width of each column if MultiColumn is set to true.
DataSource	Specifies the source of data that the ListBox will display.
Items	Contains the items that the ListBox will display.
MultiColumn	Tells whether the ListBox supports multiple columns.
SelectedIndex	The zero-based index of the selected item.
SelectedIndices	Contains the zero-based index of each selected item.
SelectedItem	Returns the selected item as an object.
SelectedItems	An object collection of the selected items.
SelectionMode	Specifies the number of items you can select at the same time. • SelectionMode.None - you cannot select anything • SelectionMode.One - you can only select one item • SelectionMode.MultiSimple - you can select multiple items by simply clicking them • SelectiomMode.MultiExtended - you can select multiple items by holding ctrl, shift and arrow keys
ScrollAlwaysVisible	Tells whether the scrollbars are always visible regardless of the number of items in the ListBox.
Sorted	Tells whether to sort the items in the ListBox alphabetically or in ascending order.
Text	If you set a string value, the first item that matches will be selected. This property returns the text of the first selected item.

Figure 10 - ListBox Properties

The CheckedListBox Control

The CheckedListBox control is similar to the ListBox control except that each item is represented by a CheckBox control. You can select each item like an ordinary ListBox, but in addition, you can also check the box beside each item.

The properties and methods of the ListBox control can also be found in the CheckedListBox control. But the CheckedBoxControl has some exclusive properties listed below.

Properties	**Description**
CheckedIndices	A collection of checked indices of the CheckedListBox control.
CheckedItems	A collection of checked items.
CheckOnClick	Specifies whether to check the box of an item if the item is selected.
ThreeDCheckBoxes	Specifies whether the checkbox should be flat(two-dimensional appearance) or normal(three-dimensional appearance).

Figure 11 - CheckedListBox Properties

The NumericUpDown Control

The NumericUpDown control is typically used to get numeric inputs and automatically restricts user for giving invalid non-numeric values. The NumericUpDown control appears like a TextBox control, but there are arrow buttons on its right or left side that is used to increment or decrement the value of the control.

The numeric value of the NumericUpDown control can be accessed or retrieved using the Value property which is of type decimal. The following are some of the properties of the NumericUpDown control.

Property	**Description**
DecimalPlaces	Indicates the number of decimal places to display.
Hexadecimal	Indicates whether the NumericUpDown control should display its value in hexadecimal.
Increment	Indicates the amount that will be incremented or decremented to the current value when the arrow buttons are pressed.
InterceptArrowKeys	If set to true, you can use the arrow keys in your keyboard to increment or decrement the value.
Maximum	Indicates the maximum value that the NumericUpDown control can contain.
Minimum	Indicates the minimum value that the NumericUpDown control can contain.
ThousandsSeperator	Indicates whether the value will be displayed with the thousands separator. (example: 1,000)
UpDownAlign	Indicates where the arrow buttons are positioned. Set to Right to place the buttons to the right side and Leftto place them to the left side.
Value	The current value of the NumericUpDown control.

Figure 12 - NumericUpDown Control Properties

The PictureBox Control

You can display images on your form by using the PictureBox control. It is a simple control which has a main purpose of displaying images. All you have to do is browse for the desired image and Visual Studio/VCE will import it to your project. You can use several image formats such as JPEG, GIF, PNG, and BMP.
The following are the useful properties of the PictureBox control:

Properties	**Description**
ErrorImage	The image that will be displayed when the loading of the actual image failed or is canceled.
Image	The image that will be displayed by the control.
ImageLocation	The path of the image to be displayed by the PictureBox.
InitialImage	The image that is initially displayed while the main image is loading.
SizeMode	Tells how the image will be displayed. Accepts values from the System.Windows.Forms.PictureBoxSizeMode
WaitOnLoad	If set to true, blocks all interaction to the form while the image is being loaded.

Figure 13 - PictureBox Properties

To display an image using the PictureBox control, there are multiple ways you can use. You can go to the Properties Window and find the Image property. Click the button to the right of it to bring out the Select Resource Dialog.

Once the image is displayed, it may not look like you want it to. If the loaded image is larger the size of the PictureBox, then the image will be clipped. You can use the SizeMode property to change the way the image is positioned or resized inside the control. The property uses values from the System.Windows.Forms.PictureBoxSizeMode enumeration which are presented below.

PictureBoxSizeMode	**Description**
Normal	The image will be positioned in the upper-left corner of the PictureBox and if the image is larger than thePictureBox, the image will be clipped.
StretchImage	Resizes the image to match the size of the PictureBox.
AutoSize	Resizes the PictureBox to match the size of the image.
CenterImage	The image is centered inside the PictureBox. If the image is larger than the PictureBox, the image will be clipped.
Zoom	Fits the whole image inside the PictureBox w while maintaining the image's size ratio.

Figure 14 - System.Windows.Forms.PictureBoxSideMode Values

The most common and default event of the PictureBox control is the Click event which is called when the image is clicked.

The LinkLabel Control

The LinkLabel control is similar to an ordinary label. But it has an underline and resembles a link on a web page. The LinkLabel control can be used to link to files, directories, or web pages. When you place your mouse over the LinkLabel, the mouse pointer will turn into a hand.

The properties of the LinkLabel control is as follows:

Property	**Description**
BorderStyle	The style of the border around the label.
FlatStyle	Determines the appearance of the LinkLabel. When set to Popup, the button is slightly raised when you hover over it.
LinkArea	Indicates the portion of the text that will be displayed as a link.
LinkColor	The color of the unvisited link.
Links	A collection of links that will be displayed. These are not the actual links that will be visited but portions of theLinkLabel that will be displayed as links.
LinkVisited	When set to true, the color of the link will be swapped by the color of the VisitedLinkColor property.
TextAlign	Specifies the location of the text within the control.
VisitedLinkColor	The color The color of a visited link.

Figure 15 - LinkLabel Properties

The MonthCalendar Control

The MonthCalendar control (System.Windows.Forms.MonthCalendar) resembles a calendar and shows a month and all its dates. TheMonthCalendar control allows you to choose a month and a date. To choose a month, click the left and right arrows to move the month to the next or previous month.

Clicking the month header, will show all the months of the current year. Clicking the header even further will show all the years of the current decade, and clicking it once more will show all the year of the current century. You can customize the MonthCalendar control using the different properties it offers.

The DateTimePicker Control

The DateTimePicker control (System.Windows.Forms.DateTimePicker) is used to pick a single date. The control appears by default, as a combo box with a calendar icon at the right part. You can select each date component and such as the month and use the arrow keys to adjust individual components.

The DateTimePicker control shows (by default) the current date or the selected date. By clicking the calendar icon during runtime, you will be presented with a calendar where you can choose dates. You can then use this calendar to choose a specific date. You use the left and right navigation arrows to move to the previous or next month. The initial view of the calendar shows

all the days plus some trailing dates of the past and next months. Clicking the Title that displays the month and year changes the view and shows all the available month of the current year.

Clicking the year will show all the years in the current decade and clicking it even more allows you to easily choose dates way back in the past.

The ListView Control

The ListView control (System.Windows.Forms.ListView) allows you to show a list of items in different views and add icons to each of them. The ListView control is composed of ListViewItems which form a grid-like structure of row and columns. Each ListViewItemhas a label and the every ListViewItem in the first column can have an icon beside them. One common use of the ListView control is to show a list of files and folders to the user.

Exercise:

1. Write a program to change color of Label text control programmatically in Asp .Net.
2. Write a program to Enable-Disable Textbox and change width of TextBox programmatically in Asp .Net
3. Creating a simple customer screen which takes customer name, collage name, city, gender, and hobby.

Review Question:

1. Explain what is happening in following Controls?
   1. Calender 2. AdRotator 3. Gridview 4. ListBound
2. What is Dialog? Explain following Dialogs with its usage.
   1) FolderBrowserDialog 2) OpenFileDialog
3. What is Dialog? Explain following Dialogs with its usage.
   1) ColorDialog 2)SaveDialog

ASP.Net Web Pages

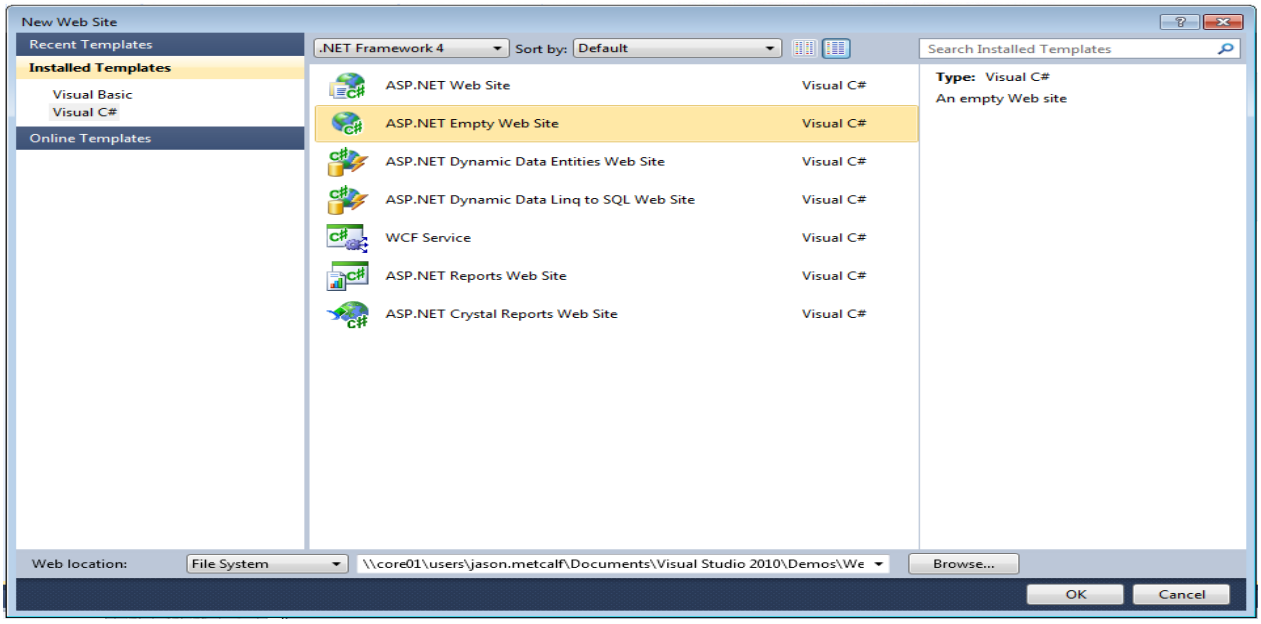
Practical No: Date:

Aim: To study about asp.net web pages (html server controls, asp.net server controls)

Theory:

STARTING A NEW WEBSITE

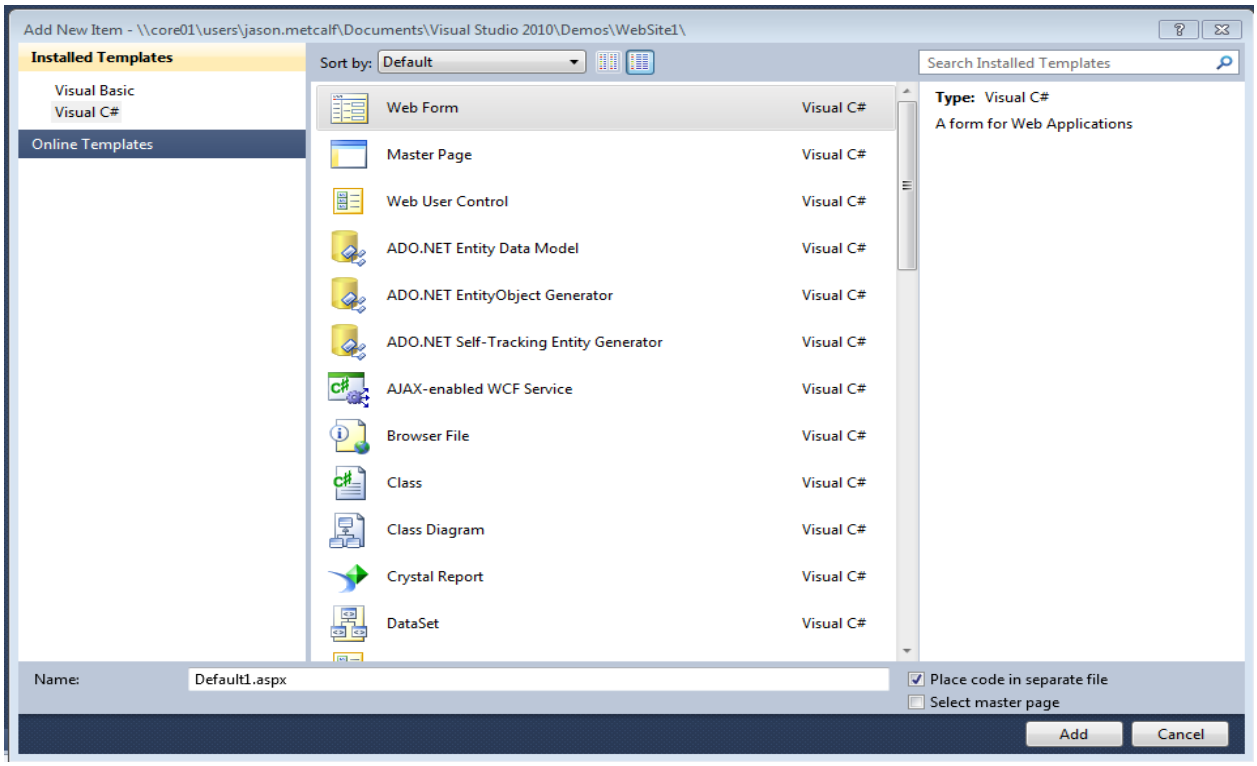
1. Open Visual Web Developer 2010 and in the file menu select the new website option from the dropdown menu or on the start page.
2. When the new website dialogue box appears, select **installed templates** and then select **visual C#** from the dropdown list to set the programing language. Visual studio comes with two pre-installed "code-behind" programming languages C# and Visual Basic. This tutorial only covers the C# language.
3. Select **Empty ASP.NET** Website from the list in the center. This will start your project without the default webpage. When starting a new website, visual studio gives you the option of either creating a blank website or opening a pre-structured website that has a default style and master page already applied to it.



4. In the Web location box, select the **browse** option and navigate to the folder location where you want to save the files to your project. For now, leave the name of the website as its default name (Example: **Website1**).
   You can rename your file by deleting the default (…/website1) at the end of the file source and replacing it with a filename of your choice before saving.
5. Click **OK** and Visual Web developer will generate the project.

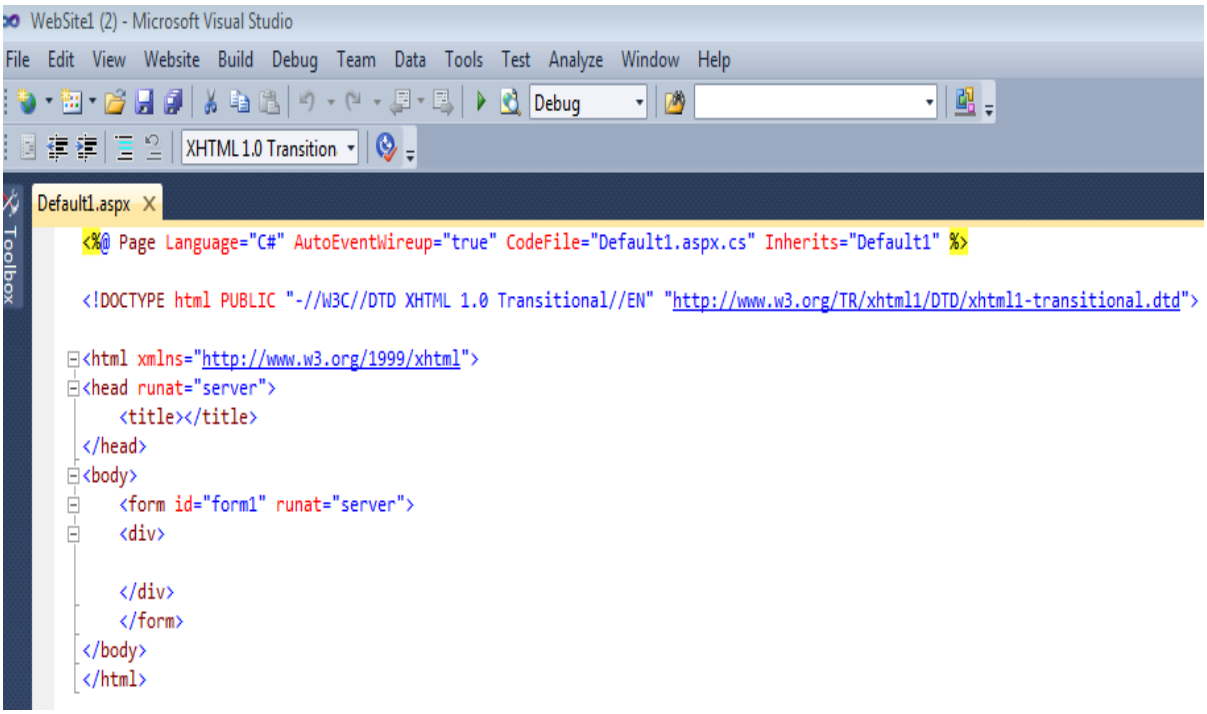
CREATING A BASIC WEB PAGE

1. In the solution explorer, right click the name of the website (Example:C:\…\…website1) and select Add New Item from the list.
2. When the Add New Item display opens, select Web Form from the installed templates list. Take look at the installed templates list. These are all of the pre- installed templates that can be used to create dynamic web applications.



3. In the name box type Home.aspx and make sure the place code in separate file box is checked, and the select master page box is unchecked.

   Checking the "place code in separate file" box will tell visual studio to create the web page with a "code behind" page, which is a separate page that is attached to the web form when you create an event handler. Basically this is where the C# or Visual basic languages will come into play. Leaving the box unchecked will tell visual studio to create an inline code on the same page. The difference between the two is really a matter of preference but for the purposes of showing the difference between the pages; we will create a code-behind page.



4. Click **Add**.

Congratulations, you just made your first webpage in visual studio! You will notice that the page opens automatically in source view and there is already some code on the page.

CONTROLS

Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

ASP.NET uses five types of web controls:

- HTML controls
- HTML Server controls
- ASP.NET Server controls
- ASP.NET Ajax Server controls
- User controls and custom controls

The HTML server controls are basically the standard HTML controls enhanced to enable server side processing. The HTML controls such as the header tags, anchor tags, and input elements are not processed by the server but are sent to the browser for display.

They are specifically converted to a server control by adding the attribute runat="server" and adding an id attribute to make them available for server-side processing.
For example, consider the HTML input control:
<input type="text" size="40">

It could be converted to a server control, by adding the runat and id attribute:

<input type="text" id="testtext" size="40" runat="server">

ADVANTAGES OF USING HTML SERVER CONTROLS

Although ASP.NET server controls can perform every job accomplished by the HTML server controls, the later controls are useful in the following cases:

- Using static tables for layout purposes.
- Converting a HTML page to run under ASP.NET.

ASP.NET server controls are the primary controls used in ASP.NET. These controls can be grouped into the following categories:

- **Validation controls** - These are used to validate user input and they work by running client-side script.
- **Data source controls** - These controls provides data binding to different data sources.
- **Data view controls** - These are various lists and tables, which can bind to data from data sources for displaying.
- **Personalization controls** - These are used for personalization of a page according to the user preferences, based on user information.
- **Login and security controls** - These controls provide user authentication.
- **Master pages** - These controls provide consistent layout and interface throughout the application.
- **Navigation controls** - These controls help in navigation. For example, menus, tree view etc.
- **Rich controls** - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

```
<asp:controlType  ID ="ControlID"
                  runat="server"
                  Property1=value1 [Property2=value2] />
```

PROPERTIES OF THE SERVER CONTROLS

ASP.NET server controls with a visual aspect are derived from the WebControl class and inherit all the properties, events, and methods of this class.

The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, PlaceHolder control or XML control.

ASP.NET server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.

Exercise

1. Create a page in ASP.Net using C#, using HTML Server controls that take use name, address, and city, state and country name from the user and display it.
2. Create a page in ASP.Net using C#, to add a list box control to a Web forms page through coding which prompts for adding or deleting Name of fruits through indexing. The list box control properties should be user defined i.e. color, font etc.of the list box control

Review Question:

1. Differentiate ASP and ASP.net
2. Explain any three HTML server controls
3. Explain any three ASP.net control

Validation Control

Practical No: Date:

Aim: To study about asp.net validation controls

Theory:

ASP.NET VALIDATION CONTROLS

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.
ASP.NET provides the following validation controls:

- Required Field Validator
- Range Validator
- Compare Validator
- Regular Expression Validator
- Custom Validator
- Validation Summary

BASE VALIDATOR CLASS

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods.

REQUIRED FIELD VALIDATOR CONTROL

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
  runat="server" ControlToValidate ="ddlcandidate"
  ErrorMessage="Please choose a candidate"
  InitialValue="Please choose a candidate">
  </asp:RequiredFieldValidator>
```

RANGE VALIDATOR CONTROL

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
  ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
  MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

COMPARE VALIDATOR CONTROL

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
  ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

REGULAR EXPRESSION VALIDATOR

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

Quantifiers could be added to specify number of times a character could appear.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
  ValidationExpression="string" ValidationGroup="string">
</asp:RegularExpressionValidator>
```

CUSTOM VALIDATOR

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
  ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">
  </asp:CustomValidator>
```

VALIDATION SUMMARY

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary** : shows the error messages in specified format.
- **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
  DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

VALIDATION GROUPS

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup*property.

Exercise:

1. Create a page in ASP.Net using C# that take Name, Password, Email add. & age from the user. Put appropriate validation. Show the summary of invalid validation.
2. Create a page in ASP.Net using C# to create a custom validation control that Check even number.
3. Create a page in ASP.Net using C# that displays registration form. Fields are FirstName, Last Name, Email, Password, repass, Age (dd-mm-yyyy), Ph. No., address, city, with appropriate validation controls such as email validation, city to choose from combo box options.

Review Question:

1. Explain Required field validator
2. What is the difference between required field validator and regular expression validator?
3. What is the use of validation summary?

Data Control

Practical No: Date:

Aim: To study about grid view control.

Theory:

GridView control

A recurring task in software development is to display tabular data. ASP.NET provides a number of tools for showing tabular data in a grid, including the **GridView** control. With the **GridView** control, you can display, edit, and delete data from many different kinds of data sources, including databases, XML files, and business objects that expose data.

To add a GridView control to a page

- Drag the **GridView** control from the **Toolbox** task pane to your page.

After you add a **GridView** control, you can specify a data source for the control.

To bind the GridView control to a data source

1. In **Design** view, right-click the **GridView** control, and then click **Show Common Control Tasks**.

2. On the **Common DropDownList Tasks** menu, click an existing data source or **<New Data Source...>** in the **Choose Data Source** dropdown.

3. If you choose <New Data Source...>, configure a new data source in the Data Source Configuration Wizard.

You can specify the layout, color, font, and alignment of the **GridView** control's rows. You can specify the display of text and data contained in the rows. Additionally, you can specify whether the data rows are displayed as items, alternating items, selected items, or edit-mode items. The **GridView** control also allows you to specify the format of the columns.

To specify GridView control display options

1. Click the **GridView** control in **Design** view to select it, right-click the control, and choose **Properties** from the shortcut menu to open the**Tag Properties** task pane.

2. Specify the styles you want for the various **GridView** elements in the **Styles** category of the **Properties** task pane. For example, under the **Font** property group in the **RowStyle** property group, set the **Name** property to the font you want for the items in the rows in the **GridView**.

EDITING AND DELETING DATA USING THE GRIDVIEW CONTROL

By default, the **GridView** control displays data in read-only mode. However, the control also supports an edit mode in which it displays a row that contains editable controls such as TextBox or CheckBoxcontrols. You can also configure the **GridView** control to display a **Delete** button that users can click to delete the corresponding record from the data source.

The **GridView** control can automatically perform editing and deleting operations with its associated data source, which allows you to enable editing behavior without writing code. Alternatively, you can control the process of editing and deleting data programmatically, such as in cases where the GridView control is bound to a read-only data source control.

To specify GridView control data editing options

1. In **Design** view, right-click the **GridView** control, and then click **Show Common Control Tasks**.

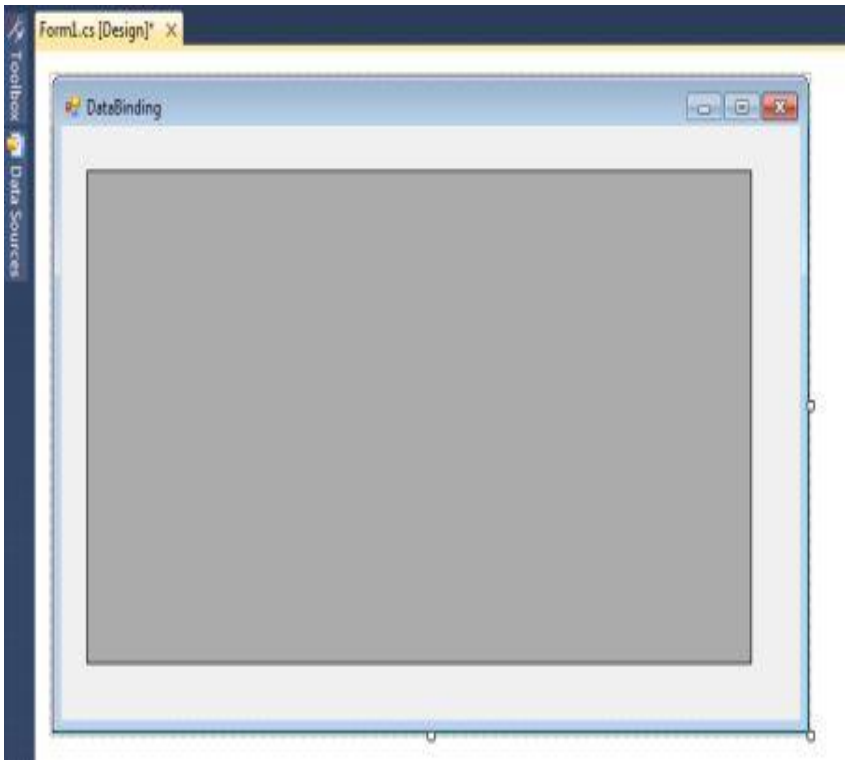
2. Check the functionality you want:

    - **Enable paging** — Displays only a subset of the records on a page and allows a user to move from page to page to display more records. **Enable sorting** — Allows a user to sort the records from the database. The GridView control supports sorting by a single column without requiring any programming. You can further customize the sort functionality of the GridView control by using the sort event and providing a sort expression.
    - **Enable editing** — Allows a user to make changes to the records.
    - **Enable deleting** — Allows a user to delete rows from the database.
    - **Enable selection** — Allows a user to select rows. You specify the look of selected rows by setting the styles in the SelectedRowStyle group in the Properties task pane.

DATABINDING WITH DATAGRIDVIEW IN ADO.NET

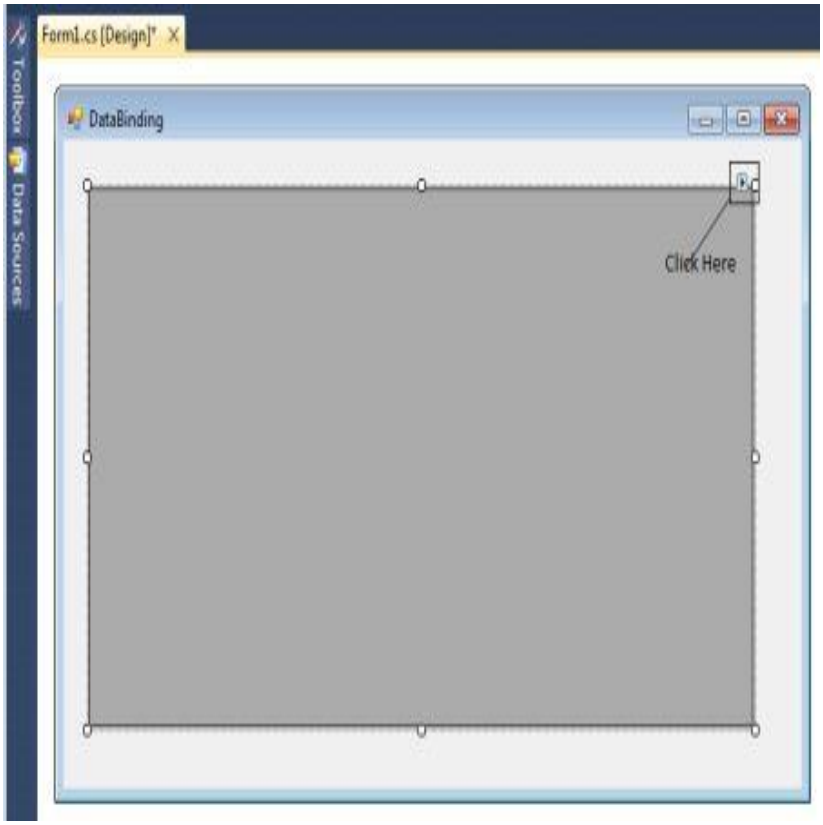
DataGridView is very powerful and flexible control for displaying records in a tabular (row-column) form. Here I am describing a different way of databinding with a DataGridView control.

Take a windows Form Application -> take a DataGridView control.

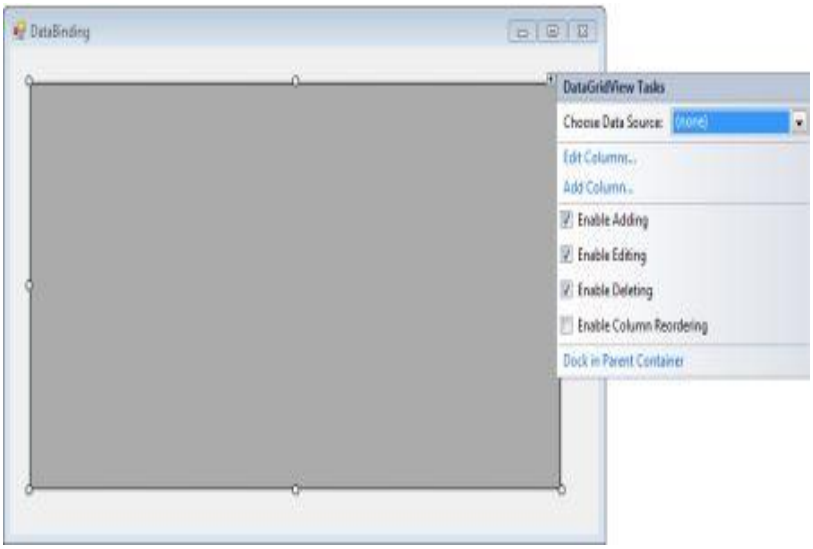


Follow the given steps.

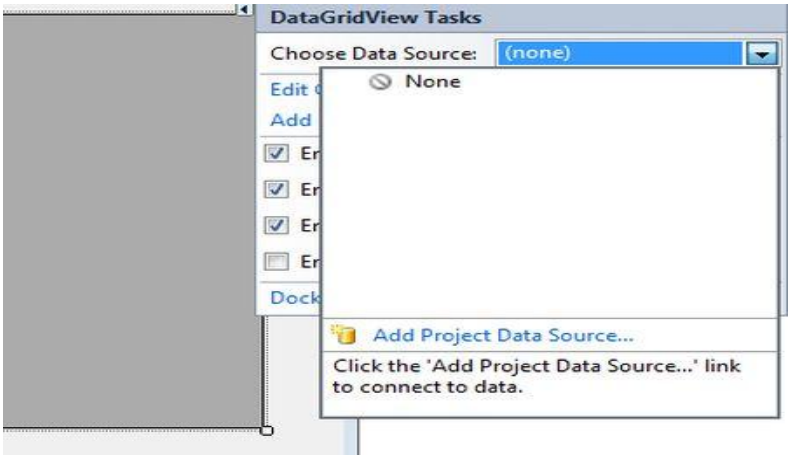
Step 1 : Select DataGridView control and click at smart property. Look at the following figure.



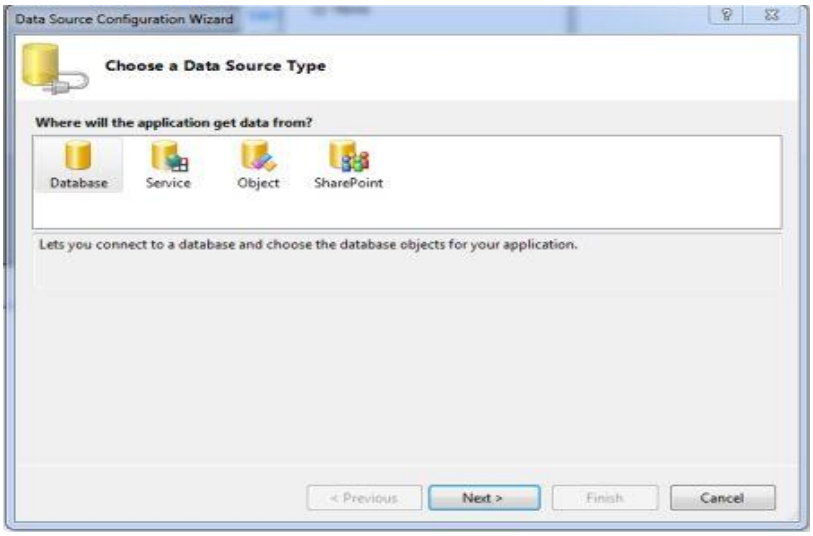
Step 2 : After clicking, a pop-up window will be open.



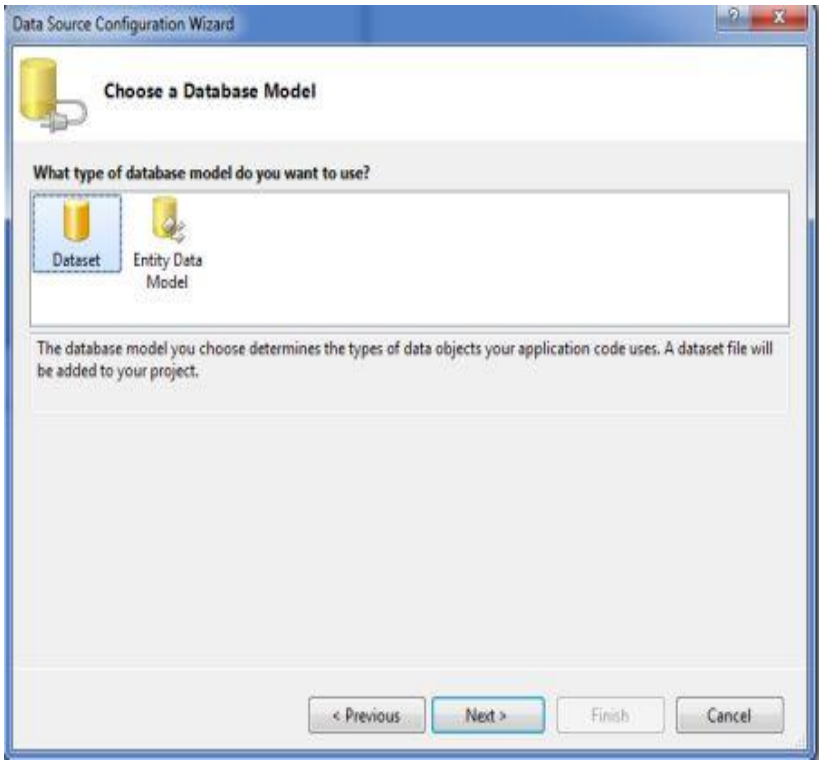
Step 3 : Click ComboBox



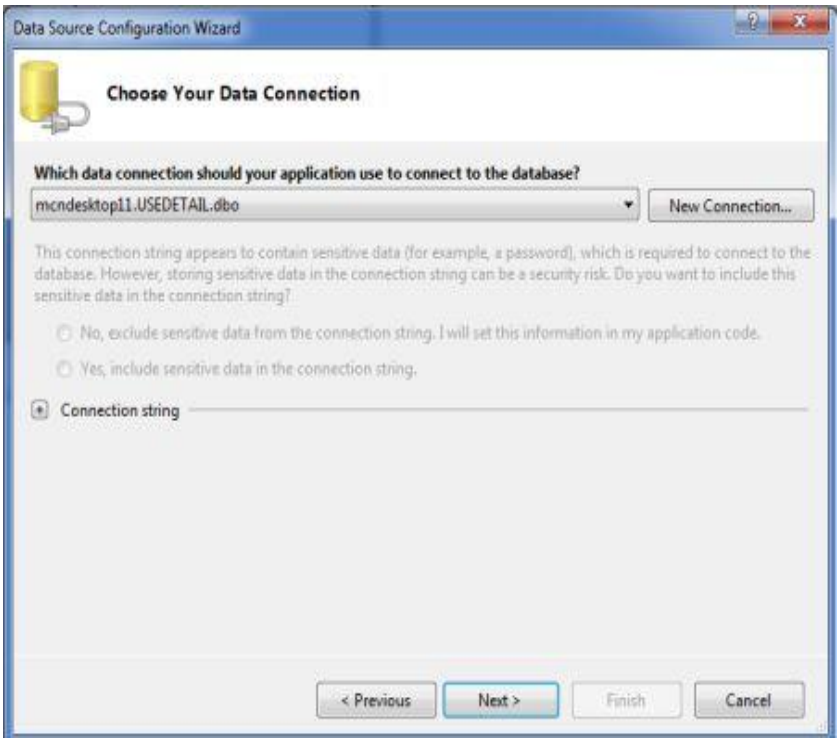
Step 4 : Click at Add Project Data Source (Look at above figure). A new window will be opened to choose Data Source Type.



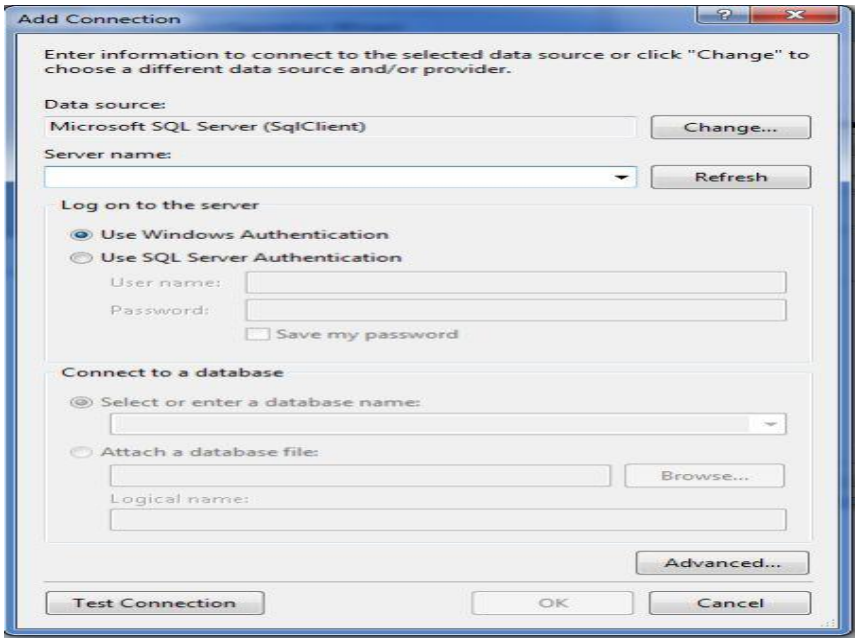
Step 5 : Choose Database (By default it is selected) and click the next button. A new window will be open to Database Model.



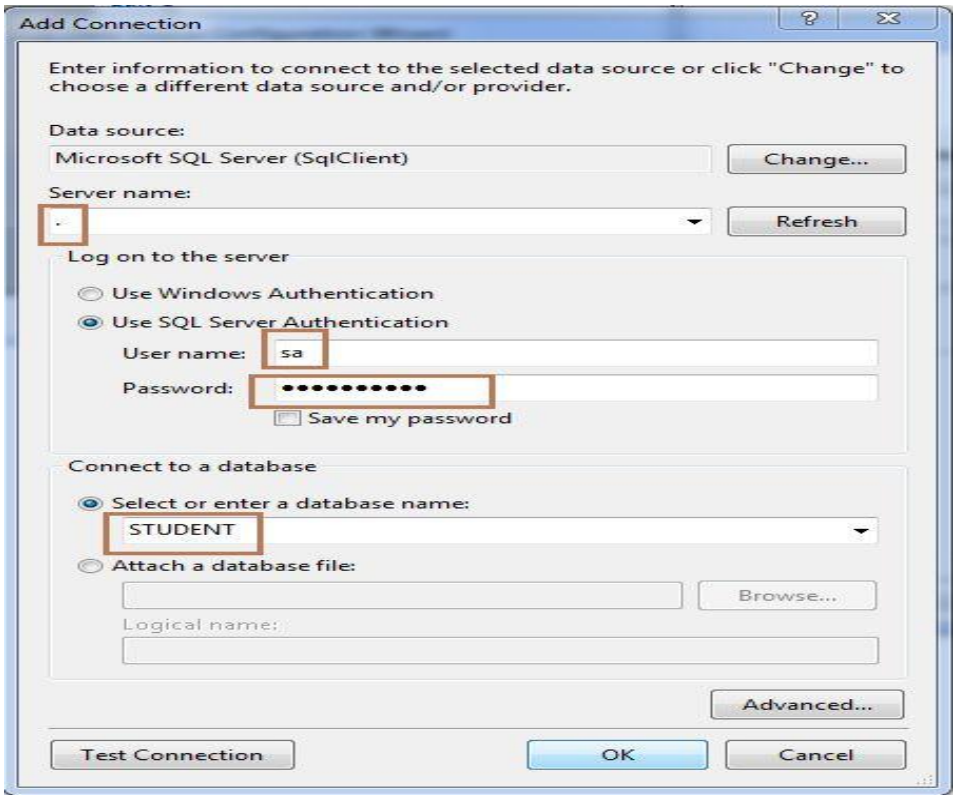
Step 6 : Select DataSet (By default it is selected) and click the next button. A new window will be open.



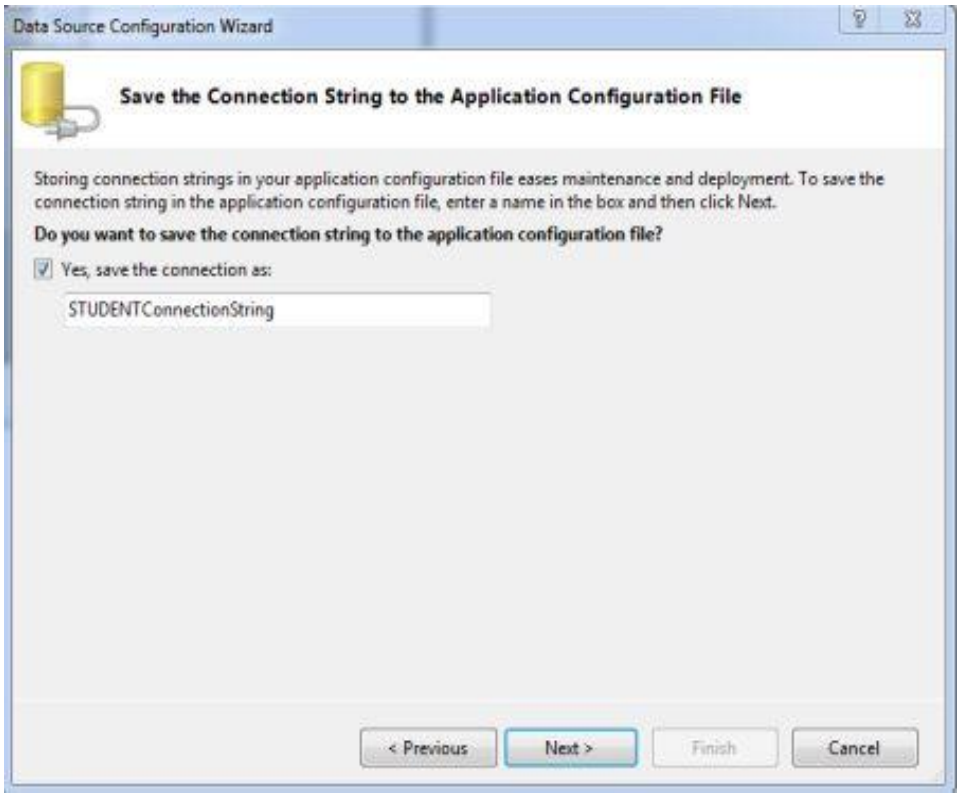
Step 7 : Click at New Connection button.



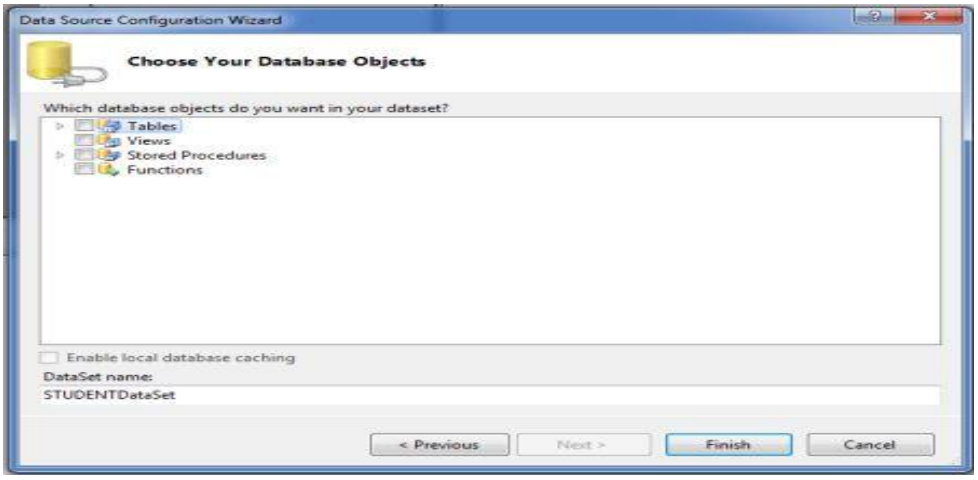
Step 8 : Write Server name, User name and Password of your SQL server and select Database name. Look at the following figure.



Step 9 : Click "ok" button. After clicking ok button, you will reach the Data Source Configuration Wizard.

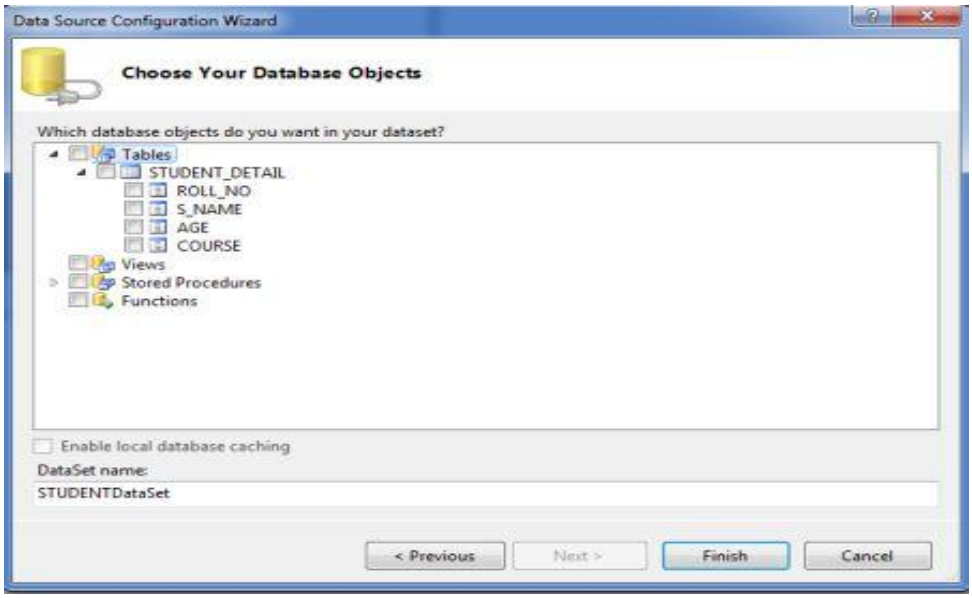


Step 10 : Click the next button

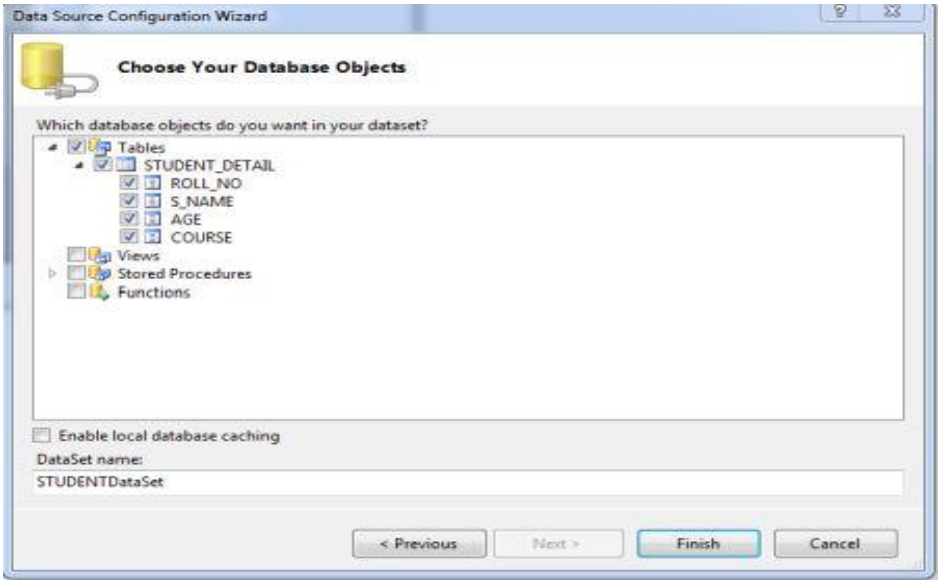


Step 11 : Click on Table to explore all tables of your Database.

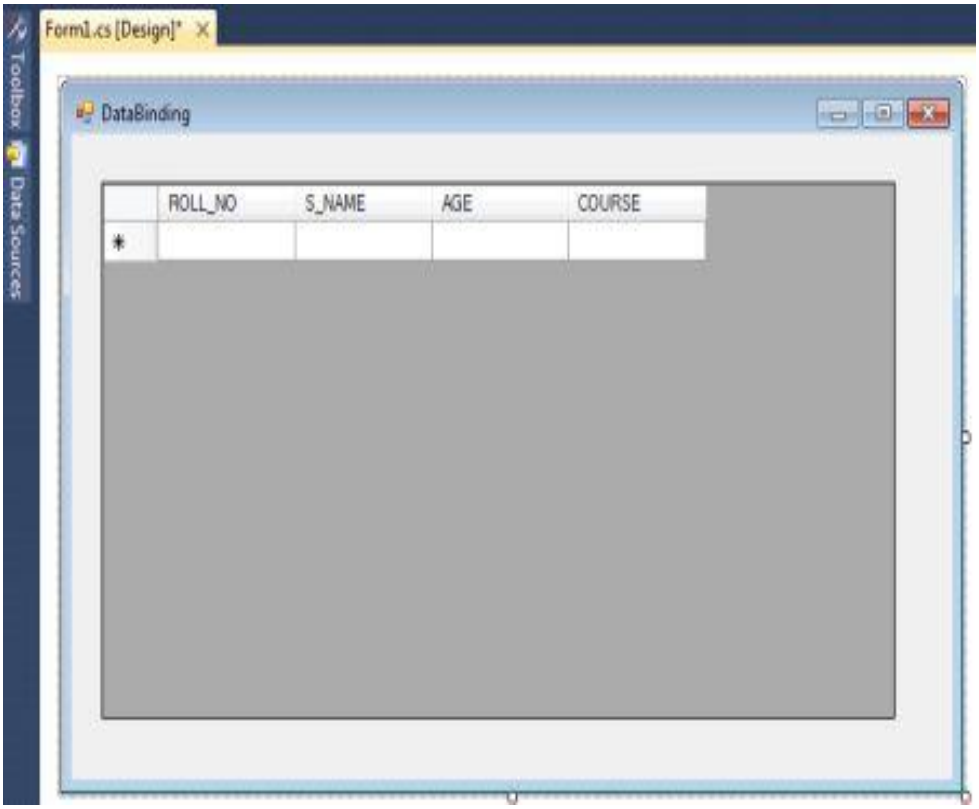
Step 12 : Click on the selected Database table to explore all columns



Step 13 : Check the CheckBox to select columns

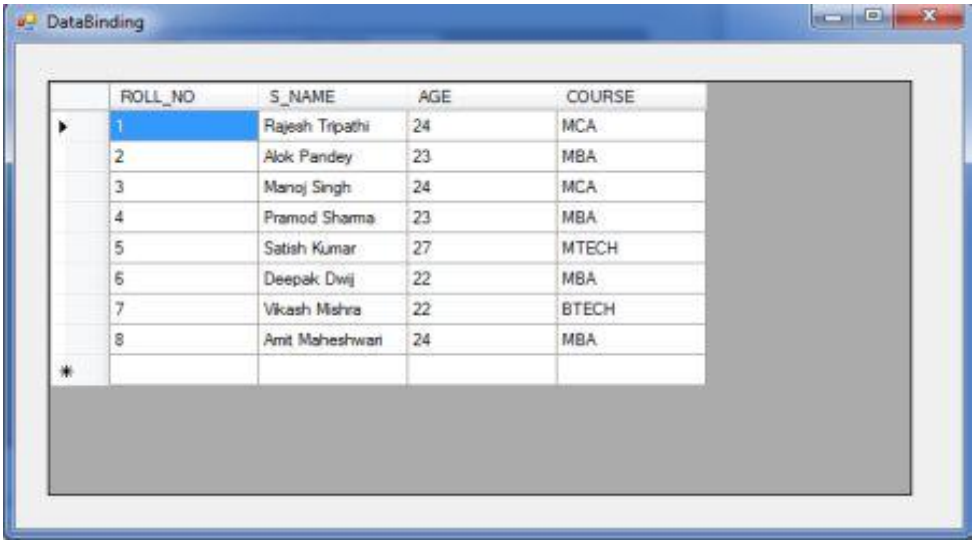


Step 14 : Click the Finish button. You will note that the DataGridView will show all columns of the table (Here, "Student_detail").



Run the application.

Output



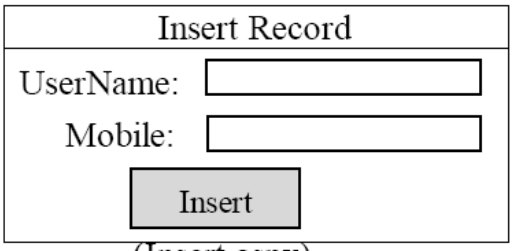
Now we bind the DataGridView with database by code. Take another DataGridView control and write the following code on the form load event.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace DatabindingWithdataGridView
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      SqlDataAdapter dadapter;
      DataSet dset;
      string connstring = "server=.;database=student;user=sa;password=wintellect";
      private void Form1_Load(object sender, EventArgs e)
      {
         dadapter = new SqlDataAdapter("select * from student_detail", connstring);
         dset = new System.Data.DataSet();
         dadapter.Fill(dset);
         dataGridView1.DataSource = dset.Tables[0].DefaultView;
      } } }
```

Exercise:

1. Create the table with the given fields.
   FIELD NAME DATA TYPE
   EmpID number
   EmpName varchar
   EmpSal number
   EmpJob varchar
   EmpDeptNo number
   For the given table design a web page to display the employee information from table to grid control.
2. Create a web application using C#.Net to insert a record into Insert.aspx webpage (as shown below) and display the same records in GridView control with update and delete facilities in Update.aspx webpage (as shown below) with following conditions:
   1.Only last 5 digits of Mobile numbers should be displayed in GridView.
   2.Blank Record should not be inserted from Insert.aspx.



(Insert.aspx)

Update Records

UserName	Mobile	Edit	Delete
Vidhi	12627	Edit	Delete
Aadi	99210	Edit	Delete
Vibha	59997	Edit	Delete
Vijay	50096	Edit	Delete

1 2 3

(Update.aspx)

Review Question:

1. Explain Ado.net Architecture.
2. Explain the process to add database in asp.net website.
3. Explain insert update and delete queries in detail.

ADO.Net Application

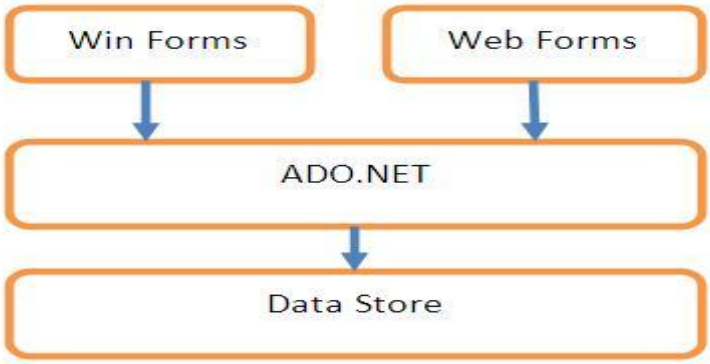
Practical No: Date:

Aim: To study about ADO.Net Application

Theory:

ADO.NET is a set of classes that comes with the Microsoft .NET framework to facilitate data access from managed languages. ADO.NET has been in existence for a long time and it provides a comprehensive and complete set of libraries for data access. The strength of ADO.NET is firstly that it lets applications access various types of data using the same methodology. If I know how to use ADO.NET to access a SQL Server database then the same methodology can be used to access any other type of database (like Oracle or MS Access) by just using a different set of classes. Secondly, ADO.NET provides two models for data access: a connected model where I can keep the connection with the database and perform data access, and another way is to get all the data in ADO.NET objects that let us perform data access on disconnected objects.

Note: Many developers and development houses are now using ORMs to perform data access instead of using ADO.NET. ORMs provide a lot of data access functionality out of the box and relieves users from writing mundane data access code again and again. Still, I think that knowing and understanding ADO.NET is crucial as a .NET developer as it gives a better understanding of the data access methodologies. Also, there are many development houses that are still using ADO.NET.



The diagram above shows that ADO.NET can be used with any kind of application, i.e., it can be used from a Windows Forms application, an ASP.NET application, or from a WPF and/or Silverlight application. Also, the data store underneath can be any data store, SQL Server, Access, or Oracle. It is just a matter of using the right set of classes specific to that data store and the methodology will remain the same.

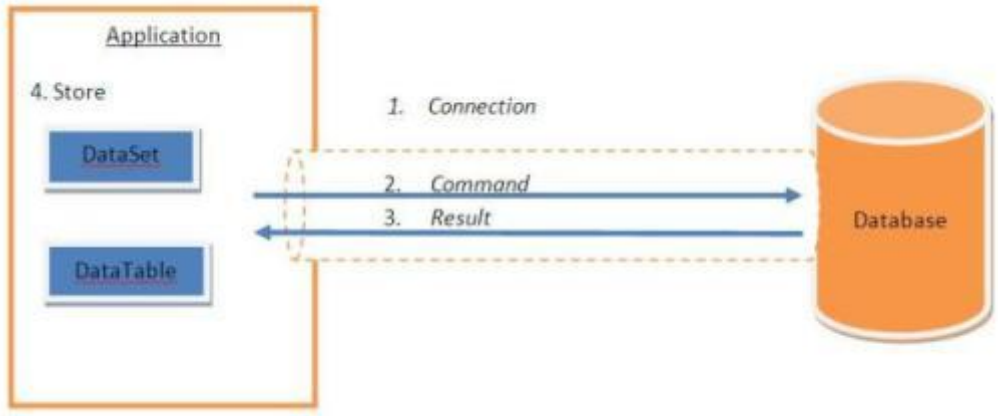
USING THE CODE

Let us try to understand a few ADO.NET classes and methodologies by writing a small web application. This application uses a sample database from Microsoft (subset of the Pubs database) and we will use this database for understanding the various classes and methods of ADO.NET. We will be using ADO.NET classes specific to SQL Server but once it is understood, the basic philosophy remains the same and can be applied with any data store.

Before jumping into the code, we will have to understand some of the important objects of ADO.NET. In a typical scenario requiring data access, we need to perform four major tasks:

1. Connecting to the database
2. Passing the request to the database, i.e., a command like select, insert, or update.
3. Getting back the results, i.e., rows and/or the number of rows effected.
4. Storing the result and displaying it to the user.

This can be visualized as:



So now we need to understand how we can achieve these functionalities using ADO.NET.

CREATING A SQLCONNECTION OBJECT

A SqlConnection is an object, just like any other C# object. Most of the time, you just declare and instantiate the SqlConnection all at the same time, as shown below:

```
SqlConnection conn = new SqlConnection(
"Data Source=(local);Initial   Catalog=Northwind; Integrated Security=SSPI");
```

The SqlConnection object instantiated above uses a constructor with a single argument of type string This argument is called a connection string. Table 1 describes common parts of a connection string.

Table 1. ADO.NET Connection Strings contain certain key/value pairs for specifying how to make a database connection. They include the location, name of the database, and security credentials.

Connection String Parameter Name	Description
Data Source	Identifies the server. Could be local machine, machine domain name, or IP Address.
Initial Catalog	Database name.
Integrated Security	Set to SSPI to make connection with user's Windows login
User ID	Name of user configured in SQL Server.
Password	Password matching SQL Server User ID.

Integrated Security is secure when you are on a single machine doing development. However, you will often want to specify security based on a SQL Server User ID with permissions set specifically for the application you are using. The following shows a connection string, using the User ID and Password parameters:

```
SqlConnection conn = new SqlConnection(
"Data Source=DatabaseServer;Initial Catalog=Northwind;User ID=YourUserID;
 Password=YourPassword");
```

Notice how the Data Source is set to DatabaseServer to indicate that you can identify a database located on a different machine, over a LAN, or over the Internet. Additionally, User ID and Password replace the Integrated Security parameter.

Using a SqlConnection

The purpose of creating a SqlConnection object is so you can enable other ADO.NET code to work with a database. Other ADO.NET objects, such as a SqlCommand and a SqlDataAdapter take a connection object as a parameter. The sequence of operations occurring in the lifetime of a SqlConnection are as follows:

1. Instantiate the SqlConnection.
2. Open the connection.
3. Pass the connection to other ADO.NET objects.
4. Perform database operations with the other ADO.NET objects.
5. Close the connection.

We've already seen how to instantiate a SqlConnection. The rest of the steps, opening, passing, using, and closing are shown in Listing 1.

Listing 1. Using a SqlConnection

```
using System;
using System.Data;
using System.Data.SqlClient;
/// <summary>
/// Demonstrates how to work with SqlConnection objects
/// </summary>
```

```
class SqlConnectionDemo
{
   static void Main()
   {
      // 1. Instantiate the connection
      SqlConnection conn = new SqlConnection(
         "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");

      SqlDataReader rdr = null;

      try
      {
         // 2. Open the connection
         conn.Open();

         // 3. Pass the connection to a command object
         SqlCommand cmd = new SqlCommand("select * from Customers", conn);

         //
         // 4. Use the connection
         //

         // get query results
         rdr = cmd.ExecuteReader();

         // print the CustomerID of each record
         while (rdr.Read())

        {
            Console.WriteLine(rdr[0]);
         }
      }
      finally
      {
         // close the reader
         if (rdr != null)
         {
            rdr.Close();
         }

         // 5. Close the connection
         if (conn != null)
         {
            conn.Close();
         }
```

```
    }
  }
}
```

As shown in Listing 1, you open a connection by calling the Open() method of the SqlConnection instance, conn. Any operations on a connection that was not yet opened will generate an exception. So, you must open the connection before using it.

Before using a SqlCommand, you must let the ADO.NET code know which connection it needs. In Listing 1, we set the second parameter to the SqlCommand object with the SqlConnection object, conn. Any operations performed with the SqlCommand will use that connection.

The code that uses the connection is a SqlCommand object, which performs a query on the Customers table. The result set is returned as a SqlDataReader and the while loop reads the first column from each row of the result set, which is the CustomerID column. We'll discuss the SqlCommand and SqlDataReader objects in later lessons. For right now, it is important for you to understand that these objects are using the SqlConnection object so they know what database to interact with.

When you are done using the connection object, you must close it. Failure to do so could have serious consequences in the performance and scalability of your application. There are a couple points to be made about how we closed the connection in Listing 1: the Close() method is called in a finally block and we ensure that the connection is not null before closing it

Exercise:

1. Create a Login Module which adds Username and Password in the database. Username in the database should be a primary key.
2. Create a web application to insert records inside the SQL database table having following fields(DeptId, DeptName, EmpName, Salary). Update the salary for any one employee and increment it to 10% of the present salary. Perform delete operation on 1 row of the database table.
3. Write an ASP.NET code to fetch the data from SQL database table Student (having

Review Question:

1. Explain SqlConnection and SqlCommand.
2. Explain ExecuteReader.
3. What is the difference between ExecuteReader , ExecuteNonQuery and ExecuteScalar?

State Management

Practical No: Date:

Aim: To study about state management

Theory:

Hyper Text Transfer Protocol (HTTP) is a stateless protocol. When the client disconnects from the server, the ASP.NET engine discards the page objects. This way, each web application can scale up to serve numerous requests simultaneously without running out of server memory.

However, there needs to be some technique to store the information between requests and to retrieve it when required. This information i.e., the current value of all the controls and variables for the current user in the current session is called the State.

ASP.NET manages four types of states:

- View State
- Control State
- Session State
- Application State

View State

The view state is the state of the page and all its controls. It is automatically maintained across posts by the ASP.NET framework.

When a page is sent back to the client, the changes in the properties of the page and its controls are determined, and stored in the value of a hidden input field named _VIEWSTATE. When the page is again posted back, the _VIEWSTATE field is sent to the server with the HTTP request.

The view state could be enabled or disabled for:

- **The entire application** by setting the EnableViewState property in the <pages> section of web.config file.
- **A page** by setting the EnableViewState attribute of the Page directive, as <%@ Page Language="C#" EnableViewState="false" %>
- **A control** by setting the Control.EnableViewState property.

It is implemented using a view state object defined by the StateBag class which defines a collection of view state items. The state bag is a data structure containing attribute value pairs, stored as strings associated with objects.

The StateBag class has the following properties:

Properties	**Description**
Item(name)	The value of the view state item with the specified name. This is the default property of the StateBag class.
Count	The number of items in the view state collection.
Keys	Collection of keys for all the items in the collection.
Values	Collection of values for all the items in the collection.

The StateBag class has the following methods:

Methods	**Description**
Add(name, value)	Adds an item to the view state collection and existing item is updated.
Clear	Removes all the items from the collection.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows it to free resources and perform other cleanup operations.
GetEnumerator	Returns an enumerator that iterates over all the key/value pairs of the StateItem objects stored in the StateBag object.
GetType	Gets the type of the current instance.
IsItemDirty	Checks a StateItem object stored in the StateBag object to evaluate whether it has been modified.
Remove(name)	Removes the specified item.
SetDirty	Sets the state of the StateBag object as well as the Dirty property of each of the StateItem objects contained by it.
SetItemDirty	Sets the Dirty property for the specified StateItem object in the StateBag object.
ToString	Returns a string representing the state bag object.

Control State

Control state cannot be modified, accessed directly, or disabled.

Session State

When a user connects to an ASP.NET website, a new session object is created. When session state is turned on, a new session state object is created for each new request. This session state object becomes part of the context and it is available through the page.

Session state is generally used for storing application data such as inventory, supplier list, customer record, or shopping cart. It can also keep information about the user and his preferences, and keep the track of pending operations.

Sessions are identified and tracked with a 120-bit SessionID, which is passed from client to server and back as cookie or a modified URL. The SessionID is globally unique and random.

The session state object is created from the HttpSessionState class, which defines a collection of session state items.

The HttpSessionState class has the following properties:

Properties	**Description**
SessionID	The unique session identifier.
Item(name)	The value of the session state item with the specified name. This is the default property of the HttpSessionState class.
Count	The number of items in the session state collection.
TimeOut	Gets and sets the amount of time, in minutes, allowed between requests before the session-state provider terminates the session.

The HttpSessionState class has the following methods:

Methods	**Description**
Add(name, value)	Adds an item to the session state collection.
Clear	Removes all the items from session state collection.
Remove(name)	Removes the specified item from the session state collection.
RemoveAll	Removes all keys and values from the session-state collection.
RemoveAt	Deletes an item at a specified index from the session-state collection.

The session state object is a name-value pair to store and retrieve some information from the session state object. You could use the following code for the same:

```
voidStoreSessionInfo()
{
String fromuser =TextBox1.Text;
Session["fromuser"]= fromuser;
}
voidRetrieveSessionInfo()
{
String fromuser =Session["fromuser"];
Label1.Text= fromuser;
}
```

The above code stores only strings in the Session dictionary object, however, it can store all the primitive data types and arrays composed of primitive data types, as well as the DataSet, DataTable, HashTable, and Image objects, as well as any user-defined class that inherits from the ISerializable object.

Application State

The ASP.NET application is the collection of all web pages, code and other files within a single virtual directory on a web server. When information is stored in application state, it is available to all the users.

To provide for the use of application state, ASP.NET creates an application state object for each application from the HTTPApplicationState class and stores this object in server memory. This object is represented by class file global.asax.

Application State is mostly used to store hit counters and other statistical data, global application data like tax rate, discount rate etc. and to keep the track of users visiting the site.

The HttpApplicationState class has the following properties:

Properties	**Description**
Item(name)	The value of the application state item with the specified name. This is the default

	property of the HttpApplicationState class.
Count	The number of items in the application state collection.

The HttpApplicationState class has the following methods:

Methods	**Description**
Add(name, value)	Adds an item to the application state collection.
Clear	Removes all the items from the application state collection.
Remove(name)	Removes the specified item from the application state collection.
RemoveAll	Removes all objects from an HttpApplicationState collection.
RemoveAt	Removes an HttpApplicationState object from a collection by index.
Lock()	Locks the application state collection so only the current user can access it.
Unlock()	Unlocks the application state collection so all the users can access it.

Application state data is generally maintained by writing handlers for the events:

- Application_Start
- Application_End
- Application_Error
- Session_Start
- Session_End

Exercise:

1. Write ASP.Net program to Store Objects in Session State and Storing Session State in SQL Server.

Review Question:

1. What is cookies?
2. Explain the types of cookies?
3. What is the need of state management techniques?