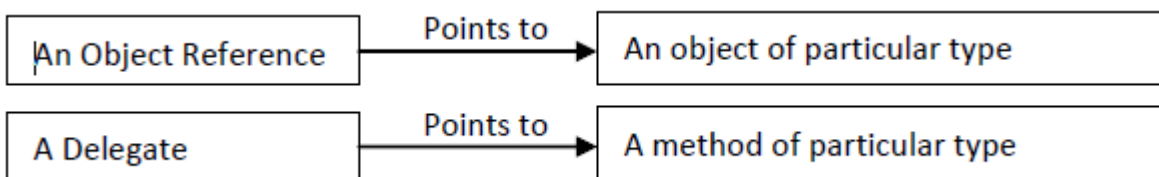


Chapter-4

ADVANCED C#

DELEGATES

- ☞ A delegate is an object that knows how to call a method. A delegate type defines the kind of method that delegate instances can call. Specifically, it defines the method's return type and its parameter types. C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime. All delegates are implicitly derived from the **System.Delegate** class



Syntax:

<Access Modifier> Delegate <Return Type> <Name of Delegate> (<Parameters>)

- ☞ Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate. To handle delegate type we must follow following five steps.

Step1: Create some method that we want to call

Step2: Create a delegate type.

Step3: Create delegate instance

Step4: Point the delegate to method.

Step5: Call a method via delegate.

Example:

```
using System;
```

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

```
using System.Linq;
```

```
using System.Text;
```

```
namespace SimpleExampleDelegate
```

```
{
```

```
    class Program
```

```
    {
```

```
        // Step2: Create delegate Type
```

```
        public delegate void delegateMethod();
```

```
        static void Main(string[] args)
```

```

{
    //Step3 Create Delegate instance
    //Step4: Points the delegate to method
    delegateMethod del = new delegateMethod(printString);
    //Step5: Call a method via Dlegate
    del.Invoke();
    Console.ReadKey();
}
//Step1. Create method to be called via delegate
public static void printString()
{
    Console.WriteLine("this is my first delegate");
}
}
}

```

Output:

this is my first delegate

Example-2:

Example program to demonstrate the delegate.

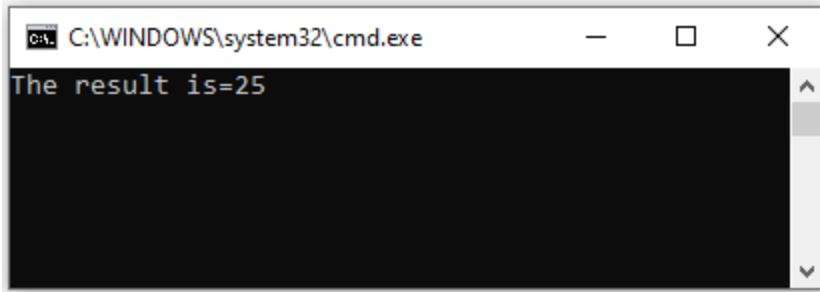
```

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

namespace TestDelegate
{
    public delegate int MyDelegate(int x);
    internal class DelegateTest
    {
        public int MyMethod(int x)
        {
            return x * x;
        }
        static void Main(string[] args)
        {
            DelegateTest dt = new DelegateTest();
            int res= dt.MyMethod(5);
            Console.WriteLine("The result is=" + res);
            Console.ReadKey();
        }
    }
}

```

Out Put:



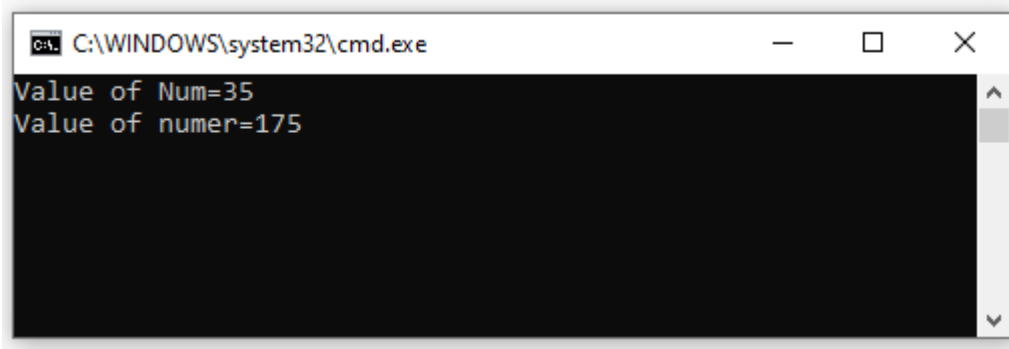
Example-3

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

namespace TestDelegate
{
    delegate int NumberChanger(int n);
    internal class DelegateTest
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num = num + p;
            return num;
        }
        public static int MultNum(int q)
        {
            num = num * q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }

        static void Main(string[] args)
        {
            NumberChanger nc1 = new NumberChanger(AddNum); //Create delegate instance
            NumberChanger nc2 = new NumberChanger(MultNum);
            //calling the method using delegate object
            nc1(25);
            Console.WriteLine("Value of Num={0}", getNum());
            nc2(5);
            Console.WriteLine("Value of numer={0}", getNum());
            Console.ReadKey();
        }
    }
}
```

Out Put:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The window contains two lines of text: 'Value of Num=35' and 'Value of number=175'. The text is white on a black background. There are standard window controls (minimize, maximize, close) in the top right corner.

```
C:\WINDOWS\system32\cmd.exe
Value of Num=35
Value of number=175
```

Multicast Delegates

- ☞ When a delegate is wrapped with more than one method that is known as multicast delegate.
- ☞ In c#, delegates are multicast, which means that they can point to more than one function at a time. They are derived from System.MulticastDelegate class.
- ☞ We can use += and -= assignment operators to implement multi cast delegates.

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

Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called **multicasting** of a delegate. The following program demonstrates multicasting of a delegate:

Example: 1

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

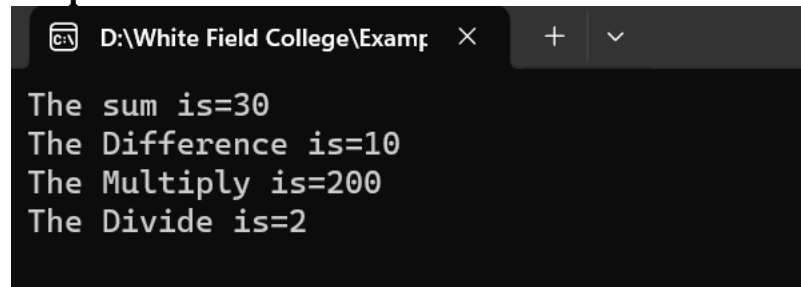
namespace MulticastDelegate
{
    public delegate void Calculate(int n1, int n2);
    internal class Program
    {
        public static void Addition(int n1, int n2)
        {
            int result = n1 + n2;
            Console.WriteLine("The sum is={0}", result);
        }
        public static void Subtraction(int n1, int n2)
        {
            int result = n1 - n2;
            Console.WriteLine("The Difference is={0}", result);
        }
    }
}
```

```

        public static void Multiply(int n1, int n2)
        {
            int result = n1 * n2;
            Console.WriteLine("The Multiply is={0}", result);
        }
        public static void Division(int n1, int n2)
        {
            int result = n1 / n2;
            Console.WriteLine("The Divide is={0}", result);
        }
        static void Main(string[] args)
        {
            Calculate obj = new Calculate(Addition);
            obj += Subtraction;
            obj += Multiply;
            obj += Division;
            obj(20, 10);
            Console.ReadKey();
        }
    }
}

```

Output:



```

D:\White Field College\Examf
The sum is=30
The Difference is=10
The Multiply is=200
The Divide is=2

```

Example:

```

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

namespace DelegateExample1
{
    class Program
    {
        public delegate void delegatemethod(string s);
        static void Main(string[] args)
        {

```

```

        delegatemethod del = new delegatemethod(helloworld);
        del = del + new delegatemethod(Filewrite);
        del("hello world");
        Console.Read();
    }
    public static void helloworld(string str)
    {
        Console.WriteLine(str);
    }
    public static void Filewrite(String str)
    {
        File.WriteAllText("D:\\vc#\\temp.txt",str);
    }
}

```

Delegates Vs Interface in C#

☞ The difference between delegates and interface are as follows:

DELEGATE	INTERFACE
It could be a method only	It contains both methods and parameters.
It can be applied to one method at a time	If a class implements an interface, then it will implement all the methods related to that interface.
If a delegate available in your scope you can use it.	Interface is used when your class implements that interface, otherwise not.
Delegate can be implemented any number of time.	Interface can be implemented only one time.
It is used to handle event.	It is not used to handle event.
When you access the method using delegates you do not require any access to the object of the class where the method is defined.	When you access the method you need the object of the class which implemented an interface.
It does not support inheritance	It supports inheritance.
It created at run time.	It created at compile time.
It can implement any method that provides the same signature with the given delegate.	It the method of interface implemented, then the same name and signature method override.

It can wrap any method whose signature is similar to the delegate and does not consider which from class it belongs.	A class can implement any number of interfaces, but can only override those methods which belongs to the interfaces.
--	--

EVENTS

- Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications.
- Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.
- The class that sends or raises an event is called a **publisher** and a class that receives or handle the event is called "**Subscriber**".
- Following are the key points about Event,
 - ❖ Event handler in C# return void and take two parameters.
 - ❖ The first parameter of Event- Source of Event means publishing object.
 - ❖ The second parameters of Event- Object derived from EventArgs.
 - ❖ The publishers determines when an event is raised and the subscribe determines what action is taken in response.
 - ❖ An event can have so many subscribers.
 - ❖ Events are basically used for the single user action like button click.
 - ❖ If an Event has multiple subscribers then event handlers are invoked synchronously.

How to declaring Events

- ☞ To declare an event inside a class, first of all, you must declare a delegate type for the even as:

public delegate string MyDelegate(string str);

then, declare the event using the **event** keyword –

event MyDelegate delg;

The preceding code defines a delegate named **MyDelegate** and an event named **delg**, which invokes the delegate when it is raised.

- ☞ To declare an event inside a class, first a delegate type for the Event must be declared like below:

public delegate void MyEventHandler(object sender, EventArgs e);

Defining an event is a two-step process.

- ❖ First you need to defining a delegate type that will hold the list of methods to be called when the event is fired.
- ❖ Next, you declare an event using the event keyword.

Example:

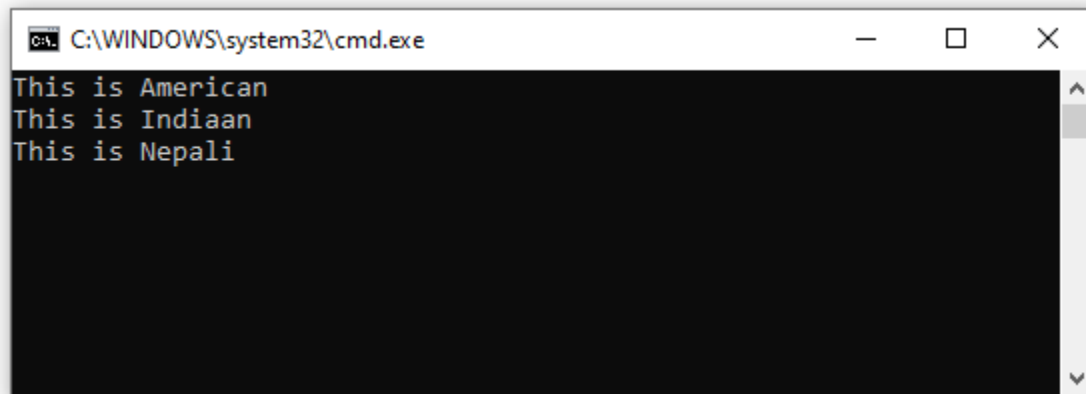
Example program to demonstrate an event to add that is associated to a single delegate **DelEventHandler**.

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

namespace Events
{
    public delegate void DelEventHandler();
    internal class Program
    {
        public static event DelEventHandler add;
        static void USA()
        {
            Console.WriteLine("This is American");
        }
        static void NEPAL()
        {
            Console.WriteLine("This is Nepali");
        }
        static void INDIA()
        {
            Console.WriteLine("This is Indiaan");
        }
        static void Main(string[] args)
        {
            add += new DelEventHandler(USA); //subscribe
            add += new DelEventHandler(INDIA);
            add += new DelEventHandler(NEPAL);
            add.Invoke();
            Console.ReadKey();
        }
    }
}
```

Note: You can just subscribe or unsubscribe the event by += or -= operators and nothing else.

Out put:



```
C:\WINDOWS\system32\cmd.exe
This is American
This is Indiaan
This is Nepali
```

Example-2

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

namespace DelegateEvent
{
    public delegate void Transformer(int x);
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please Enter a number=");
            int i = int.Parse(Console.ReadLine());
            Transformer t; //Create instance
            t = Square; //point to the function
            t += Cuber; //two methods till now
            t.Invoke(i); //invoking the delegate
            NotificationofMethods obj = new NotificationofMethods();
            obj.transformerEvent += User1.Xhandler;
            obj.transformerEvent += User2.Yhandler;
            obj.NotifyOnCell(i);

            Console.ReadKey();
        }

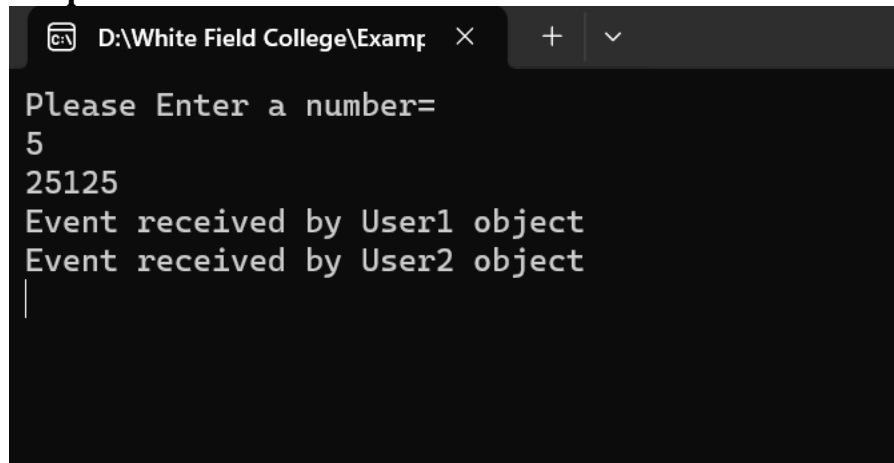
        static void Square(int x)
        {
            Console.Write(x * x);
        }
        static void Cuber(int x)
        {
            Console.WriteLine(x * x * x);
        }
    }
}
class NotificationofMethods
{
    public event Transformer transformerEvent;
    public void NotifyOnCell(int x)
```

```

        {
            if(transformerEvent != null)
            {
                transformerEvent(x);
            }
        }
    }
}
class User1
{
    public static void Xhandler(int x)
    {
        Console.WriteLine("Event received by User1 object");
    }
}
class User2
{
    public static void Yhandler(int x)
    {
        Console.WriteLine("Event received by User2 object");
    }
}
}

```

Output:



```

D:\White Field College\Exam...
Please Enter a number=
5
25125
Event received by User1 object
Event received by User2 object
|

```

Anonymous Method in C#

- ☞ An anonymous method is a method which doesn't contain any name which is introduced in C# 2.0.
- ☞ It is useful when the user wants to create an inline method and also wants to pass parameter in the anonymous method like other method.
- ☞ An anonymous method is defined using the **delegate** keyword and the user can assign this method to a variable of the delegate type.

Syntax:

```

delegate (parameter_list)
{
    //code here
}

```

```
};
```

Important Features of Anonymous Method

- ❖ Anonymous method can be defined using the **delegate** keyword
- ❖ Anonymous method must be assigned to a delegate.
- ❖ Anonymous method can access outer variables or functions.
- ❖ Anonymous method can be passed as a parameter.
- ❖ Anonymous method can be used as event handlers.

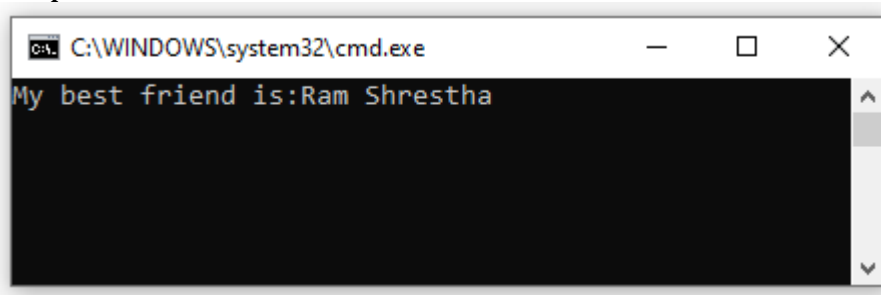
Example:

Example program to demonstrate anonymous method in C#.

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

namespace Anonymous
{
    internal class Program
    {
        public delegate void Besite(string name);
        static void Main(string[] args)
        {
            Besite best = delegate (string name)
            {
                Console.WriteLine("My best friend is:{0}",name);
            };
            best("Ram Shrestha");
            Console.ReadKey();
        }
    }
}
```

Output:



Example-2:

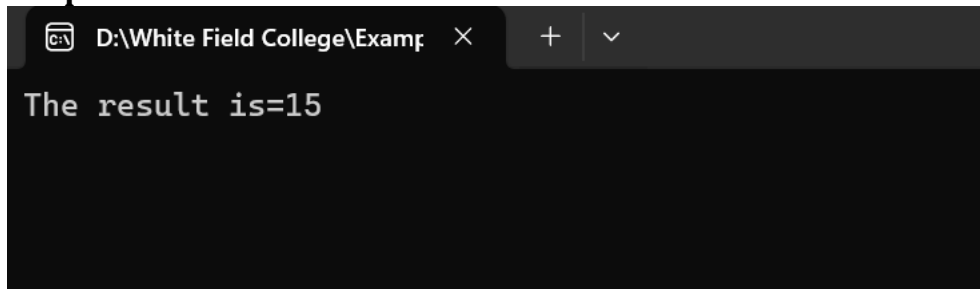
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Anonymous
{
    public delegate void MyDelegate(int num);
    internal class Program
    {
        //public static void MyMethod(int a)
        //{
        //    a += 10;
        //    Console.WriteLine("The result is="+a);
        //}
        static void Main(string[] args)
        {
            //MyDelegate obj = new MyDelegate(Program.MyMethod);
            MyDelegate obj= delegate (int a){
                a += 10;
                Console.WriteLine("The result is="+a);

            };

            //obj.Invoke(5);
            obj(5);
            Console.ReadKey();
        }
    }
}
```

Output:

A screenshot of a Windows console window. The title bar shows the file path 'D:\White Field College\Exam...' and standard window controls. The console output displays the text 'The result is=15' in white on a black background.

LAMBDA EXPRESSIONS

- ☞ Lambda expression in C# are used like anonymous functions, with the difference that in Lambda expressions you don't need to specify the type of the value that you input thus making it more flexible to use.
- ☞ **The '=>' is the lambda operator which is used in all lambda expressions.** The Lambda expression is divided into two parts, the left side is the input and the right is the expression.
- ☞ The Lambda Expressions can be of two types:
 - ❖ **Expression Lambda:** Consists of the input and the expression.

Syntax:

Input=>expression;

- ❖ **Statement Lambda:** Consists of the input and a set of statements to be executed. It can be used along the delegates.

Syntax:

Input =>{statements};

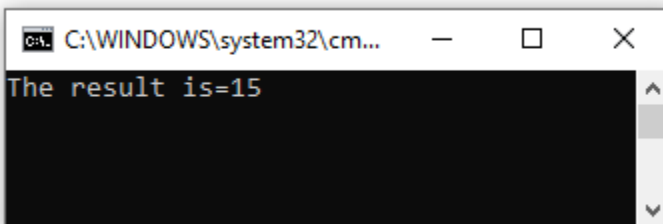
Example:

Example program to demonstrate lambda expression.

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

namespace Lambda
{
    internal class LambdaTest
    {
        static int test1() => 5;
        static int test2(int x) => x + 10;
        static void Main(string[] args)
        {
            int x = test1();
            int res = test2(x);
            Console.WriteLine("The result is=" + res);
            Console.ReadKey();
        }
    }
}
```

Output:



EXCEPTION HANDLING

- ☞ An exception is a problem that arises during the execution of a program.

- ☞ A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- ☞ Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: **try**, **catch**, **finally** and **throw**.
 - ❖ **try**: A try block identifies a block of code for which particular exception is activated. It is followed by one or more catch blocks.
 - ❖ **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
 - ❖ **finally**: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
 - ❖ **throw**: A program throws an exception when a problem shows up. This is done using **throw** keyword.

Exception Class in C#

- ☞ The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the System.Exception class are the System.ApplicationException and System.SystemException classes.
- ☞ The System.ApplicationException class supports exception generated by application programs. Hence the exceptions defined by the programmers should derived from this class.
- ☞ The System.SystemException class is the base class for all predefined system exception.
- ☞ Some predefined exception classes derived from the System.SystemException class are:

SN	Exception Class	Description
1	System.IO.IOException	It is used to handle I/O error
2	System.IndexOutOfRangeException	It is used to handles error generated when method refers to an array index out of range.
3	System.ArrayTypeMismatchException	It is used to handle errors generated when type is mismatched with the array type.
4	System.NullReferenceException	It handles errors generated from referencing a null object.
5	System.DivideByZeroException	It is used to handles error generated from dividing a dividend with zero.
6	System.InvalidCastException	It is used to handle errors generated during typecasting.

7	System.OutOfMemoryException	It is used to handles errors generated from insufficient free memory.
8	System.StackOverflowException	It is used to handle errors generated from stack overflow.

Syntax for exception handling

```
try
{
    //exception may get thrown within execution of this block.
}
catch(ExceptionA ex)
{
    //handling exception of type ExceptionA
}
catch(ExceptionB ex)
{
    //handle exception of type ExceptionB
}
finally
{
    //clean up code
}
}
```

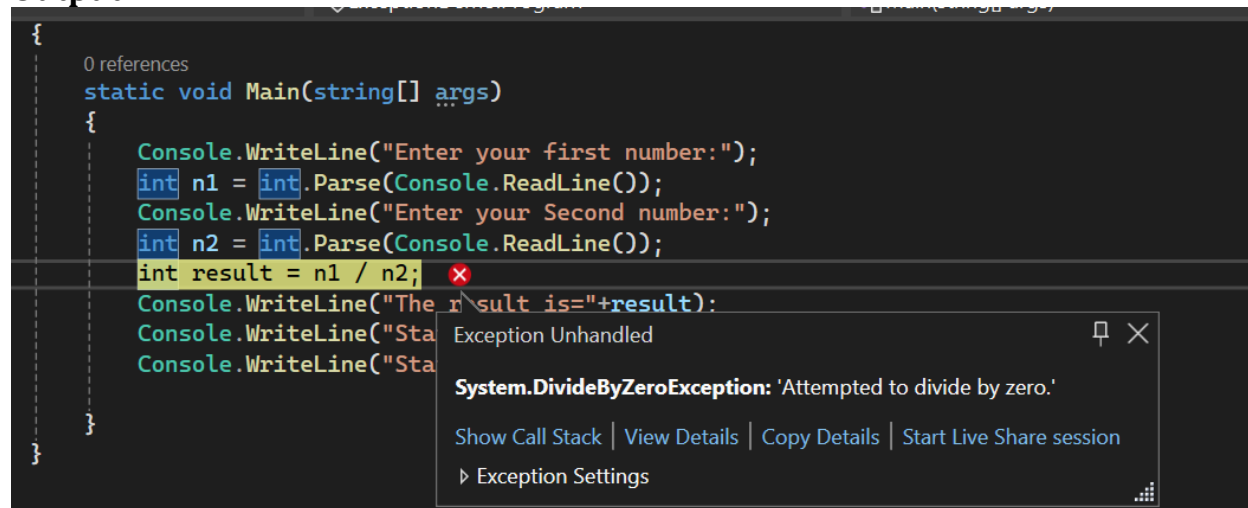
Example:

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

namespace ExceptionDemo
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your first number:");
            int n1 = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter your Second number:");
            int n2 = int.Parse(Console.ReadLine());
            int result = n1 / n2;
            Console.WriteLine("The result is="+result);
            Console.WriteLine("Statement first");
            Console.WriteLine("Statement second");
        }
    }
}
```

```
}  
}
```

Output:



The screenshot shows a code editor with a C# program. The code prompts the user to enter two numbers and then divides the first by the second. An exception dialog is displayed over the code, indicating a `System.DivideByZeroException` with the message "Attempted to divide by zero." The dialog also includes links for "Show Call Stack", "View Details", "Copy Details", and "Start Live Share session", as well as an "Exception Settings" link.

```
{  
    0 references  
    static void Main(string[] args)  
    {  
        Console.WriteLine("Enter your first number:");  
        int n1 = int.Parse(Console.ReadLine());  
        Console.WriteLine("Enter your Second number:");  
        int n2 = int.Parse(Console.ReadLine());  
        int result = n1 / n2;  
        Console.WriteLine("The result is="+result);  
        Console.WriteLine("Statement first");  
        Console.WriteLine("Statement second");  
    }  
}
```

Exception Unhandled
System.DivideByZeroException: 'Attempted to divide by zero.'
[Show Call Stack](#) | [View Details](#) | [Copy Details](#) | [Start Live Share session](#)
[Exception Settings](#)

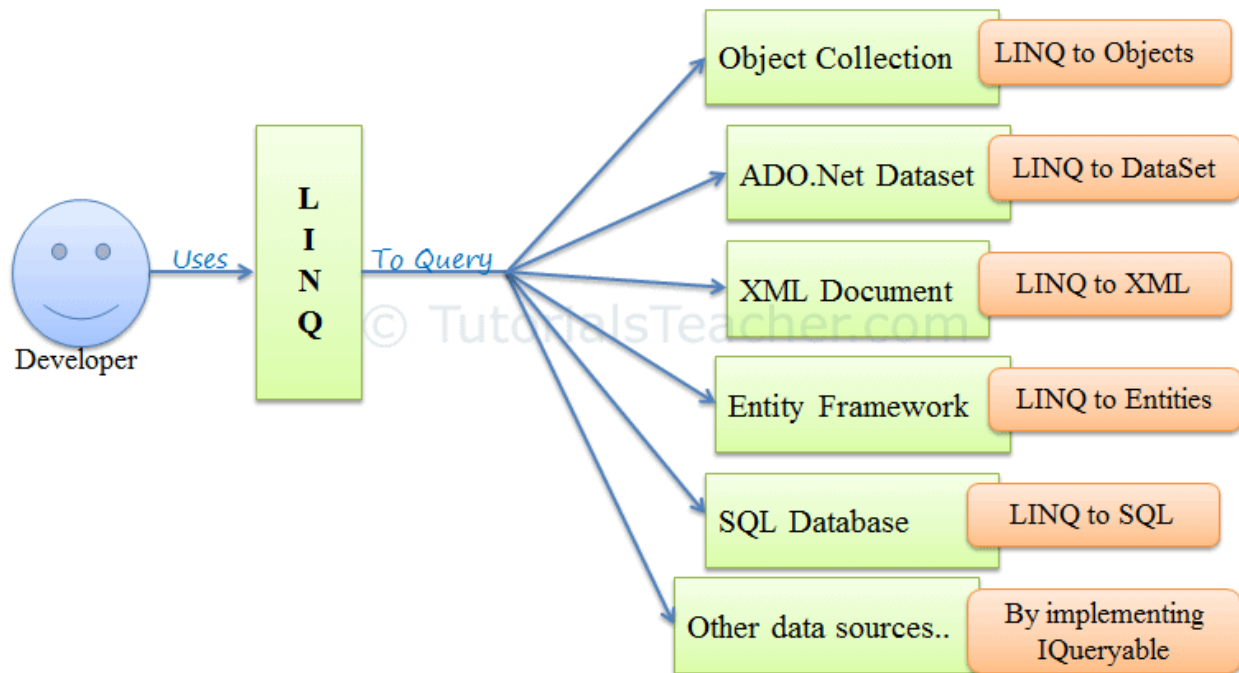
```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace ExceptionDemo  
{  
    internal class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Enter your first number:");  
            int n1 = int.Parse(Console.ReadLine());  
            Console.WriteLine("Enter your Second number:");  
            int n2 = int.Parse(Console.ReadLine());  
            try  
            {  
                int result = n1 / n2;  
                Console.WriteLine("The result is=" + result);  
            }  
            catch(DivideByZeroException ex)  
            {  
                Console.WriteLine("You cannot divide a number by zero!");  
                Console.WriteLine(ex.Message);  
            }  
  
            Console.WriteLine("Statement first");  
            Console.WriteLine("Statement second");  
            Console.ReadKey();  
        }  
    }  
}
```


Output:

```
D:\White Field College\Examf  X  +  v
Enter your first number:
20
Enter your Second number:
0
You cannot divide a number by zero!
Attempted to divide by zero.
Statement first
Statement second
```

INTRODUCTION TO LINQUE

- ☞ Language-Integrated Query (LINQ) is a powerful set of technologies based on the integration of query capabilities directly into the C# language. LINQ Queries are the first-class language construct in C# .NET, just like classes, methods, events. The LINQ provides a consistent query experience to query objects (LINQ to Objects), relational databases (LINQ to SQL), and XML (LINQ to XML).
- ☞ LINQ (Language Integrated Query) is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats. It is integrated in C# or VB, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.
- ☞ For example, SQL is a Structured Query Language used to save and retrieve data from a database. In the same way, LINQ is a structured query syntax built in C# and VB.NET to retrieve data from different types of data sources such as collections, ADO.Net DataSet, XML Docs, web service and MS SQL Server and other databases.



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



- ☞ The following example demonstrates a simple LINQ query that gets all strings from an array which contains 'a'.

Example: LINQ Query to Array

```

// Data source
string[] names = {"Bill", "Steve", "James", "Mohan"};

// LINQ Query
var myLinqQuery = from name in names
    where name.Contains('a')
    select name;
  
```

```
// Query execution
foreach(var name in myLinqQuery)
    Console.Write(name + " ");
```

LINQ Method

- ☞ The following is a sample LINQ method syntax query that returns a collection of strings which contains a word "Tutorials". We use lambda expression for this purpose.

Example: LINQ Method Syntax in C#

```
// string collection
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials",
    "Java"
};
```

```
// LINQ Method Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));
```

The following figure illustrates the structure of LINQ method syntax.

```
var result = strList.Where(s => s.Contains("Tutorials"));
```

© TutorialsTeacher.com

Extension method

Lambda expression

FORM HANDLING

- ☞ A form is a section of an HTML document where you put user-input controls, like text boxes, check boxes, radio buttons, and pull-down lists. You use forms when you want to collect and process user input.
- ☞ ASP.NET provides important feature event handling to Web Forms. It let us to implement event-based model for our application. As a simple example, we can add a button to an ASP.NET Web Forms page and then write an event handler for the button's click event. ASP.NET Web Forms allows events on both client and server sides.
- ☞ In ASP.NET Web Forms pages, however, events associated with server controls originate on the client but are handled on the Web server by the ASP.NET.
- ☞ ASP.NET Web Forms follow a standard .NET Framework pattern for event-handler methods. All events pass two arguments: an object representing the

object that raised the event, and an event object containing any event-specific information.

ASP.Net Web Form Features

☞ ASP.NET has multiple features and provides many tools to create and develop web applications. Here are some of the features of web forms:

- i. Server Controls
- ii. Master Pages
- iii. Working with data
- iv. Membership
- v. Client Script and Client Frameworks
- vi. Routing
- vii. State Management
- viii. Security
- ix. Performance
- x. Error Handling

i. Server Control

☞ It provides a vast set of server controls. These controls are like objects, and they run when they are requested and rendered to the browser. Some web pages are similar to HTML elements like text-box, button, checkbox, and hyperlink.

ii. Master Pages

☞ Master Pages is responsible for the consistent layout of our web applications. It gives a proper appearance and standard to different pages.

iii. Routing

☞ URL routing can be configured to a web application. A request URL is a URL that a user enters in a browser to browse in a specific place.

iv. Security

☞ Security always plays a crucial role in software development. ASP.NET provides different configuration options and extensibility points to make our systems more secure.

ASP .NET FORM LABEL

☞ This control is used to display textual information on the web forms.

☞ It is mainly used to create caption for the other controls like: textbox.

☞ This is server side control, asp provides own tag to create label.

Example:

```
<asp: labelID = "label1" runat= "server" Text = "label1"></asp: label>
```

Property	Description
Access Key	It is used to set a keyboard shortcut for the label.
Tab Index	The tab order of the control.
Back Color	It is used to set the background color of the label.
Border Color	It is used to set border color of the label.
Border width	It is used to set the width of the border of the label.
Font	It is used to set the font for the label text.
Fore Color	It is used to set the color of the label text.
Text	It is used to set text to be shown for the label.
ToolTip	It displays the text when the mouse is over the label.
Visible	To set the visibility of control on the form
Height	It is used to set the height of the control.
Width	It is used to set the width of the control.

ASP .Net WEB FORM TEXTBOX

- ☞ This is an input control which is used to take user input.
- ☞ To create a textbox either we can write a code or use the drag and drop facility of visual studio IDE.
- ☞ This is server side control, run asp provides own tag to create it.

<asp:TextBoxID= "TextBox1" runat= "server"></asp:TextBox>

Property	Description
Access Key	It is used to set keyboard shortcut for the control.
Tab Index	The tab order of the control.
Back Color	It is used to set the background color of the control.
Border Color	It is used to set the border color of the control.
Border Width	It is used to set width of border of the control.
Font	It is used to set font for the control text.
Fore Color	It is used to set color of the control.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when mouse is over the control.
Visible	To set the visibility of control on the form.
Height	It is used to set height of the control.
Width	It is used to set width of the control
Max length	It is used to set maximum number of characters that can be entered.
Read Only	It is used to make control read only.

ASP .NET DROP DOWN LIST

- ☞ The DropDownList is a web server control which is used to create an HTML Select component.
- ☞ It allows us to select an option from the dropdown list.
- ☞ It can contain any number of items ASP.Net provides a tag to create DropDownList for web application.

Example:

```
<asp:DropDownlist id= "DropDownlist1" runat = "server" DataSource = "<% databindingexpression %>"
```

```
DataSourceField = "DataSourceField"
```

```
DataSourceField = "DataSourceField"
```

```
AutoPostBack = "True | False"
```

```
OnSelectedIndexChanged = "OnSelectedIndexChangedMethod">
```

```
<asp: ListItem Value = "value" selected = "True | False">
```

```
Text
```

```
</asp: ListItem>
```

```
</asp: DropDownList>
```

CODE:

```
protected void Button1_click(object sender, EventArgs e)
{
    if(DropDownList1.SelectedValue == "")
    {
        Label1.Text = "Please Select a City";
    }
    else
    {
        Label1.Text = " Your Choice is:" + DropDownList1.SelectedValue;
    }
}
```

}

ASP .NET WEB FORMS RADIO BUTTON

- ☞ It is an input control which is used to take input from the user. It allows user to select a choice from the group of choices.

```
<asp: RadioButtonID = "RadioButton1" runat = "server" Text = "Male"  
GroupName = "Gender"/>
```

Property	Description
Access key	It is used to set a keyboard shortcut for the control.
TabIndex	It is used to set Tab order of the control.
BackColor	It is used to set the background color of the control.
BorderColor	It is used to set the border color of the control.
BorderWidth	It is used to set the width of the border of the control.
Font	It is used to set the font for the control text.
ForeColor	It is used to set the color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It display the text when the mouse is over the control.
Visible	To set visibility of control on the form.
Height	It is used to set the height of the control.
Width	It is used to set the width of the control.
GroupName	It is used to set the name of the radio button group.

FORM CHECKBOX

- ☞ Form CheckBox is used to get multiple inputs from the user.
- ☞ It allows user to select choices from the set of choices.
- ☞ It takes user input in yes or no format.
- ☞ It is useful when we want multiple choices from the user.

Property	Description
Access Key	It is used to set a keyboard shortcut for the control.
Tab Index	The tab order of the control
Back Color	It is used to set the background color of the control.
Border Color	It is used to set the border color of the control.
Border Width	It is used to set the border width of the control
Font	It is used to set the font for the control text.

Fore Color	It is used to set the color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when the mouse is over the control
Visible	To set visibility of control on the form
Height	It is used to set height of the control
Width	It is used to set width of the control.
Checked	It is used to set check state of the control either true or false.

Example:

```
<asp: checkbox ID = "CheckBox1" runat = "server" Text = "MVC" />
```

```
<asp: checkbox ID = "CheckBox2" runat = "server" Text = "CORE" />
```

```
<asp: checkbox ID = "CheckBox3" runat = "server" Text = "ASP.NET" />
```

ASP .NET FORM BOTTON

☞ This control is used to perform events. It is also used to submit client request to the server.

Property	Descriptions
Access Key	It is used to set a keyboard shortcut for the control.
Tab Index	It is used to set Tab order of the control.
Back Color	It is used to set the background color of the control.
Border Color	It is used to set the background color of the control.
Border Width	It is used to set the width of the border of the control.
Font	It is used to set the font for the control text.
Fore Color	It is used to set the color of the control text.
Text	It is used to set text to be shown for the control.
ToolTip	It displays the text when the mouse is over the control.
Visible	To set visibility of control on the form.
Height	It is used to set the height of the control.
Width	It is used to set the width of the control.

Example:

```
<asp: ButtonID = "Button1" runat= "server" Text = "Submit" BorderStyle =  
"Solid" ToolTip = "Submit"/>
```

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


VALIDATION CONTROL IN ASP.NET

- ☞ ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.
- ☞ ASP.NET provides a set of validation controls that provide an easy to use but powerful way to check for errors and if necessary, display messages to the user.
- ☞ There are six types of validation control in ASP.Net:
 - ❖ RequiredFieldValidator
 - ❖ RangeValidator
 - ❖ CompareValidator
 - ❖ RegularExpressionValidator
 - ❖ CustomValidator
 - ❖ ValidationSummary

Validation Control	Description
RequiredFieldValidator	It is used to make a control required.
CompareValidator	It is used to compare the value of an input control against a value of another input control.
RangeValidator	It is used to compare the value of an input control against a value of another input control.
RegularExpressionValidator	It evaluates the value of an input control to determine whether it matches a pattern defined by a regular expression.
CustomValidator	Allows you to write a method to handle the validation of the value entered.
ValidationSummary	It displays a list of all validation errors on the Web page.

Example:

Example program to demonstrate validator

Code:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="compare_validator_demo.aspx.cs"
Inherits="asp.netexample.compare_validator_demo" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<style type="text/css">
.auto-style1 {
width: 100%;
    }
.auto-style2 {
height: 26px;
    }
.auto-style3 {
height: 26px;
width: 93px;
    }
.auto-style4 {
width: 93px;
    }
</style>
</head>
<body>
<form id="form1" runat="server">
<table class="auto-style1">
<tr>
```

```

<td class="auto-style3">
    First value</td>
<td class="auto-style2">
<asp:TextBox ID="firstval" runat="server" required="true"></asp:TextBox>
</td>
</tr>
<tr>
<td class="auto-style4">
    Second value</td>
<td>
<asp:TextBox ID="secondval" runat="server"></asp:TextBox>
    It should be greater than first value</td>
</tr>
<tr>
<td class="auto-style4"></td>
<td>
<asp:Button ID="Button1" runat="server" Text="save"/>
</td>
</tr>
</table>
< asp:CompareValidator ID="CompareValidator1" runat="server"
ControlToCompare="secondval"
ControlToValidate="firstval" Display="Dynamic" ErrorMessage="Enter valid value"
ForeColor="Red"
Operator="LessThan" Type="Integer"></asp:CompareValidator>
</form>
</body>

```

</html>

Output:

Index - My ASP.NET Application

localhost:52174/Students/Index

Music Store Home About Contact Register Log in

Index

Student

Name

User name is required

Email

Email is required

Contact

Contact is required

Create

Index - My ASP.NET Application

localhost:52174/Students/Index

Music Store Home About Contact Register Log in

Index

Student

Name

Email

Invalid Email Address

Contact

Not a valid Phone number

Create

DATABASE CONNECTIVITY

- ☞ A database connection is a facility in computer science that allows client software to communicate with database server software, whether on the same

machine or not. A connection is required to send commands and receive answers.

ADO.NET

- ☞ ADO.NET is a module of .Net Framework which is used to establish a connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consist of classes that can be used to connect, retrieve, insert and delete data.

ADO.NET CLASSES

- ☞ Some important ADO.NET objects that are responsible for the CRUD operations is as follows:
 - ❖ **DataSet:**
Think about DataSet as a copy of a database stored in server's memory. It is used only for querying multiple SQL tables at once.
 - ❖ **SqlDataReader:**
It is used for querying data from a single SQL table.
 - ❖ **DataTable**
Data Table is a sub item of a DataSet and represents a database table stored in the memory.
 - ❖ **SqlConnection**
Object responsible with storing the data.
 - ❖ **SqlCommand**
Object responsible with sending the SQL query to the server and returning the results.
 - ❖ **SqlDataAdapter**
SqlDataAdapter is responsible with filling a DataSet with the data returned from the database.
 - ❖ **DataReader**
This retrieve data in forward only and read only form.
 - ❖ **DataAdapter**
This acts as a bridge between dataset and data source to load the dataset and reconcile changes made in dataset back to the source.

Web.config file with the new connection string

Syntax:

```
<connectionStrings>
```

```

    <add name = "your connectionStringName" connectionString = "Data Source =
DatabaseServerName; Integrated Security = true; Internal Catalog =
YouDatabaseName; uid= YourUserName; Password=yourpassword;" providerName=
"System.Data.SqlClient"/>
</connectionStrings>

```

Example:

```

<connectionStrings>

    <add name = "myconnection" connectionString = " Data Source = miniproject;
Integrated Security = true; Internal Catalog = MyWebDatabase" providerName =
"System.Data.SqlClient"/>

</connectionStrings>

```

Database Connection into .cs file:

```

SqlConnection con = new
SqlConnection(ConfigurationManager.ConnectionStrings["connectionstrings"].ToString());

```

Example:

```

SqlConnection con = new
SqlConnection(ConfigurationManager.ConnectionStrings["myconnection"].ToString())
;

```

Here **myconnection** is connection string that we create into Web.config file.

Creating SQL Statements

1. How to create database

```

CREATE DATABASE database_name;
USE Database_name;

```

```

Eg. CREATE DATABASE imsdB;
USE imsdB;

```

2. Dropping the Database (Deleting the database)

```

Syntax: DROP DATABASE database_name;

```

Eg. DROP DATABASE imsdbs;

3. How to create Table

```
CREATE TABLE Table_name  
(  
    ID INT NOT NULL IDENTITY(1,1),  
    FirstName VARCHAR(50),  
    SecondName VARCHAR(50)  
)
```

4. Adding Column on Table

Syntax: ALTER TABLE table_name
ADD column_name datatype;
Eg. ALTER TABLE tbl_user
ADD Email varchar(255);

5. Deleting Column from Table

Syntax: ALTER TABLE table_name
DROP COLUMN column_name;
Eg. ALTER TABLE tbl_user
DROP COLUMN Email;

6. Modifying Column (changing the data type of a column in a table)

Syntax: ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
Eg. ALTER TABLE tbl_user
MODIFY COLUMN class int;

7. Deleting Table

Syntax: DROP TABLE table_name;
Eg. DROP TABLE tbl_user;

SQL Statement to insert, select, update and delete data

8. Inserting data into table

Syntax: INSERT INTO table_name(column1, column2,
column3....)VALUES(value1, value2, value3,.....);
Eg. INSERT INTO tbl_user(uid, uname, upassword)VALUES(01,'Saroj', 'singh');

9. SELECT QUERY

SELECT * FROM Table_name;
Eg. SELECT * FROM tbl_user;

10. Selecting data using condition

Syntax: SELECT column1, column2,...

FROM table_name

WHERE condition;

Eg. SELECT uname, upassword FROM tbl_user WHERE uid=2;

11. Updating data (modifying the existing record in a table)

Syntax: UPDATE table_name

SET column1=value1 , column2=value2

WHERE condition;

[∴ WHERE condition is optional]

Eg. UPDATE tbl_user SET uname='Ram'

WHERE uid=01;

12. Deleting data

Syntax: DELETE FROM table_name WHERE condition;

Eg. DELETE FROM tbl_user WHERE uname='Ram';

Write a C# program to make simple calculator using asp.net with C#.

Calculator.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Calculator.aspx.cs"
Inherits="NabKshitijCollege.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table width="600">
                <tr><th width="300">Enter your first Number</th><td
width="300"><asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></td></tr>
                <tr><th width="300">Enter your Second Number</th><td width="300">
                    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox></td></tr>
                <tr><th width="300"></th><td width="300"></td></tr>
                <tr><th width="300"></th><td width="300"><asp:Label ID="result"
runat="server" Text="Result"></asp:Label>
```



```

        </td></tr><tr><th width="300"></th><td width="300">
        </td></tr>
        <tr><th width="300">
        </th><td width="300">
            <asp:Button ID="Button1" runat="server" Text="Sum"
OnClick="Button1_Click" />&nbsp;&nbsp;&nbsp;<asp:Button ID="Button2" runat="server"
Text="Sub" OnClick="Button2_Click" />&nbsp;&nbsp;&nbsp;<asp:Button ID="Button3"
runat="server" Text="Mul" OnClick="Button3_Click" />&nbsp;&nbsp;&nbsp;<asp:Button
ID="Button4" runat="server" Text="Div" OnClick="Button4_Click" />
        </td></tr>
    </table>
</div>
</form>
</body>
</html>

```

Calculator.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NabKshitijCollege
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            int res = int.Parse(TextBox1.Text) + int.Parse(TextBox2.Text);
            result.Text = res.ToString();
        }

        protected void Button2_Click(object sender, EventArgs e)
        {
            int res = int.Parse(TextBox1.Text) - int.Parse(TextBox2.Text);
            result.Text = res.ToString();
        }

        protected void Button3_Click(object sender, EventArgs e)
        {
            int res = int.Parse(TextBox1.Text) * int.Parse(TextBox2.Text);
            result.Text = res.ToString();
        }

        protected void Button4_Click(object sender, EventArgs e)
        {
            int res = int.Parse(TextBox1.Text) / int.Parse(TextBox2.Text);
            result.Text = res.ToString();
        }
    }
}

```

```
}  
}
```

Output:

Enter your first Number 5

Enter your Second Number 3

8

Sum Sub Mul Div

Example program to display form data in another page.

RegistrationForm.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="RegistrationForm.aspx.cs"  
Inherits="NabKshitijCollege.RegistrationForm" %>
```

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
```

```
<title></title>
```

```
</head>
```

```
<body>
```

```
<center><h1>Registration Form</h1></center>
```

```
<hr />
```

```
<center>
```

```
<table width="500">
```

```
<form id="form1" runat="server">
```

```
<div>
```

```
<tr><th></th><th></th></tr>
```

```
<tr><th></th><th></th></tr>
```

```
<tr><th></th><th></th></tr>
```

```
<tr><th></th><th></th></tr>
```

```
<tr>
```

```
<th width="200"><asp:Label runat="server" Text="First Name"
```

```
ID="Label1"></asp:Label></th> <td width="300"><asp:TextBox ID="fname"
```

```
runat="server"></asp:TextBox></td>
```

```
</tr>
```

```
<tr>
```

```

        <th width="200"><asp:Label ID="Label2" runat="server" Text="Last
Name"></asp:Label></th><td width="300"><asp:TextBox ID="lname"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
        <th width="200"><asp:Label ID="Label3" runat="server"
Text="Contact"></asp:Label></th><td width="300"><asp:TextBox ID="contact"
runat="server"></asp:TextBox></td>
    </tr>
    <tr><th></th><th></th></tr>
    <tr><th></th><th></th></tr>
    <tr>
        <th width="200"></th><td width="300"><asp:Button ID="Button1"
runat="server" Text="Save" OnClick="Button1_Click" /></td>
    </tr>
</div>
</form>
</table></center>
</body>
</html>

```

RegistrationForm.aspx.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NabKshitijCollege
{
    public partial class RegistrationForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Response.Redirect("Report.aspx?fname="+
                this.fname.Text + "lname"+
                this.lname.Text + "contact"+
                this.contact.Text

            );
        }
    }
}

```

Report.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Report.aspx.cs"
Inherits="NabKshitijCollege.Report" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

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

            <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>

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

Report.aspx.cs

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

namespace NabKshitijCollege
{
    public partial class Report : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = Request.QueryString["fname"];
            Label2.Text = Request.QueryString["lname"];
            Label3.Text = Request.QueryString["contact"];
        }
    }
}
```

Output:

Registration Form

First Name

Ram

Last Name

Chaudhary

Contact

9818084521

Save



RamlnameChaudharycontact9818084521