Madan Bhandari Memorial College

Binayak Nagar, New Baneshwor

NET CENTRIC COMPUTING (CSC 367)

LAB REPORT

Submitted By:

Aayush Basnet
Symbol: 29131
Date:

Submitted To:

Saroj Kumar Mahato
Dot Centric Computing Lecturer

Lab 1: **Date:**

Title: Installation of Visual Studio

Introduction:

1. Net Centric Computing:

    Net-centric computing is a computing architecture that uses networked computers and devices to provide services and applications to end-users. It involves the use of distributed computing technologies to provide scalable, reliable, and flexible computing solutions. Data and applications are stored and processed on distributed computers connected by a network, allowing users to access them from any location and device with an internet connection.

2. Visual Studio:

    Visual Studio is a powerful integrated development environment (IDE) created by Microsoft for building a wide range of software applications, including desktop applications, web applications, mobile applications, and games. It offers a comprehensive suite of tools and services for developers, including code editors, debuggers, compilers, and project management tools. Visual Studio also includes built-in support for many popular programming languages, such as C++, C#, JavaScript, and Python, as well as a variety of frameworks and libraries. With its rich set of features and powerful development tools, Visual Studio is widely regarded as one of the best IDEs for professional software development.

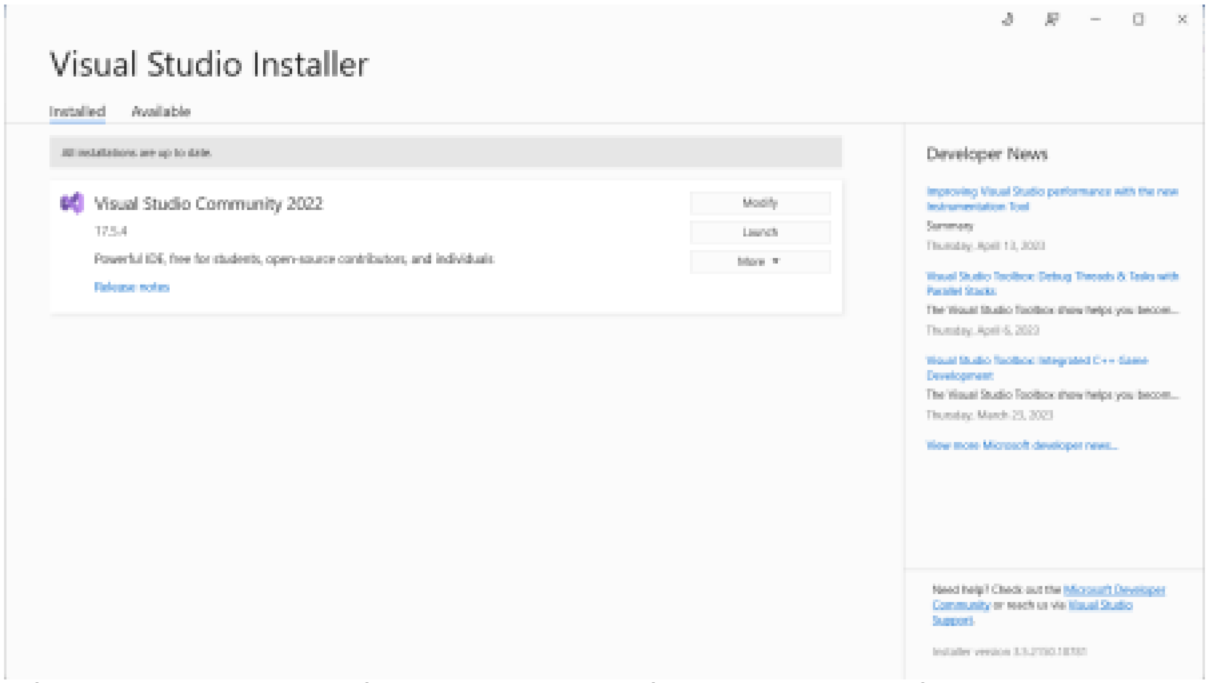
3. C #:

    C# is a modern, object-oriented programming language developed by Microsoft as part of its .NET framework. It was designed to be simple, efficient, and easy to learn, while still being powerful enough to build complex software applications. C# is similar in syntax to other popular programming languages like Java and C++, making it easy for developers to learn and transition between different languages. It is commonly used to

develop a wide
range of applications, including desktop applications, web applications, mobile applications, and games. C# supports a variety of programming paradigms, including procedural, functional, and object-oriented programming, and offers features such as automatic memory management and garbage collection. Overall, C# is a versatile and widely-used programming language with a strong emphasis on developer productivity and software quality.

Installation of Visual Studio:

To install Visual Studio, the first step is to download it from the official Microsoft website (https://visualstudio.microsoft.com/vs). There are different versions of Visual Studio available (Community, Enterprise, Professional), but the Community version is available for free and provides a comprehensive set of tools and features for building a wide range of applications. After selecting the Community version, you can download the installation file, which is usually in the form of an executable (.exe) file, and then run it to start the installation process. During the installation process, you may need to select the specific features and components that you want to install, depending on your specific requirements. The latest version of Visual Studio Community is 2022, and it offers a range of new features and improvements to help developers build high-quality software applications. Overall, the installation process for Visual Studio is straightforward and user-friendly, allowing developers to quickly get started with their projects.



All installations are up to date.

Visual Studio Community 2022

17.5.4

Powerful IDE, free for students, open-source contributors, and individuals.

Release notes

Improving Visual Studio performance with the new Instrumentation Tool

Summary

Thursday, April 13, 2023

Visual Studio Toolbox: Debug Threads & Tasks with Parallel Stacks

The Visual Studio Toolbox show helps you becom...

Thursday, April 6, 2023

Visual Studio Toolbox: Integrated C++ Game Development

The Visual Studio Toolbox show helps you becom...

Thursday, March 23, 2023

View more Microsoft developer news...

Need help? Check out the Microsoft Developer Community or reach us via Visual Studio Support.

Installer version [illegible]

fig: installation window of visual studio 2022 (after the installation of community version)

Opening C# in Visual Studio:

To open the C# in visual studio we follow the steps mention below:

i. Open Visual Studio 2022.

ii. Create a New Project.

iii. Continue without code.

iv. Go to menu file → New → Project → Create a new project → search for templates.

v. Console application for .net framework

Console.App (.net framework)

vi. Click Next.

vii. Configure your new project

Project Name → Lab 1

Framework → .net framework 6.0

viii. Click Next.

ix. Project will be displayed and now modify.

x. To run the program → click on start (press F5).

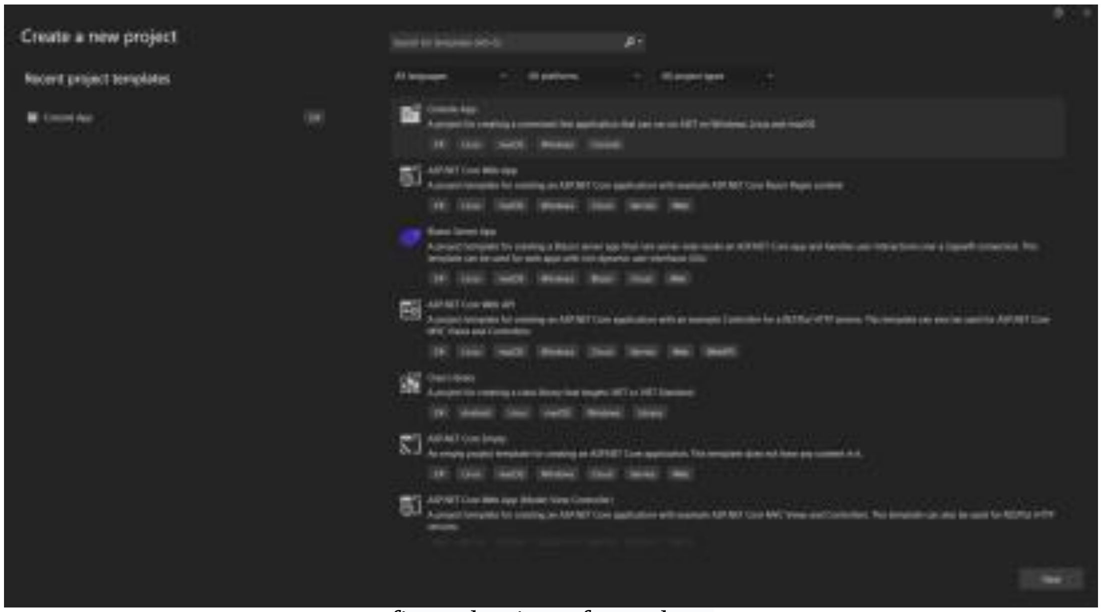
xi. To display the message

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

xii. Save.

xiii. Start → Result will be shown on console.

Some Snapshots of the above steps

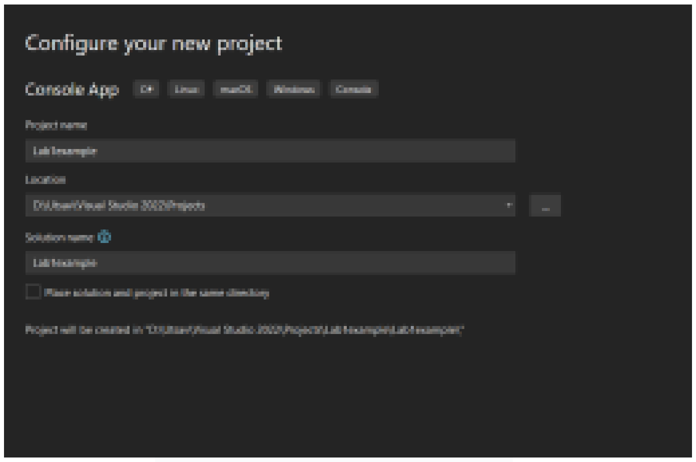

Create a new project

Recent project templates

Console App

[illegible]

fig: selection of template.


Configure your new project

Console App — C# · Linux · macOS · Windows · Console

Project name: Lab1example

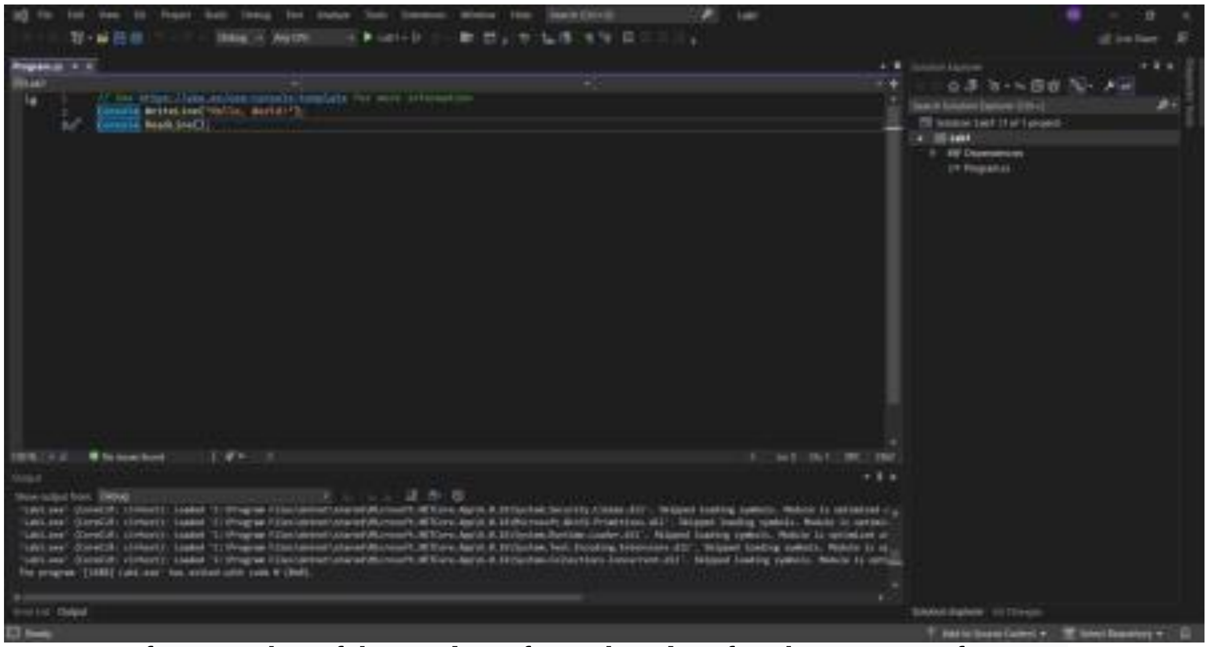
Location: D:\Utsav\Visual Studio 2022\Projects

Solution name: Lab1example

☐ Place solution and project in the same directory

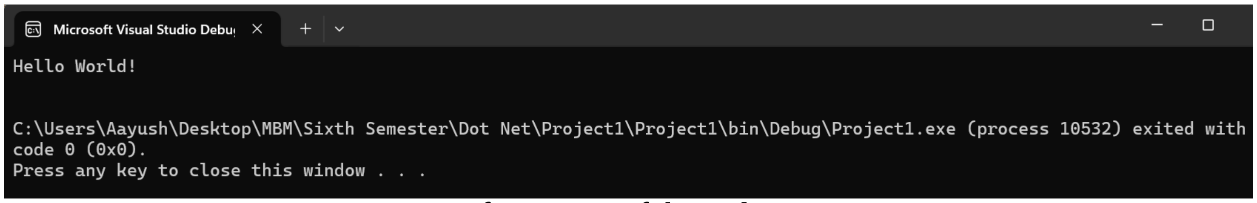
Project will be created in "D:\Utsav\Visual Studio 2022\Projects\Lab1example\Lab1example\"

fig: configuration of project.



```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
Console.ReadLine();
```

fig: snapshot of the window of visual studio after the creation of project.



```
Hello World!


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 10532) exited with
code 0 (0x0).
Press any key to close this window . . .
```

fig: output of the code.

Lab 2: Date:

Title: Basic C# programs

Introduction:

C# is a modern, object-oriented programming language developed by Microsoft as part of its .NET framework. It was designed to be simple, efficient, and easy to learn, while still being powerful enough to build complex software applications. C# is similar in syntax to other popular programming languages like Java and C++, making it easy for developers to learn and transition between different languages. It is commonly used to develop a wide range of applications, including desktop applications, web applications, mobile applications, and games. C# supports a variety of programming paradigms, including procedural, functional, and object-oriented programming, and offers features such as automatic memory management and garbage collection. Overall, C# is a versatile and widely-used programming language with a strong emphasis on developer productivity and software quality.

<u>Syntax Used</u>:

i. using System; → is a directive that indicates that this program will use the System namespace, which contains a variety of useful classes and methods for working with the .NET framework.

ii. using System.Collections.Generic; → is another directive that indicates that this program will use the System.Collections.Generic namespace, which contains a variety of collection classes that can be used to store and manipulate data.

iii. using System.Linq; → is a directive that indicates that this program will use the System.Linq namespace, which contains a set of extension methods that can be used for querying and manipulating data.

iv. using System.Text; → is a directive that indicates that this program will use the System.Text namespace, which contains classes for working with character encodings and string manipulation.

v. using System.Threading.Tasks; → is a directive that indicates that this program will use the System.Threading.Tasks namespace, which contains classes and methods for working with asynchronous programming and multithreading.

vi. namespace Lab2new_Introduction, → defines a namespace called Lab2new_Introduction that contains the program code.

vii. class Program, → defines a class called Program that contains the main method for the program.

viii. static void Main(string[] args), → is the main method of the program, which is the entry point for the program. It takes an array of string arguments as input and returns nothing. ix. Console.WriteLine("");, → uses the Console class to print the message to the console window.

x. Console.ReadLine();, → waits for the user to press the Enter key before closing the console window.

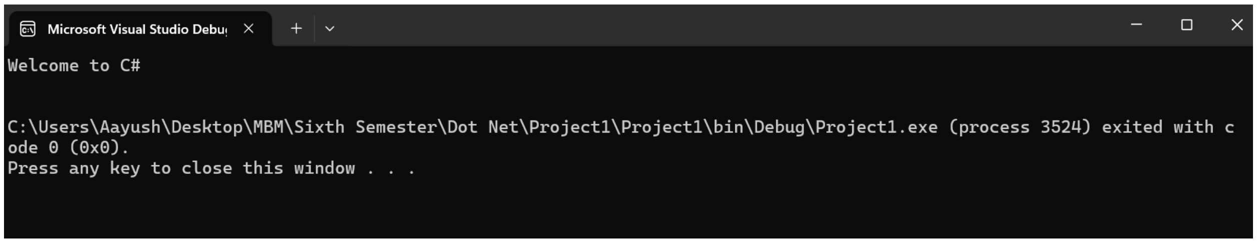
xi. Console.WriteLine("Hello,"+name); → This line of code uses string concatenation to print the message "Hello," followed by the value of the name variable. The + operator is used to concatenate the string literal "Hello," with the value of the name variable.

xii. Console.WriteLine("Welcome {0}",name); → This line of code uses string formatting to print the message "Welcome " followed by the value of the name variable. The {0} placeholder is used to indicate where the value of the name variable should be inserted in the string. The value of the name variable is then passed as an argument to the Console.WriteLine() method, which replaces the {0} placeholder with the value of the name variable. This approach is more flexible and allows you to format the output in a variety of ways, depending on your specific requirements.

Code:

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

namespace Lab2new_Introduction
{
 class Program
 {
 static void Main(string[] args)
 {
     Console.WriteLine("Welcome to C#");
    Console.ReadLine();
 }
 }
}
```

Output:

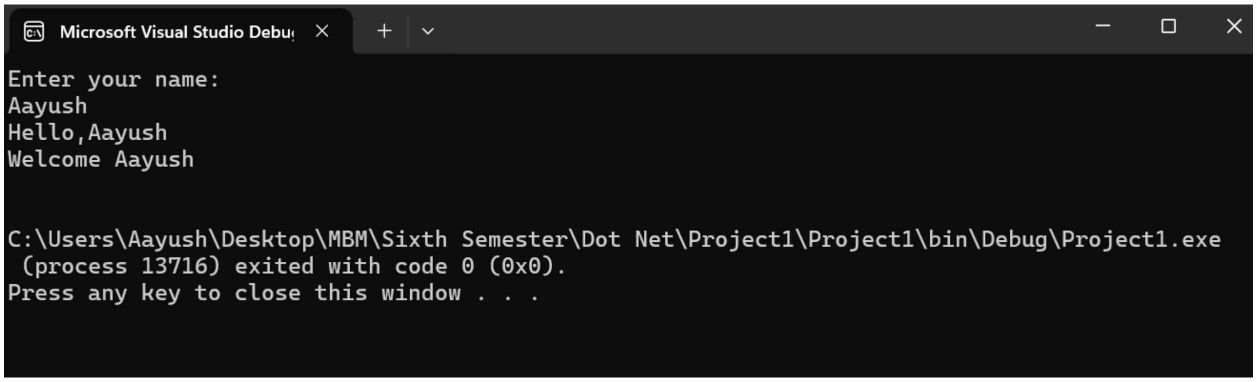


```
Welcome to C#


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 3524) exited with c
ode 0 (0x0).
Press any key to close this window . . .
```

Code:

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

namespace Lab2_Introduction
{
 class Program
 {
 static void Main(string[] args)
 {
             Console.WriteLine("Enter your name:");
            string name = Console.ReadLine();
             Console.WriteLine("Hello," + name);
         //Concatination Syntax
             Console.WriteLine("Welcome {0}", name);
 Console.ReadLine();
 }
 }
}
```

Output:



```
Enter your name:
Aayush
Hello,Aayush
Welcome Aayush


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe
 (process 13716) exited with code 0 (0x0).
Press any key to close this window . . .
```

Lab 3: **Date:**

Title: Entering Multiple String

Introduction:

In C#, a string is a sequence of characters that represents text. It is a reference type, which means that it is a class and is allocated on the heap. String literals are enclosed in double quotes, and can contain any combination of characters, including letters, numbers, and symbols. For example, "Hello, world!" is a string literal.

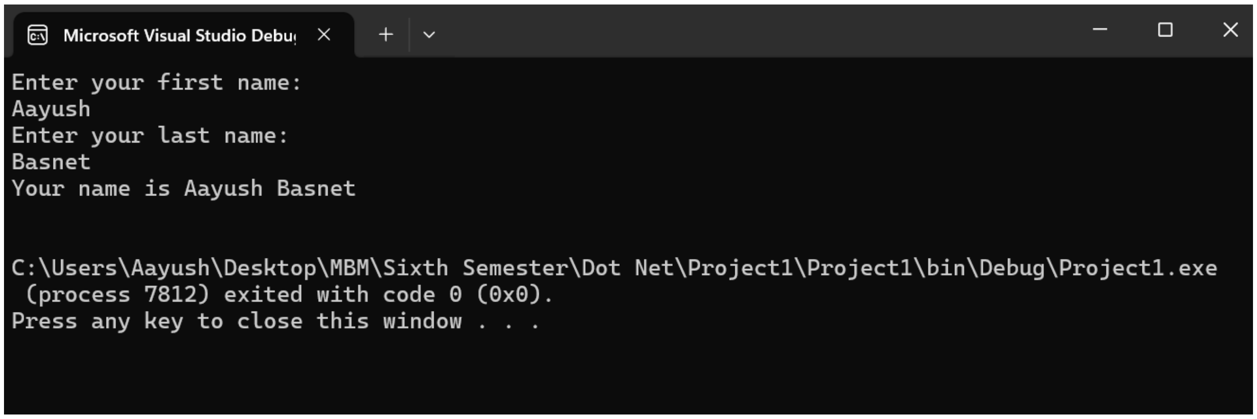
Syntax Used:

1. Console.WriteLine → to output a message to the console.
2. Console.ReadLine → reads a line of text from the standard input stream.

Code:

```
//Entering Multiple Strings
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab3IntroductionToCsharp
{
class Program
{
static void Main(string[] args)
{
        Console.WriteLine("Enter your first name:");
         String fname = Console.ReadLine();
         Console.WriteLine("Enter your last name:");
        String lname = Console.ReadLine();
    Console.WriteLine("Your name is {0} {1}",fname,lname);
Console.ReadLine();
}
}
}
```

Output:



```
Enter your first name:
Aayush
Enter your last name:
Basnet
Your name is Aayush Basnet


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe
 (process 7812) exited with code 0 (0x0).
Press any key to close this window . . .
```

Lab 4: Date:

Title: Passing Integer Data Type In C# Using Parse

Introduction:

In C#, the int.Parse() method is used to convert a string representation of an integer to an actual integer value. This is useful when you need to convert user input, file input, or other string data into an integer for use in calculations or other operations.

To use int.Parse(), you first read a string input using Console.ReadLine() or some other method, and then pass that string to int.Parse(). If the string can be converted to an integer, the method returns the integer value. If not, it throws an exception.

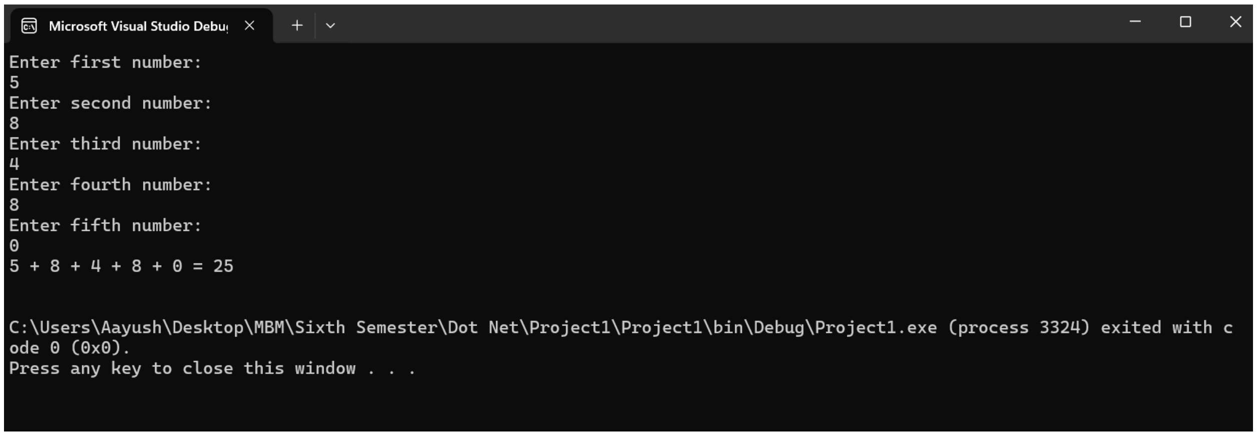
Syntax Used:

1. Console.WriteLine → to output a message to the console.
2. Console.ReadLine → reads a line of text from the standard input stream.
3. int.Parse() → to convert the string input to an integer value.

Code:

```
//Entering and passing integer data type in c# using parse.

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

namespace Lab4IntroductionToCsharp
{
class Program
{
static void Main(string[] args)
{
       Console.WriteLine("Enter first number:");
       int num1 = int.Parse(Console.ReadLine());
      Console.WriteLine("Enter second number:");
       int num2 = int.Parse(Console.ReadLine());
       Console.WriteLine("Enter third number:");
       int num3 = int.Parse(Console.ReadLine());
       Console.WriteLine("Enter fourth number:");
       int num4 = int.Parse(Console.ReadLine());
       Console.WriteLine("Enter fifth number:");
       int num5 = int.Parse(Console.ReadLine());
   int sum = num1 + num2 + num3 + num4 + num5;
 Console.WriteLine("{0} + {1} + {2} + {3} + {4} = {5}", num1, num2, num3, num4,
                            num5, sum);
Console.ReadLine();
}
}
}
```

Output:



```
Enter first number:
5
Enter second number:
8
Enter third number:
4
Enter fourth number:
8
Enter fifth number:
0
5 + 8 + 4 + 8 + 0 = 25


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 3324) exited with c
ode 0 (0x0).
Press any key to close this window . . .
```

Lab 5: **Date:**

Title: Built In Data Types In C#

Introduction:

In C#, a data type is a classification of data that specifies the type of value that a variable can hold. C# has several built-in data types, which are used to represent different types of values, such as integers, floating-point numbers, characters, and Boolean values. These data types are built into the C# language and are provided by the .NET Framework. They are used to declare variables, parameters, and return types of methods.

<u>Integral Type</u>:

- Signed Integer (which takes negative and positive values)
- Unsigned Integer (which only takes positive values) e.g. age

<u>Integer Types (Based on size)</u>:

- SBYTE
- BYTE
- SHORT
- USHORT
- INT
- UINT
- LONG
- ULONG

<u>Methods to see range of particular Datatype</u>:

- MINVALUE() METHOD
- MAXVALUE() METHOD

Memory Management Size:

Types	**Range**	**Size**
sbyte	-128 to 127	Signed 8-bit integer
byte	0 to 255	Unsigned 8-bit integer
char	U+0000 to U+ffff	Unicode 16-bit character
short	-32,768 to 32,767	Signed 16-bit integer
ushort	0 to 65,535	Unsigned 16-bit integer
int	-2,147,483,648 to 2,14,483,647	Signed 32-bit integer
uint	0 to 4,294,967,295	Unsigned 32-bit integer
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer

Syntax Used:

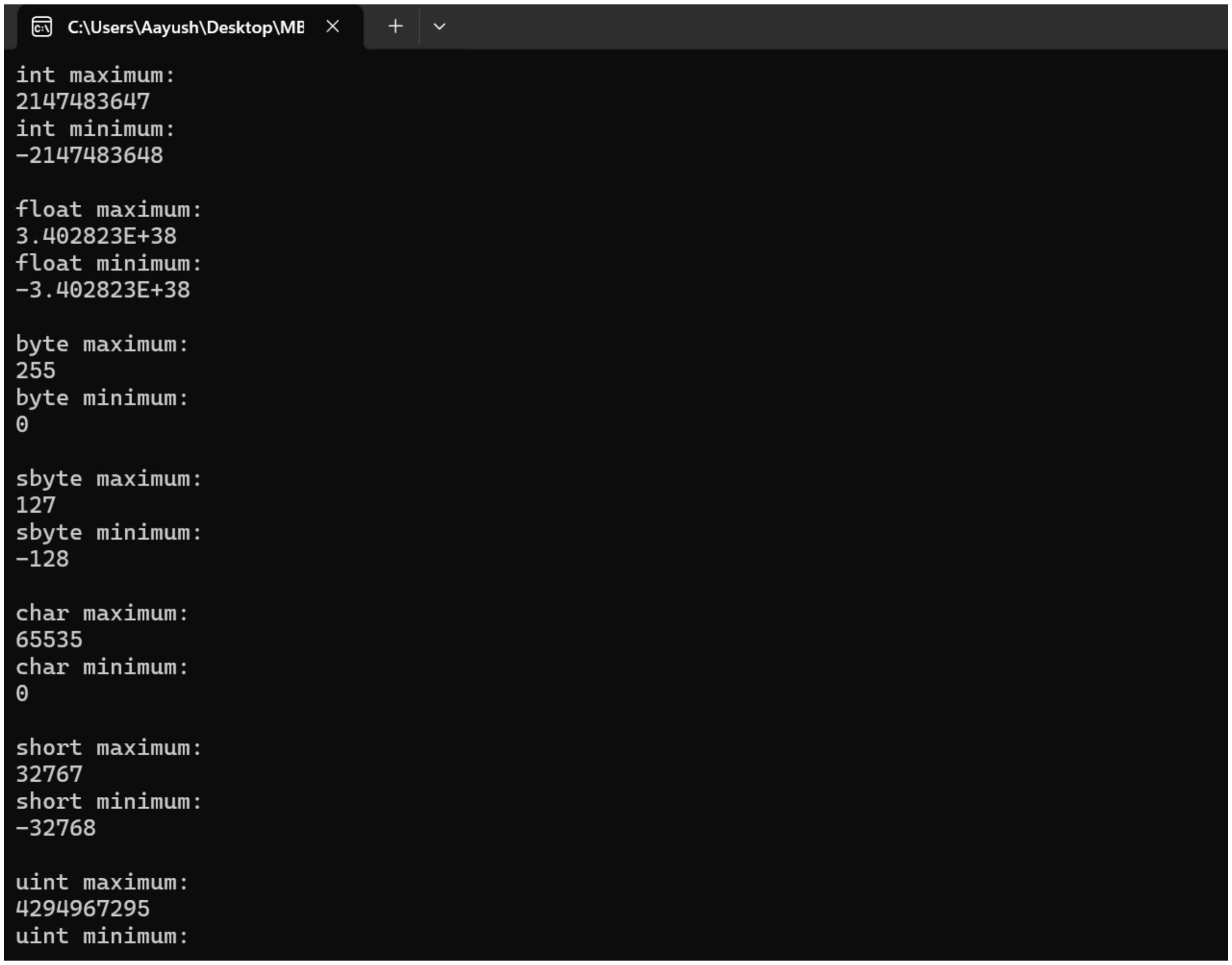
4. Console.WriteLine → to output a message to the console.
5. Console.ReadLine → reads a line of text from the standard input stream.
6. datatype.MaxValue → used to obtain maximum value of datatype.
7. datatype.MinValue → used to obtain minimum value of datatype.
8. “(int)char.MaxValue” and “(int)char.MinValue” → used to convert the char values into their corresponding integer values.

Code:

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

namespace DataTypeCSharP
{
 class Program
 {
 static void Main(string[] args)
 {

 //int
    Console.WriteLine("int maximum:");
    Console.WriteLine(int.MaxValue);
    Console.WriteLine("int minimum:");
    Console.WriteLine(int.MinValue);
 //float
     Console.WriteLine("\nfloat maximum:");
     Console.WriteLine(float.MaxValue);
     Console.WriteLine("float minimum:");
     Console.WriteLine(float.MinValue);
 //byte
    Console.WriteLine("\nbyte maximum:");
    Console.WriteLine(byte.MaxValue);
    Console.WriteLine("byte minimum:");
    Console.WriteLine(byte.MinValue);
 //sbyte
    Console.WriteLine("\nsbyte maximum:");
    Console.WriteLine(sbyte.MaxValue);
    Console.WriteLine("sbyte minimum:");
    Console.WriteLine(sbyte.MinValue);
 //char
    Console.WriteLine("\nchar maximum:");
     Console.WriteLine((int)char.MaxValue);
    Console.WriteLine("char minimum:");
     Console.WriteLine((int)char.MinValue);
 //short
    Console.WriteLine("\nshort maximum:");
    Console.WriteLine(short.MaxValue);
    Console.WriteLine("short minimum:");
    Console.WriteLine(short.MinValue);
 //uint
    Console.WriteLine("\nuint maximum:");
    Console.WriteLine(uint.MaxValue);
    Console.WriteLine("uint minimum:");
    Console.WriteLine(uint.MinValue);
 //long
    Console.WriteLine("\nlong maximum:");
    Console.WriteLine(long.MaxValue);
    Console.WriteLine("long minimum:");
    Console.WriteLine(long.MinValue);

//ulong
    Console.WriteLine("\nulong maximum:");
    Console.WriteLine(ulong.MaxValue);
    Console.WriteLine("ulong minimum:");
```

```
                    Console.WriteLine(ulong.MinValue);
                   Console.ReadLine();
                }
            }
        }
```

Output:


C:\Users\Aayush\Desktop\MB

```
int maximum:
2147483647
int minimum:
-2147483648

float maximum:
3.402823E+38
float minimum:
-3.402823E+38

byte maximum:
255
byte minimum:
0

sbyte maximum:
127
sbyte minimum:
-128

char maximum:
65535
char minimum:
0

short maximum:
32767
short minimum:
-32768

uint maximum:
4294967295
uint minimum:
```

Lab 6: **Date:**

Title: Boolean Data Type In C#

Introduction:

In C#, bool is a value type that represents a Boolean value, which can have only one of two possible values: true or false. The bool type is used to represent logical values and is frequently used in conditional expressions and logical operations.

A bool value can be assigned using the true or false keywords, or by the result of a comparison or logical operation.

In C#, bool values are typically used in logical operations, such as if statements, loops, and conditional expressions, to control the flow of the program based on the result of a comparison or a condition.

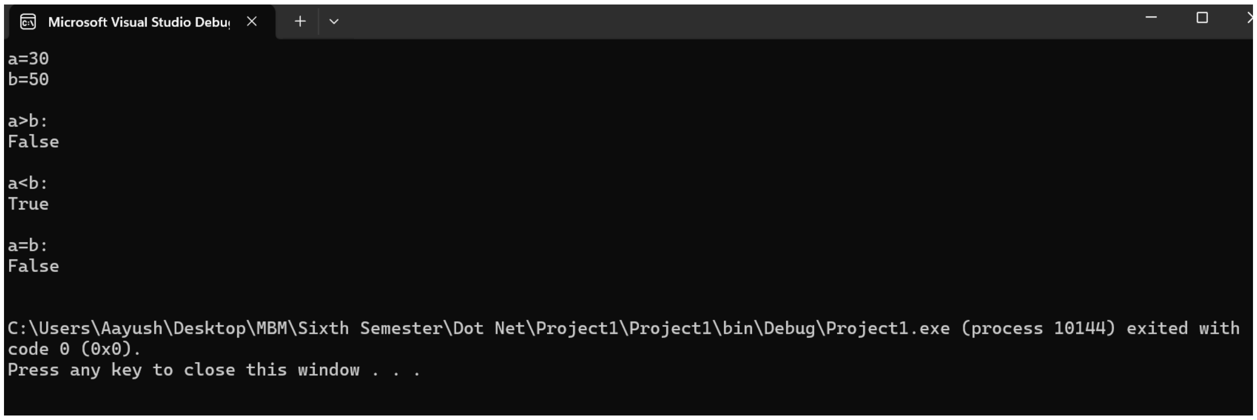
The bool type has a default value of false. When a bool variable is declared without initialization, it is automatically assigned the value false.

Syntax Used:

1. Console.WriteLine → to output a message to the console.
2. Console.ReadLine → reads a line of text from the standard input stream. 3. bool <variable> → stores true or false value on variable based on the condition.

Code:

```
/*Boolean DataType
    Bool Keyword is used for Boolean Data Type which only
                        stored True or False.
  */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataTypeBoolCSharP
{
 class Program
 {
 static void Main(string[] args)
 {
 int a = 30;
 int b = 50;
 //a greater than b
 bool c = a > b;
    Console.WriteLine("a=30\nb=50\n\na>b:");
    Console.WriteLine(c);
 //a smaller than b
 bool d = a < b;
       Console.WriteLine("\na<b:");
       Console.WriteLine(d);
 //a equals to b
 bool e = a == b;
       Console.WriteLine("\na=b:");
      Console.WriteLine(e);
      Console.ReadLine();
 }
      }
}
```



```
a=30
b=50

a>b:
False

a<b:
True

a=b:
False


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 10144) exited with
code 0 (0x0).
Press any key to close this window . . .
```

Lab 7: Date:

Title: Float, Double and Decimal Data Type In C#

Introduction:

1) float:

Float is a 32-bit floating-point type that is used to represent fractional numbers with up to seven significant digits. The float datatype is commonly used in scientific computations, graphics, and engineering applications where precision is important but storage space is limited. To declare a float variable, use the float keyword. f suffix is used to indicate that the literal value should be interpreted as a float.

2) double:

Double is a 64-bit floating-point type that is used to represent fractional numbers with up to 15 or 16 significant digits. The double datatype is commonly used in general-purpose applications where a higher degree of precision is required than what is provided by float. To declare a double variable, use the double keyword.

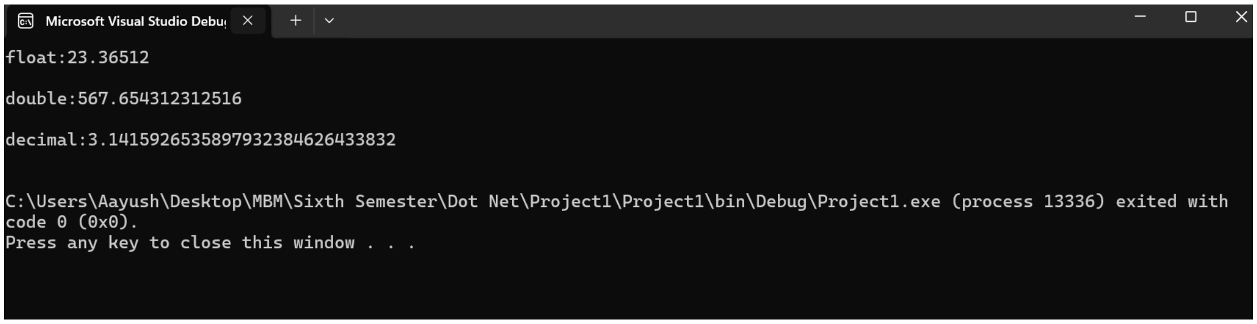
3) decimal:

Decimal is a 128-bit decimal type that is used to represent fractional numbers with up to 28 or 29 significant digits. The decimal datatype is commonly used in financial and monetary calculations where precision is critical. Unlike float and double, the decimal type provides exact decimal representation and does not suffer from rounding errors. To declare a decimal variable, use the decimal keyword. m suffix is used to indicate that the literal value should be interpreted as a decimal.

Code:

```
//Float Double and Decimal Data Type
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DataTypesFloat
{
 class Program
 {
 static void Main(string[] args)
 {
 //float
        float a = 23.36512345565769f;
       Console.WriteLine("float:{0}", a);
 //double
           double b = 567.65431231251589283471d;
        Console.WriteLine("\ndouble:{0}", b);
 //decimal
           decimal c = 3.1415926535897932384626433832m;
         Console.WriteLine("\ndecimal:{0}",c);
 Console.ReadLine();
 }
   }
}
```

Output:



```
float:23.36512

double:567.654312312516

decimal:3.1415926535897932384626433832


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 13336) exited with
code 0 (0x0).
Press any key to close this window . . .
```

Lab 8: **Date:**

Title: Synchronous And Asynchronous Programming

Introduction:

Synchronous Programming:

Synchronous programming is a programming model where operations take place sequentially.

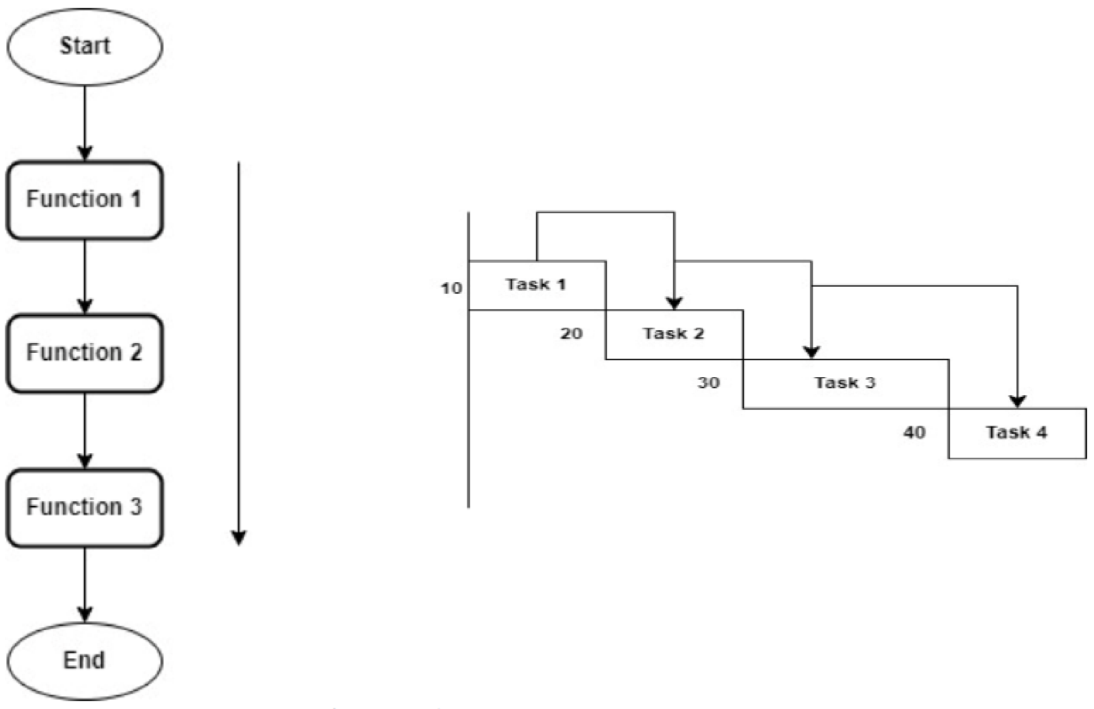
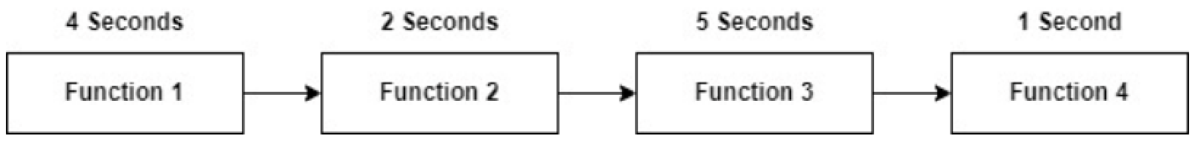


fig: Synchronous Programming

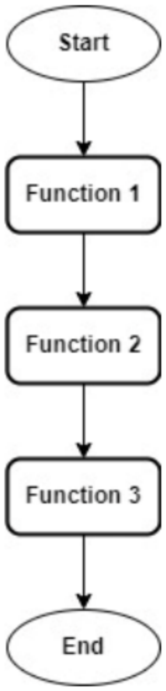


Executed in sequence.
Dependent on each other.

To overcome this delay, we use asynchronous programming model.

Asynchronous Programming:

Asynchronous programming is a programming model where operations does not depend on each other.



In this case function 1 will be executed and function 3 will be executed without waiting for function 2.

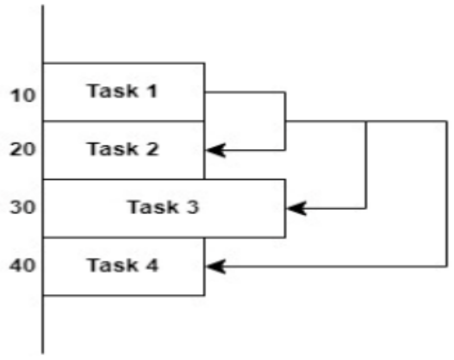
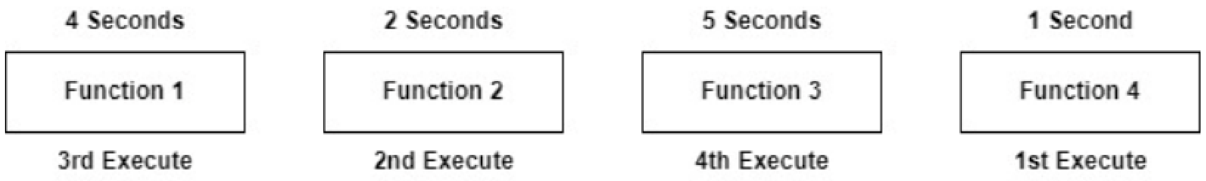


fig: Asynchronous Programming



Not executed in a sequence.
Not dependent on each other.

Syntax Used:

i. using System; → is a directive that indicates that this program will use the System namespace, which contains a variety of useful classes and methods for working with the .NET framework.

ii. using System.Collections.Generic; → is another directive that indicates that this program will use the System.Collections.Generic namespace, which contains a variety of collection classes that can be used to store and manipulate data.

iii. using System.Linq; → is a directive that indicates that this program will use the System.Linq namespace, which contains a set of extension methods that can be used for querying and manipulating data.

iv. using System.Text; → is a directive that indicates that this program will use the System.Text namespace, which contains classes for working with character encodings and string manipulation.

v. using System.Threading; → provides classes and interfaces to support multithreaded programming.

vi. using System.Threading.Tasks; → is a directive that indicates that this program will use the System.Threading.Tasks namespace, which contains classes and methods for working with asynchronous programming and multithreading.

vii. static void Main(string[] args), → is the main method of the program, which is the entry point for the program. It takes an array of string arguments as input and returns nothing. viii. public static void task_no() → is a method declaration.

ix. public static async void task_no() → is a method declaration but it marks the method as asynchronous method.

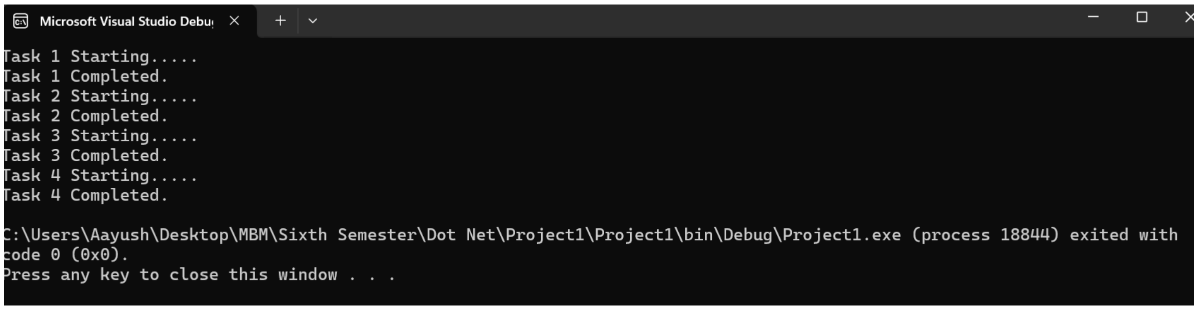
x. Task1(), Task2(), Task3(), and Task4() → these methods simulate the execution of some tasks.

xi. The await Task.Run(() => { ... }) → used to asynchronously execute a delegate (a lambda expression) on a background thread and wait for its completion, without blocking the calling thread.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
namespace AsyncAwaitDemo
{
internal class Program
{
static void Main(string[] args)
{
Task1();
Task2();
Task3();
Task4();
}
public static void Task1()
{
       Console.WriteLine("Task 1 Starting.....");
       Thread.Sleep(3000);
      Console.WriteLine("Task 1 Completed.");
}
public static void Task2()
{
       Console.WriteLine("Task 2 Starting.....");
       Thread.Sleep(1000);
     Console.WriteLine("Task 2 Completed.");
}
public static void Task3()
{
       Console.WriteLine("Task 3 Starting.....");
       Thread.Sleep(4000);
     Console.WriteLine("Task 3 Completed.");
}
public static void Task4()
{
       Console.WriteLine("Task 4 Starting.....");
       Thread.Sleep(2000);
     Console.WriteLine("Task 4 Completed.");
}
      }
    }
```

Output:

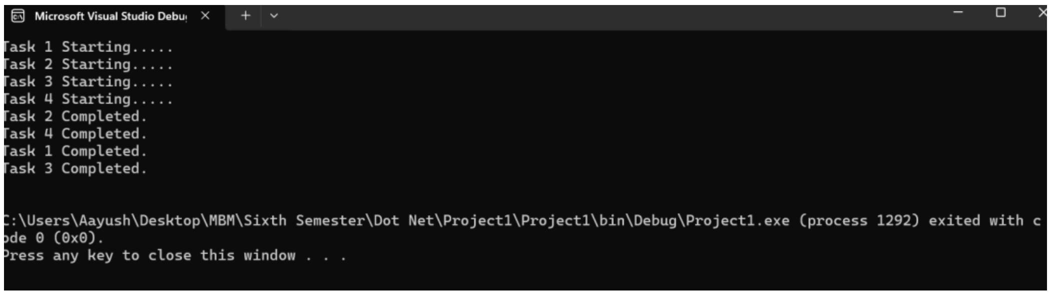


```
Task 1 Starting.....
Task 1 Completed.
Task 2 Starting.....
Task 2 Completed.
Task 3 Starting.....
Task 3 Completed.
Task 4 Starting.....
Task 4 Completed.

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 18844) exited with
code 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
namespace AsyncAwaitDemo {
 internal class Program {
 static void Main(string[] args) {
        Task1();
        Task2();
        Task3();
        Task4();
        Console.ReadLine();
 }
public static async void Task1() {
         await Task.Run(() =>
        {
        Console.WriteLine("Task 1 Starting.....");
        Thread.Sleep(3000);
        Console.WriteLine("Task 1 Completed.");
        });
 }
public static async void Task2() {
        await Task.Run(() =>
        {
                        Console.WriteLine("Task 2 Starting.....");
                        Thread.Sleep(1000);
                   Console.WriteLine("Task 2 Completed.");
        });
 }
 public static async void Task3() {
        await Task.Run(() =>
        {
                        Console.WriteLine("Task 3 Starting.....");
                        Thread.Sleep(4000);
                       Console.WriteLine("Task 3 Completed.");
        });
 }
 public static async void Task4() {
         await Task.Run(() =>
        {
                        Console.WriteLine("Task 4 Starting.....");
                        Thread.Sleep(2000);
                      Console.WriteLine("Task 4 Completed.");
        });
 } }}
```

Output:



```
Task 1 Starting.....
Task 2 Starting.....
Task 3 Starting.....
Task 4 Starting.....
Task 2 Completed.
Task 4 Completed.
Task 1 Completed.
Task 3 Completed.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Project1\Project1\bin\Debug\Project1.exe (process 1292) exited with c
ode 0 (0x0).
Press any key to close this window . . .
```

Lab 9: Date:

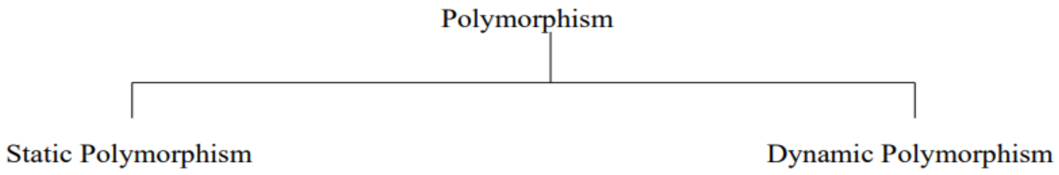
Title: Polymorphism

Introduction:

Polymorphism is a key concept in object-oriented programming (OOP) that allows developers to create code that is flexible and adaptable to a variety of scenarios. Essentially, polymorphism is the idea that a single name or method can be used in different ways depending on the context in which it is being used. This means that a single function or method can take on many different forms, depending on the input it receives or the objects it is working with.

Polymorphism is one of the four fundamental pillars of OOP, along with encapsulation, inheritance, and abstraction. These pillars are key principles that guide developers in creating effective and efficient code that is easy to maintain and update over time. Polymorphism is particularly important because it allows developers to write code that is more modular and reusable, reducing the amount of code they need to write from scratch and making their programs more efficient and effective.

There are two main types of polymorphism: static and dynamic. Static polymorphism, also known as compile-time polymorphism, occurs when the compiler determines which version of a function or method to use at compile time, based on the types of parameters passed in. Dynamic polymorphism, also known as runtime polymorphism, occurs when the correct version of a function or method is determined at runtime, based on the actual type of the object being worked with. Both types of polymorphism have their own advantages and uses, and understanding the differences between them is key to writing effective OOP code.



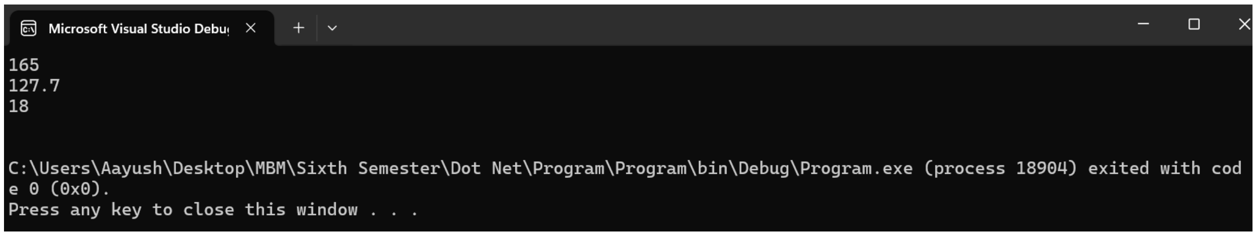
Static polymorphism, also called compile-time polymorphism, is a type of polymorphism where the type of an object is determined at compile time. In this type of polymorphism, the programmer defines multiple methods or operators with the same name but with different parameter types or number of parameters in a class. When a method or operator is called, the compiler determines which version to use based on the arguments passed in. Static polymorphism offers performance benefits over dynamic polymorphism since the method resolution is done at compile time rather than runtime. However, it is less flexible than dynamic polymorphism since the programmer must define all possible methods or operators in advance.

On the other hand, dynamic polymorphism, also known as runtime polymorphism, is a type of polymorphism where the type of an object is determined at runtime. It is achieved through inheritance and method overriding, where a subclass inherits a method from its parent class and provides its own implementation. When the method is called on an object of the subclass, the implementation of the subclass is executed instead of the parent class, allowing the subclass to customize the behavior of the inherited method. Dynamic polymorphism provides flexibility and extensibility to the code, as new subclasses can be created that inherit and override existing methods without modifying the original code. It also enables the use of abstract classes and interfaces, which define a set of methods that must be implemented by any concrete class that implements them. However, dynamic polymorphism incurs a performance overhead since the method resolution is done at runtime and involves an additional lookup step to determine the actual implementation to be called.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PolymorphismStatic
{
 class Program
 {
 public void Add() {
 int a = 120;
 int b = 45;
 int c = a + b;
 Console.WriteLine(c);

 }
 public void Add(int a, int b) {
 int c = a + b;
 Console.WriteLine(c);
}
 public void Add(string a, string b) {
 string c = a + " " + b;
 Console.WriteLine(c);
 }
 public void Add(float a, float b) {
 float c = a + b;
 Console.WriteLine(c);
 }
 static void Main(string[] args) {
 Program P = new Program();
 P.Add();
 P.Add(123f, 4.7f);
 P.Add(10, 8);
P.Add("Aayush"," ","Basnet");
 Console.ReadLine();
 }
}
}
```

Output:

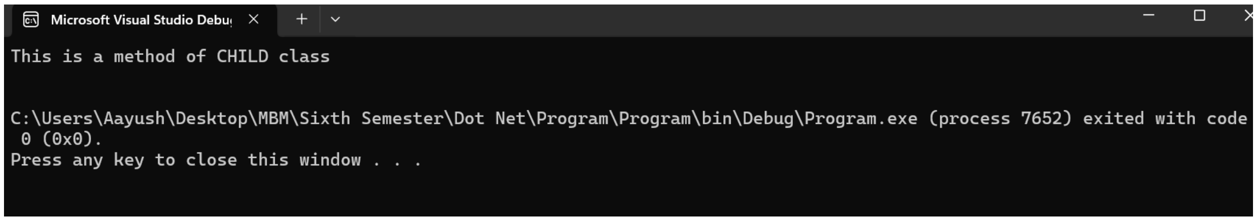


```
165
127.7
18


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 18904) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PlymorphismDynamicHiding
{
 class parent
 {
        public void print() {
Console.WriteLine("This is a method of PARENT class.");
        }
 }
 class child : parent {
        public void print() {
         Console.WriteLine("This is a method of CHILD class");
        }
 }
 class Program {
        static void Main(string[] args) {
                        child c = new child();
                       parent p = new child();
                       c.print();
                        Console.ReadLine();
        }
 }
}
```

Output:

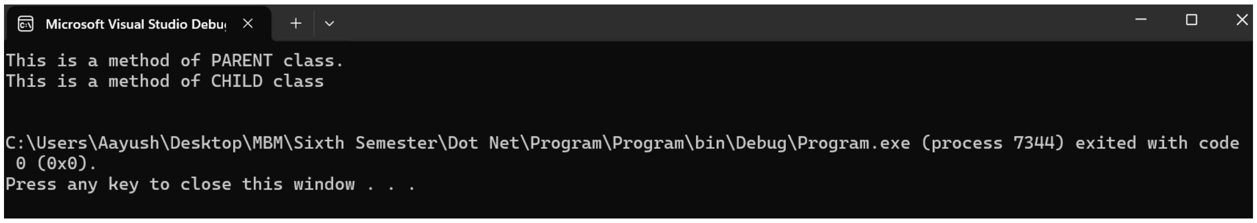


```
This is a method of CHILD class


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 7652) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PlymorphismDynamicOverloading {
 class parent {
       public virtual void print() {
      Console.WriteLine("This is a method of PARENT class.");
       }
 }
 class child : parent {
        public override void print(){
                    base.print();
              Console.WriteLine("This is a method of CHILD class");
       }
 }
 class Program {
       static void Main(string[] args) {
                     parent p = new child();
                    p.print();
                    Console.ReadLine();
       }
 }
}
```

Output:



```
This is a method of PARENT class.
This is a method of CHILD class


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 7344) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Lab 10: **Date:**

Title: Lambda Expression and Lambda Statement

Introduction:

Lambda Expression:

In C#, a lambda expression is a concise and powerful way to create anonymous functions. It allows you to write code that can be passed as an argument to a method or assigned to a variable, without having to define a separate named method.

A lambda expression consists of three parts: the input parameters, the lambda operator =>, and the function body. The input parameters specify the values that the lambda expression takes as input. The function body specifies the code that the lambda expression executes, and the lambda operator separates the input parameters from the function body.

Here's an example of a lambda expression that adds two numbers:

```
(int a, int b) => a + b
```

In this lambda expression, (int a, int b) specifies the input parameters, which are two integers named a and b. The lambda operator => separates the input parameters from the function body, which is a + b.

Lambda Statement:

A lambda statement is similar to a lambda expression in C#, but it allows you to write more complex functions with multiple statements. Like a lambda expression, a lambda statement is a concise and powerful way to create anonymous functions, which can be passed as arguments to a method or assigned to a variable.

A lambda statement consists of three parts: the input parameters, the lambda operator =>, and the function body. The input parameters specify the values that the lambda statement takes as input. The lambda operator => separates the input parameters from the function body, which is a block of code enclosed in curly braces.

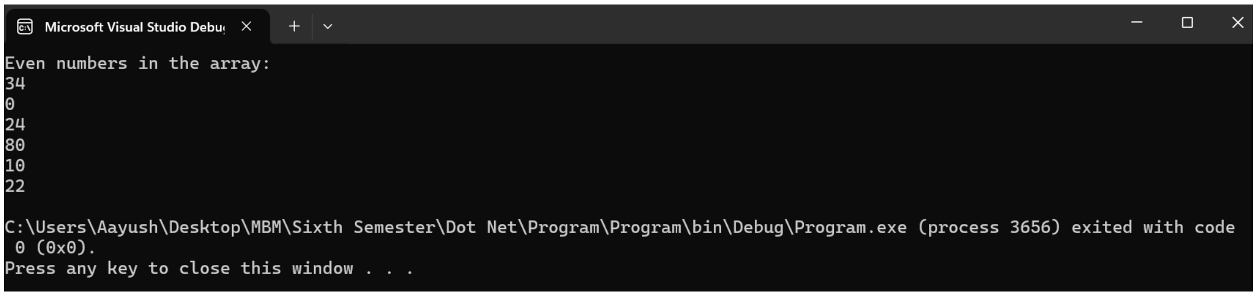
Syntax Used:

1. int[] numbers → declares an array of integers.
2. evenNumbers → a variable declared using the var keyword. The var keyword is used to declare a variable whose type is inferred by the compiler based on the value assigned to it.
3. var evenNumbers = Array.FindAll(numbers, n => n % 2 == 0) → uses a lambda expression to filter the array and create a new array that contains only the even numbers.
4. foreach (int num in evenNumbers) → loops through the evenNumbers array using a foreach loop and prints each even number to the console.
5. var evenNumbers = Array.FindAll(numbers, (int n) => { return n % 2 == 0; }); → uses a lambda statement to filter the array and create a new array that contains only the even numbers.
6. Array.FindAll → method that takes two arguments, the first argument is the array numbers that needs to be filtered and the second argument is a lambda expression/lambda statement.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace lambdaexpression
{
 class Program
 {
        static void Main(string[] args) {
                    // Create an array of numbers

         int[] numbers = { 34, 23, 43, 0, 24, 45, 87, 80, 3, 10, 22 };

                 // Use a lambda expression to filter the array
               var evenNumbers = Array.FindAll(numbers, n => n  % 2 == 0);

               // Print the even numbers to the console
                 Console.WriteLine("Even numbers in the array:");
                 foreach (int num in evenNumbers) {
                        Console.WriteLine(num);
                     }
                 }
         }
}
```

Output:

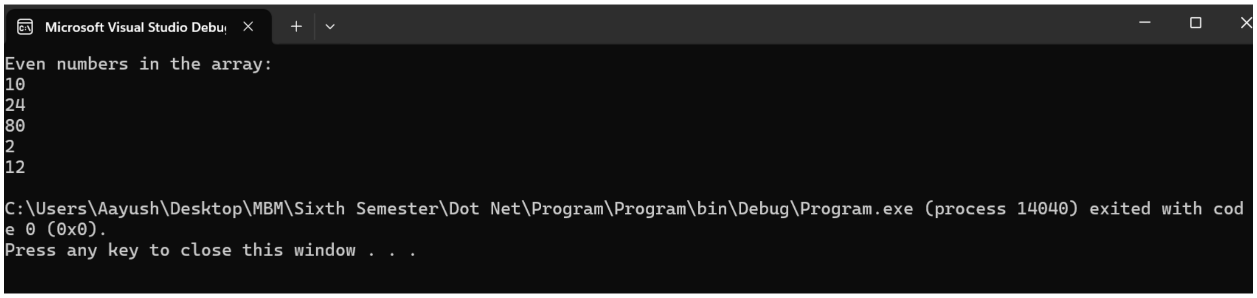


```
Even numbers in the array:
34
0
24
80
10
22

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 3656) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace lambdastatement
{
 class Program
 {
 static void Main(string[] args)
 {
             // Create an array of numbers
            int[] numbers = { 10, 23, 43, 24, 65, 77, 80, 3,2, 12};

           // Use a lambda statement to filter the array
          var evenNumbers = Array.FindAll(numbers, (int n)
                  =>
              {
              return n % 2 == 0;
              });

         // Print the even numbers to the console
          Console.WriteLine("Even numbers in the array:");
           foreach (int num in evenNumbers)
              {
                       Console.WriteLine(num);
               }
     }
 }
}
```

Output:



```
Even numbers in the array:
10
24
80
2
12

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 14040) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Lab 11: Date:

Title: Exception Handling

Introduction:

Exception Handling:

Exception handling in C# is a mechanism that allows programmers to handle and recover from runtime errors or exceptional situations that may occur while the program is running. These errors may be caused by a variety of reasons, such as unexpected user input, hardware or software malfunctions, or resource constraints.

In C#, exceptions are represented by objects of the Exception class or its derived classes. When an exception is thrown, the runtime searches for an appropriate exception handler that can catch and handle the exception. If no handler is found, the program terminates and the exception is logged.

There are two type of exception handling mechanism: Logical Implementation and Try Catch Implementation.

Try Catch Implementation:

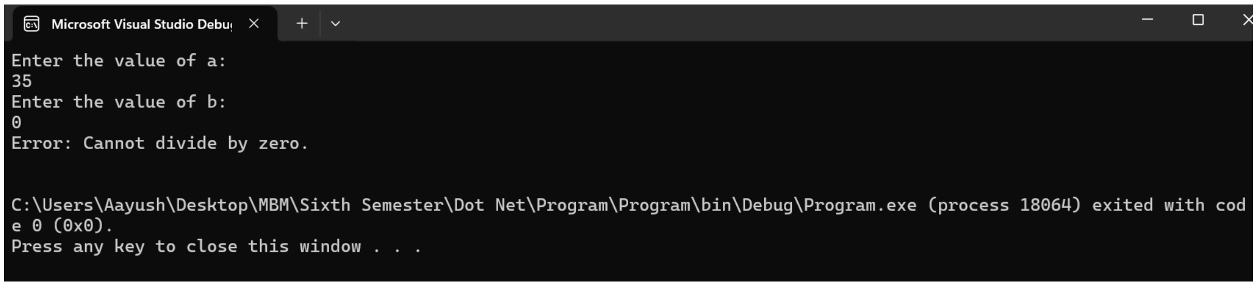
The try-catch implementation of exception handling in C# provides a mechanism for handling runtime errors that may occur during program execution. By enclosing the code that may throw an exception in a try block, you can catch any exception that is thrown and handle it in a catch block. This allows us to gracefully handle runtime errors by displaying appropriate error messages, logging the exception details, or taking other appropriate actions.

In addition, you can use a finally block to perform cleanup operations or release any resources that were allocated in the try block, regardless of whether an exception was thrown or not. The try-catch implementation of exception handling is an important aspect of writing robust and reliable C# code.

Code:

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

namespace ExceptionLogical
{
    class Program {
            static void Main(string[] args)  {
                    int a, b, c;

                     Console.WriteLine("Enter the value of a:");
                        a = int.Parse(Console.ReadLine());

                     Console.WriteLine("Enter the value of b:");
                        b = int.Parse(Console.ReadLine());

            if (b == 0) {
        Console.WriteLine("Error: Cannot divide by zero.");
}
else  {
c = a / b;
  Console.WriteLine("Division of a by b is = " + c);
}
Console.ReadLine();
}
}
}
```

Output:

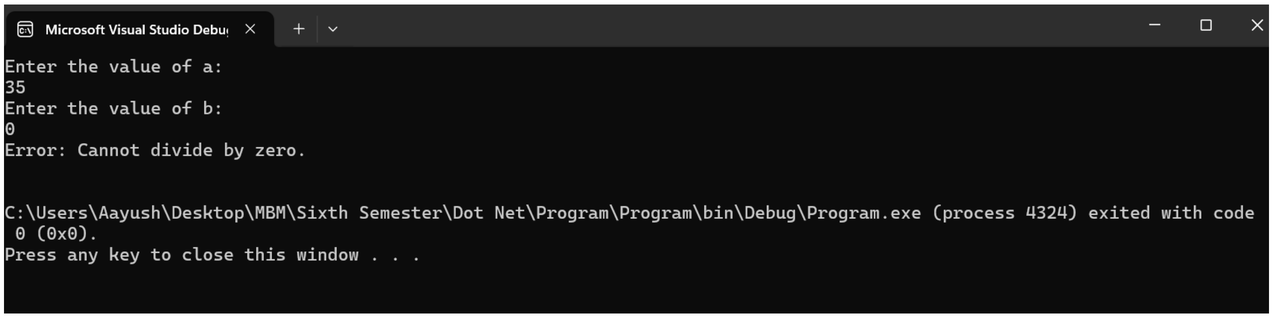


```
Enter the value of a:
35
Enter the value of b:
0
Error: Cannot divide by zero.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 18064) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Exceptiontrycatch
{
        class Program {
                static void Main(string[] args) {
                        int a, b, c;

                         Console.WriteLine("Enter the value of a:");
                            a = int.Parse(Console.ReadLine());

                         Console.WriteLine("Enter the value of b:");
                            b = int.Parse(Console.ReadLine());

 try
 {
 if (b == 0)
 {
         throw new DivideByZeroException("Error: Cannot divide by zero.");
 }

 c = a / b;
   Console.WriteLine("Division of a by b is = " + c);
 }
           catch (DivideByZeroException ex)
 {
               Console.WriteLine(ex.Message);
 }

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

Output:



```
Enter the value of a:
35
Enter the value of b:
0
Error: Cannot divide by zero.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 4324) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Lab 12: Date:

Title: Constructor

Introduction:

Constructor:

Constructor is a special method of a class which will invoke automatically whenever instance or object of class is created. Constructor name should match with class and constructor does not have any return type.

Constructor are responsible for object initialization and memory allocation of its class. If we create any class without constructor, compiler will automatically create default constructor for that class. There is always at least one constructor in every class.

Types of constructors:

i. Default Constructor: A constructor without having any parameter are called default constructor. In this constructor every instance of the class will be initialized without any parameter value.

ii. Parameterized Constructor: A constructor with at least one parameter is called a parameterized constructor. The advantage of a parameterized constructor is that we can initialize the value at the time of certain of object of class.

iii. Copy Constructor: A parameterized constructor that contains a parameter of same class type is called as copy constructor. The purpose of copy constructor is to initialize new instance to values of an existing instance.

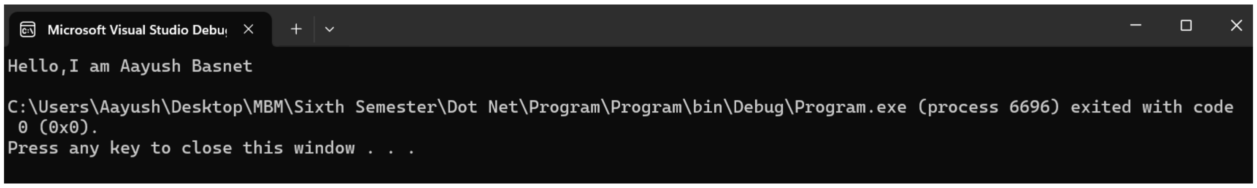
iv. Private Constructor: A constructor with private access modifier is known as private constructor. It is generally used in classes that contain static members only.

v. Static Constructor: A constructor with static keyword is known as static constructor. It is used to initialize static fields of the class and to write the code that needs to be executed only once.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
 public class MyClass
 {
        private string message;

       // Default constructor
       public MyClass()
       {
             message = "Hello,I am Aayush Basnet";
       }

        public void DisplayMessage()
       {
                    Console.WriteLine(message);
       }
 }

 public class Program
 {
    public static void Main(string[] args)
       {
                   MyClass obj = new MyClass();
                    obj.DisplayMessage();
 }
 }
}
```

Output:

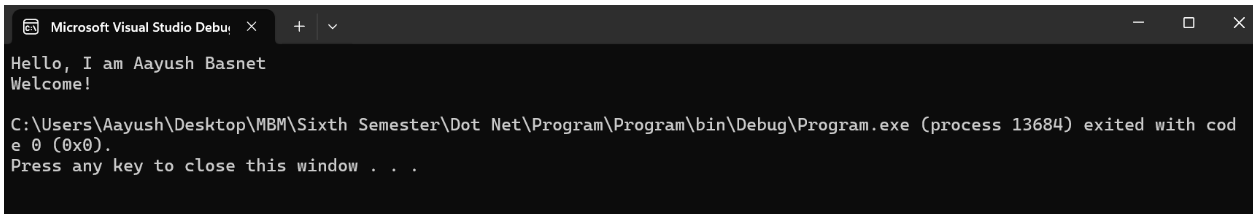


```
Hello,I am Aayush Basnet

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 6696) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
    public class MyClass
    {
            private string message;

            // Parameterized constructor
    public MyClass(string msg)
    {
            message = msg;
}

    public void DisplayMessage()
    {
    Console.WriteLine(message);
    }
}

public class Program
{
  public static void Main(string[] args)
    {
 MyClass obj1 = new MyClass("Hello, I am Aayush Basnet");
    obj1.DisplayMessage();

  MyClass obj2 = new MyClass("Welcome!");
     obj2.DisplayMessage();
}
}
  }
```

Output:

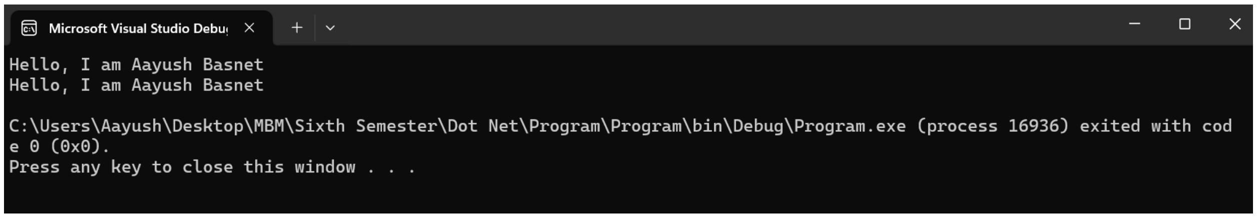


```
Hello, I am Aayush Basnet
Welcome!

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 13684) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
        public class MyClass
       {
                    Private string message;

                    public MyClass(string msg)
                    {
                                message = msg;
                    }

                     // Copy method
                    public MyClass Copy()
                    {
                         MyClass copy = new MyClass(this.message);
                       return copy;
                    }

                    public void DisplayMessage()
                    {
                                Console.WriteLine(message);
                    }
}

public class Program
{
    public static void Main(string[] args)
       {
         MyClass obj1 = new MyClass("Hello, I am Aayush Basnet");
                    obj1.DisplayMessage();

       MyClass obj2 = obj1.Copy();
       obj2.DisplayMessage();
}
}
   }
```

Output:



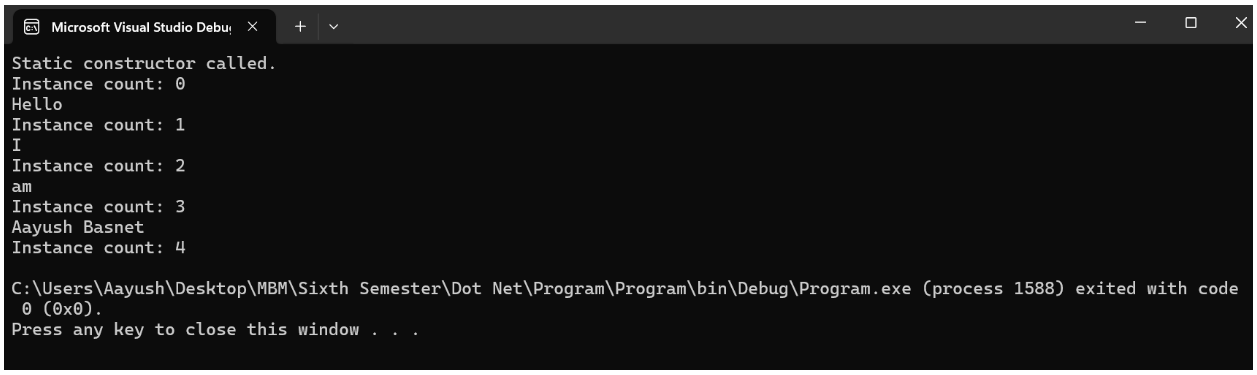
Microsoft Visual Studio Debu

```
Hello, I am Aayush Basnet
Hello, I am Aayush Basnet

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 16936) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
 public class MyClass
 {
                  private static int instanceCount;
 private string message;

 // Static constructor
 static MyClass()
 {
 instanceCount = 0;
 Console.WriteLine("Static constructor called.");  }

 public MyClass(string msg)
 {
 message = msg;
 instanceCount++;
 }

 public void DisplayMessage()
 {
       Console.WriteLine(message);
 }

                 public static void DisplayInstanceCount()
 {
        Console.WriteLine("Instance count: " +  instanceCount);
 }
 }

 public class Program
 {
    public static void Main(string[] args)
 {
          MyClass.DisplayInstanceCount();

            MyClass obj1 = new MyClass("Hello");
                     obj1.DisplayMessage();
          MyClass.DisplayInstanceCount();

           MyClass obj2 = new MyClass("I");
                     obj2.DisplayMessage();
          MyClass.DisplayInstanceCount();

          MyClass obj3 = new MyClass("am");
                     obj3.DisplayMessage();
          MyClass.DisplayInstanceCount();

    MyClass obj4 = new MyClass("Aayush Basnet");
                     obj4.DisplayMessage();
          MyClass.DisplayInstanceCount();
 }
 }
    }
```

Output:

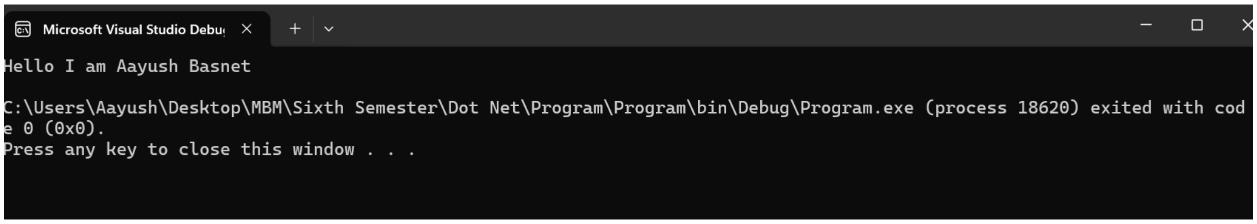


```
Static constructor called.
Instance count: 0
Hello
Instance count: 1
I
Instance count: 2
am
Instance count: 3
Aayush Basnet
Instance count: 4

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 1588) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace constructors
{
public class MyClass
{
private string message;

// Private constructor
private MyClass(string msg)
{
message = msg;
}

public void DisplayMessage()
{
Console.WriteLine(message);
}

            public static MyClass CreateInstance(string msg)
                  {
      return new MyClass(msg);
}
}

public class Program
{
           public static void Main(string[] args)
                  {

                      /* Trying to create an instance using the
            private constructor will result in an error MyClass
               obj = new MyClass("Hello"); Error: The constructor
                 MyClass.MyClass(string) is not accessibleInstead, we
                  use the public static method CreateInstance to create
                  an instance*/

 MyClass obj = MyClass.CreateInstance("Hello I am Aayush Basnet");
obj.DisplayMessage();
}
}
}
```

Output:



```
Hello I am Aayush Basnet

C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 18620) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Lab 13: Date:

Title: Inheritance

Introduction:

Inheritance:

In C#, inheritance is a fundamental concept in object-oriented programming (OOP) that allows you to define a new class based on an existing class. The new class, called the derived class or subclass, inherits the properties, methods, and behavior of the existing class, known as the base class or superclass. This enables code reuse and promotes a hierarchical structure for organizing and extending classes.

To establish an inheritance relationship between classes in C#, you use the colon (:) symbol followed by the name of the base class after the derived class declaration. The syntax for defining a derived class that inherits from a base class is as follows:

```
class DerivedClass : BaseClass
{
// Additional members and methods specific to the derived class
}
```

Here, DerivedClass is the name of the new class you're creating, and BaseClass is the name of the existing class from which you want to inherit.

The derived class automatically acquires all the public and protected members (fields, properties, and methods) of the base class. It can also add new members or override existing members of the base class to provide specialized behavior. This process is known as extending or overriding the base class functionality.

Types of inheritance:

1. Single Inheritance: Single inheritance refers to a derived class inheriting from a single base class. C# supports single inheritance, where a class can have only one direct base class.

2. Multilevel Inheritance: Multilevel inheritance involves creating a chain of derived classes, where each derived class inherits from a base class, and subsequent derived classes inherit from those derived classes. This creates a hierarchical structure of inheritance.

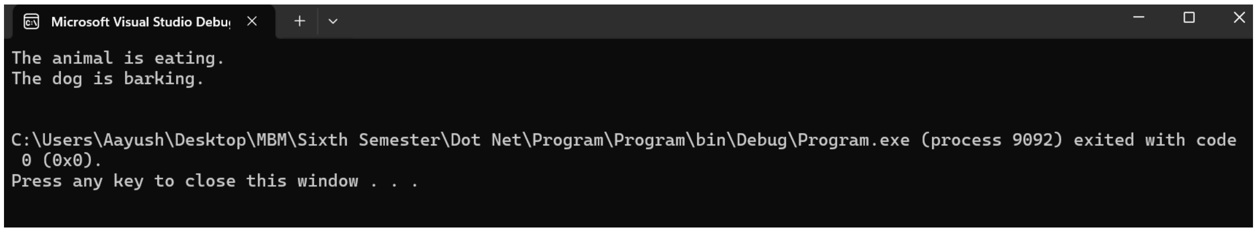
3. Hierarchical Inheritance: Hierarchical inheritance occurs when multiple derived classes inherit from a single base class. Each derived class shares common characteristics and behavior from the base class while adding its own specific features.

4. Multiple Interface Inheritance: C# supports multiple interface inheritance, where a class can implement multiple interfaces. An interface defines a contract specifying a set of methods and properties. By implementing multiple interfaces, a class can exhibit behavior from different sources.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
// Base class
    class Animal
    {
            public void Eat()
            {
                Console.WriteLine("The animal is eating.");
            }
    }

   // Derived class inheriting from Animal
     class Dog : Animal
    {
            public void Bark()
            {
               Console.WriteLine("The dog is barking.");
            }
    }

    class Program
    {
            static void Main(string[] args)
            {
             // Create an instance of the derived class
                    Dog myDog = new Dog();

            // Access base class method
            myDog.Eat();

            // Access derived class method
            myDog.Bark();

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

Output:

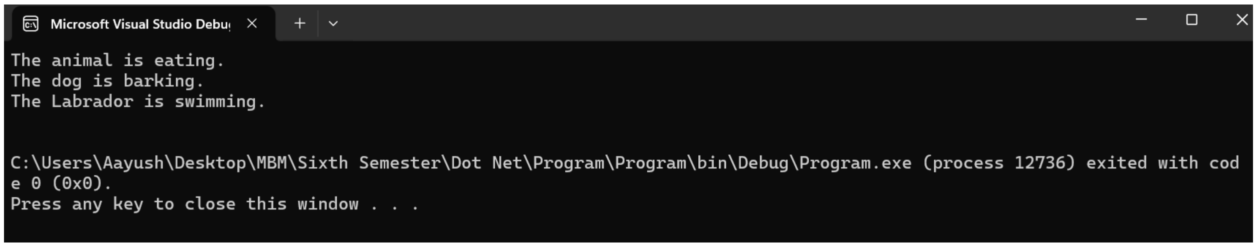


```
The animal is eating.
The dog is barking.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 9092) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // Base class
    class Animal {
            public void Eat()
            {
               Console.WriteLine("The animal is eating.");
            }
    }

   // Derived class inheriting from Animal
    class Dog : Animal
    {
            public void Bark()
            {
              Console.WriteLine("The dog is barking.");
            }
    }

    // Derived class inheriting from Dog
    class Labrador : Dog
    {
            public void Swim() {
              Console.WriteLine("The Labrador is swimming.");  }
            }

    class Program {
            static void Main(string[] args) {
             // Create an instance of the derived class
            Labrador myLabrador = new Labrador();

            // Access base class method
            myLabrador.Eat();

             // Access intermediate derived class method
            myLabrador.Bark();

          // Access derived class method
            myLabrador.Swim();

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

Output:

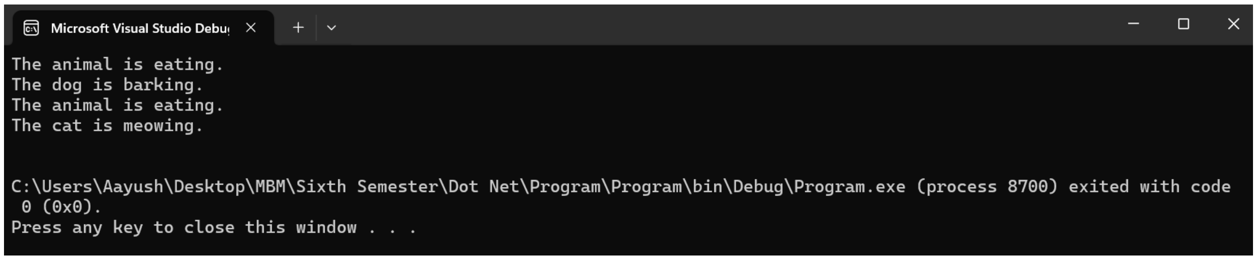


```
The animal is eating.
The dog is barking.
The Labrador is swimming.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 12736) exited with cod
e 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // Base class
    class Animal
    {
            public void Eat() {
              Console.WriteLine("The animal is eating.");
            }
    }
   // Derived class inheriting from Animal
    class Dog : Animal
    {
            public void Bark() {
             Console.WriteLine("The dog is barking.");
            }
    }
    // Another derived class inheriting from Animal
    class Cat : Animal
    {
            public void Meow() {
             Console.WriteLine("The cat is meowing.");
            }
    }
    class Program
    {
            static void Main(string[] args) {
        // Create instances of the derived classes
            Dog myDog = new Dog();
            Cat myCat = new Cat();

       // Access base class method from Dog
            myDog.Eat();

       // Access derived class method from Dog
            myDog.Bark();

       // Access base class method from Cat
            myCat.Eat();

        // Access derived class method from Cat
            myCat.Meow();

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

Output:

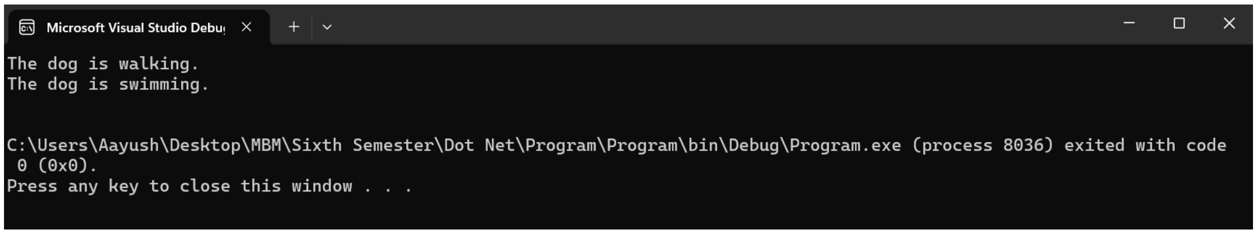


```
The animal is eating.
The dog is barking.
The animal is eating.
The cat is meowing.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 8700) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace inheritance
{
    // First interface
    interface IWalkable
    {
            void Walk();
     }

    // Second interface
    interface ISwimmable
    {
            void Swim();
    }

     // Class implementing both interfaces
    class Dog : IWalkable, ISwimmable
    {
            public void Walk()
            {
               Console.WriteLine("The dog is walking.");
            }

            public void Swim()
            {
              Console.WriteLine("The dog is swimming.");
            }
    }

    class Program
    {
            static void Main(string[] args)
            {
               // Create an instance of the class
            Dog myDog = new Dog();

           // Access methods from both interfaces
            myDog.Walk();
            myDog.Swim();

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

Output:



Microsoft Visual Studio Debu

```
The dog is walking.
The dog is swimming.


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 8036) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Lab 14: Date:

Title: Structs and Enums

Introduction:

Structs:

C# struct also known as C# structure is a simple user-defined type, a lightweight alternative to a class. Similar to classes, structures have behaviors and attributes. C# structs support access modifiers, constructors, indexers, methods, fields, nested types, operators, and properties.

Example:

```
Struct Books{
        public string title;
        public string subject;
        public int id;
};
```

Enums:

An enumeration is a set of named integer constants. An enumerated type is declared using the Enum keyword. C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

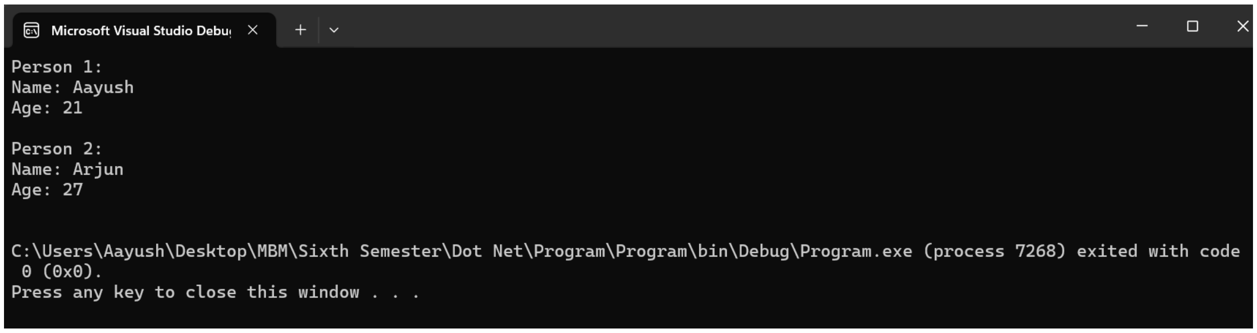
Declaring Enum variable syntax:

```
enum <enum_name>{
        enumeration list
};
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace structure
{
    struct Person
    {
            public string Name;
            public int Age;
    }

    class Program
    {
            static void Main(string[] args)
            {
              // Create a new instance of the Person struct
            Person person1;
            person1.Name = "Aayush";
            person1.Age = 21;

           // Access and display the struct members
          Console.WriteLine("Person 1:");
           Console.WriteLine("Name: " + person1.Name);
            Console.WriteLine("Age: " + person1.Age);

              // Create another instance of the Person struct
              Person person2 = new Person();
             person2.Name = "Ram";
             person2.Age = 30;

               Console.WriteLine("\nPerson 2:");
                Console.WriteLine("Name: " + person2.Name);
              Console.WriteLine("Age: " + person2.Age);

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

Output:

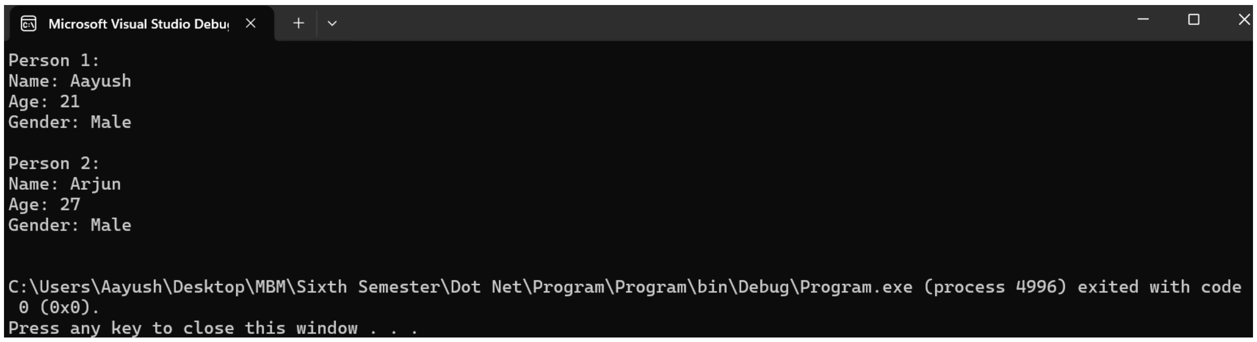


```
Person 1:
Name: Aayush
Age: 21

Person 2:
Name: Arjun
Age: 27


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 7268) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace enumuration
{
    enum Gender
    {
            Male,
            Female,
            Other
    }
    class Program
    {
            static void Main(string[] args)
            {
              // Create a new instance of the Person struct
           string person1Name = "Aayush";
            int person1Age = 21;
          Gender person1Gender = Gender.Male;

             // Access and display the enum members
           Console.WriteLine("Person 1:");
           Console.WriteLine("Name: " + person1Name);
            Console.WriteLine("Age: " + person1Age);
            Console.WriteLine("Gender: " + person1Gender);

          // Create another instance of the Person struct
            string person2Name = "Arjun";
            int person2Age = 27;
          Gender person2Gender = Gender.Male;

                Console.WriteLine("\nPerson 2:");
                Console.WriteLine("Name: " + person2Name);
                 Console.WriteLine("Age: " + person2Age);
               Console.WriteLine("Gender: " + person2Gender);

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

Output:



```
Person 1:
Name: Aayush
Age: 21
Gender: Male

Person 2:
Name: Arjun
Age: 27
Gender: Male


C:\Users\Aayush\Desktop\MBM\Sixth Semester\Dot Net\Program\Program\bin\Debug\Program.exe (process 4996) exited with code
 0 (0x0).
Press any key to close this window . . .
```

Lab 15:

Title: Razor Implementation

Introduction:

Razor:

In the realm of net-centric computing, Razor stands as a prominent player, leveraging its expertise to provide cutting-edge solutions and services. With a focus on connectivity, Razor offers a range of products and technologies that enable seamless integration and efficient communication within networked environments. Their innovative networking solutions empower businesses and individuals to harness the full potential of net-centric computing, facilitating data exchange, collaboration, and streamlined operations. Razor's commitment to delivering reliable, scalable, and secure solutions has earned them a reputation for excellence in the ever-evolving landscape of net-centric computing, driving progress and enabling digital transformation for a connected world.

MVC:

MVC, which stands for Model-View-Controller, is a software architectural pattern widely used in the development of web and desktop applications. The MVC pattern separates an application into three interconnected components: the model, the view, and the controller. The model represents the application's data and business logic, handling data manipulation and storage. The view handles the presentation of data to the user, providing the visual interface and user interaction elements. The controller acts as the intermediary between the model and the view, receiving user input, updating the model, and updating the view accordingly. This separation of concerns promotes modularity, maintainability, and reusability, allowing developers to efficiently manage and evolve complex applications. The MVC pattern has become a cornerstone of modern software development, providing a structured approach to building robust and scalable applications.

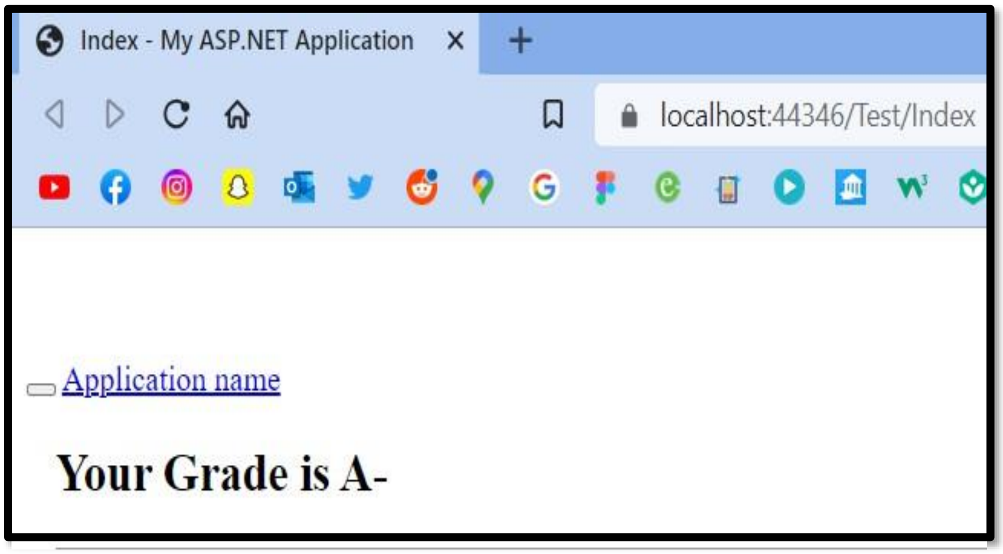
Controller Code:

```
 using System;
 using System.Collections.Generic;
using System.Linq; using System.Web;
 using System.Web.Mvc;
namespace Lab_15.Controllers {
            public class TestController : Controller {
            // GET: Test
             public ActionResult  Index() {
             return View();
            }
      }
}
```

View Code:

```
@{
            int marks = 80;
            if(marks >=80){
                        <h2> Your Grade is A- </h2>
            } else {
                        <h2> Your Grade is A<.h2>
            }
}
```

Output:



Your Grade is A-

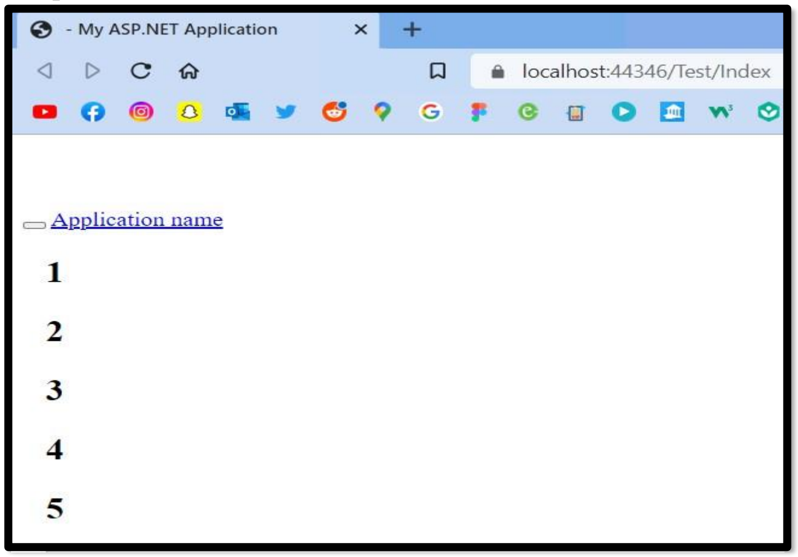
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
 using System.Web;
using System.Web.Mvc;
namespace Lab_15.Controllers {
        public class TestController : Controller {
         // GET: Test
        public ActionResult  Index() {
                return View();
                 }
        }
 }
```

View Code:

```
<!--Loops-->
 @{
        For(int i; i<5;i++){
             <h2>@i</h2>
        }
}
```

Output:



Application name

```
1
2
3
4
5
```

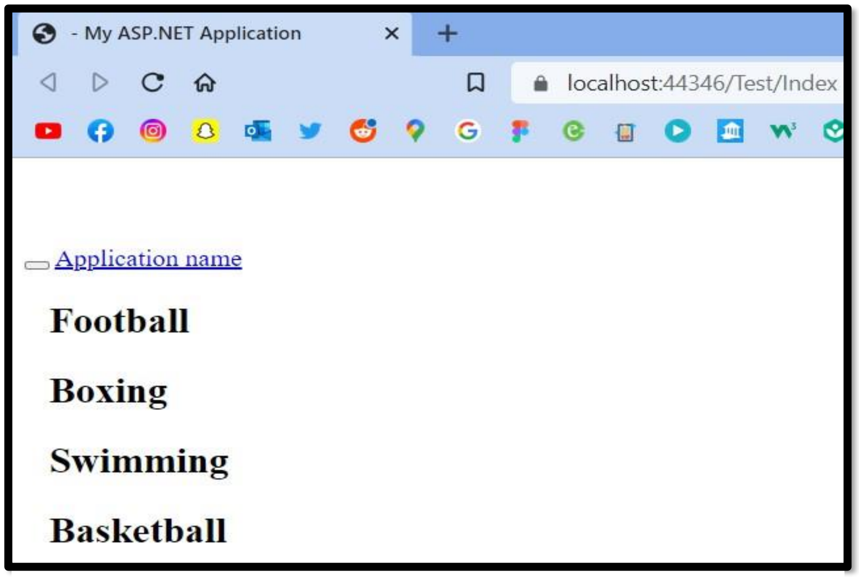
Controller Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace Lab_15.Controllers {
public class TestController : Controller {
        // GET: Test
        public ActionResult  Index() {
        Return View();
        }
    }
 }
```

View Code:

```
@{
 String[] sports = { "Football", "Boxing", "Swimming", "Basketball" };
foreach(String i in sports) {
                <h2>@i</h2>
} }
```

Output:



Application name

Football

Boxing

Swimming

Basketball

Lab 16:

Title: Action Method

Introduction:

The Action method in C# Razor is a helper method used to generate a URL for a specific action within a controller. It is commonly used in conjunction with other Razor helper methods, such as Html.ActionLink or Html.BeginForm, to create links or forms that trigger specific actions in a controller. The Action method takes parameters such as the name of the action method, the name of the controller, and any additional route parameters required by the action method. By utilizing the Action method, developers can dynamically generate URLs that correspond to specific actions in their ASP.NET MVC or Razor Pages applications.

Controller Code:

```
using System;
 using System.Collections.Generic;
using System.Linq; using System.Web;
using System.Web.Mvc;
namespace Lab_16.Controllers {
public class HomeController : Controller {
         // GET: Home
         public ActionResult Index() {
         return View();
         }
         public String show() {
        return "This is a second action method of home controller";
        }
         public ActionResult aboutus() {
         return View();
         }
         public int studentID(int id) {
         return id;
        }
    }
}
```

View Code (Index.cshtml)

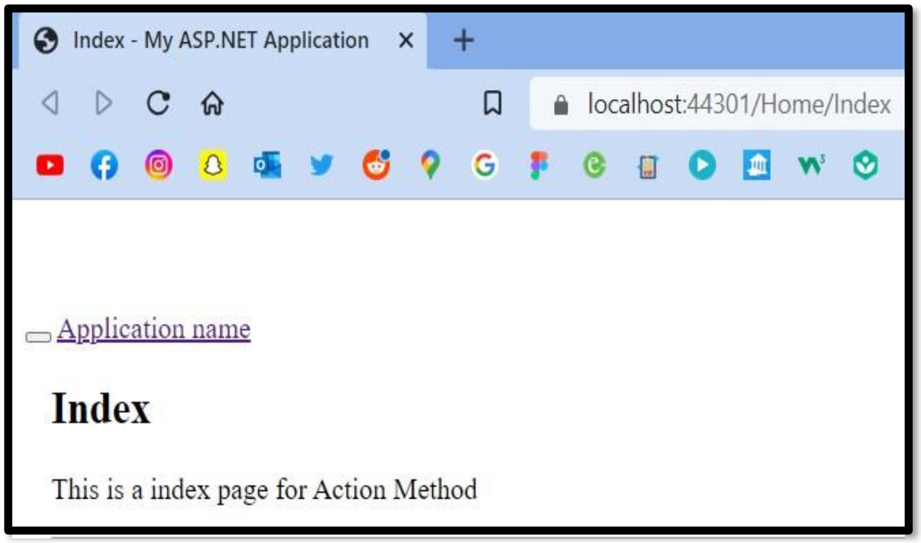
```
@{
 ViewBag.Title = "Index";
}

<h2>Index</h2>

<p> This is a index page for Action Method</p>
```

View Code (about.cshtml)

```
@{
        ViewBag.Title = "aboutus";
 }

<h2>AboutUs</h2>
<p>This is the AboutUs page for Action Method</p>
```

Output:

Index:



Index - My ASP.NET Application

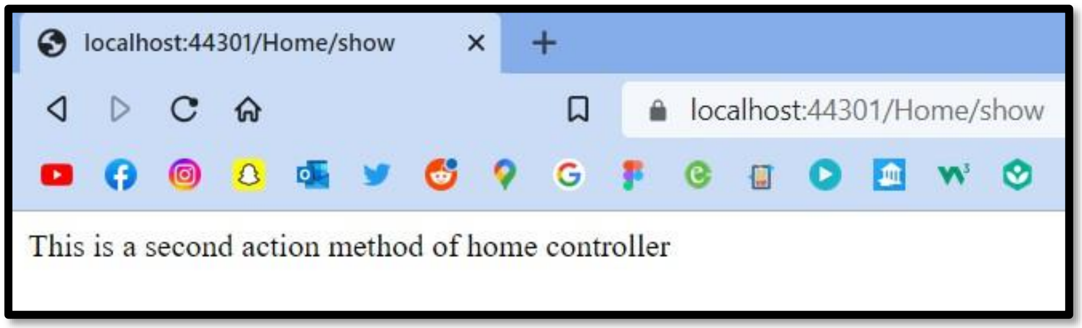
localhost:44301/Home/Index

Application name

Index

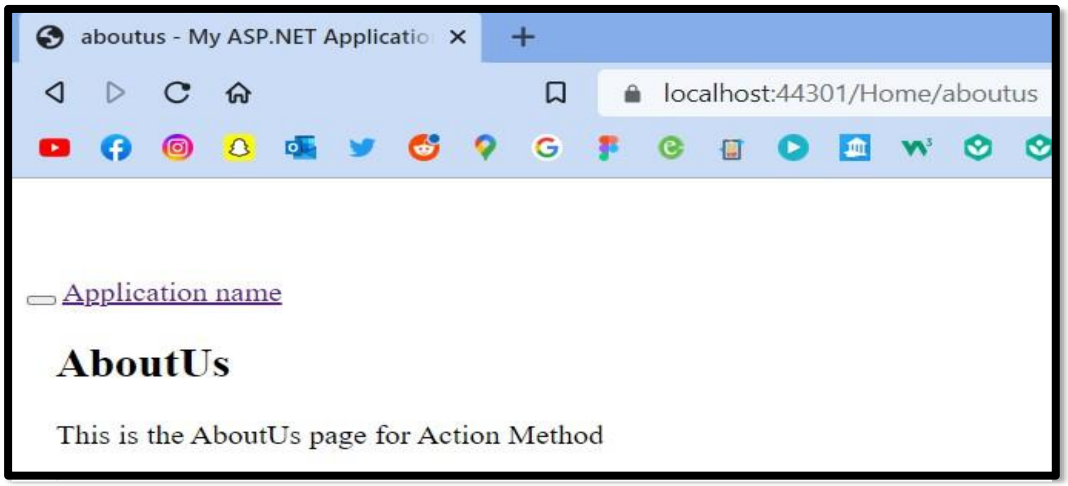
This is a index page for Action Method

Show:



This is a second action method of home controller

AboutUs:



AboutUs

This is the AboutUs page for Action Method

Lab 17:

Title: Tag Helper (TextBox)

Introduction:

The purpose of this lab report is to introduce the Tag Helper HTML.Textbox, a sophisticated tool designed to revolutionize the creation and functionality of web forms. With its advanced features and intuitive user interface, this tool enables developers to incorporate tagging functionality seamlessly, providing users with a more structured and organized approach to data entry. By implementing the Tag Helper HTML.Textbox, researchers and web developers can enhance the efficiency and effectiveness of their web applications, whether they are utilized in blogging, e-commerce, or collaborative platforms.

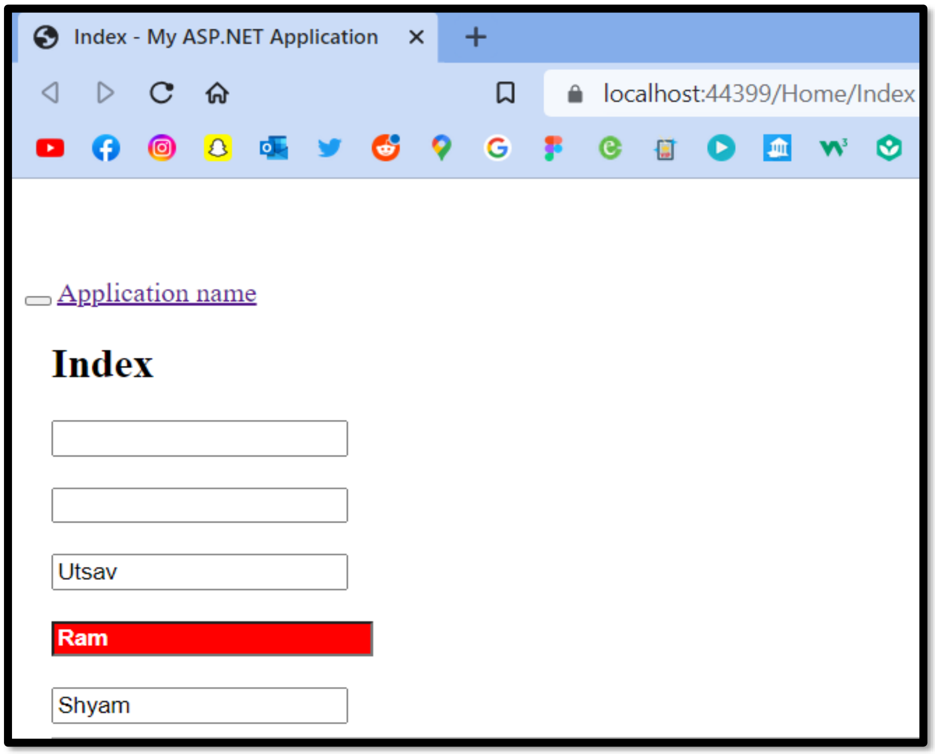
Controller Code:

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

namespace Lab_17.Controllers{
            public class HomeController : Controller
            {
           // GET: Home
            public ActionResult Index()
            {
            return View();
            }
      }
}
```

View Code:

```
@{
        ViewBag.Title = "Index";
}

<h2>Index</h2>
<input type="text" name="fullname" id="fullname" value=""
/>
<br />
<br />
@H tml.TextBox("fullname")
<br />
<br />
@Html.TextBox("fullname", "Utsav")
<br />
<br />
@Html.TextBox("fullname", "Ram", new { style = "background-color:Red;
color:White; font-weight:bold" })
<br />
<br />
@Html.TextBox("fullname","Shyam", new {@class = "form- control", @readonly = true
})
```

Ouput:


Index - My ASP.NET Application

localhost:44399/Home/Index

Application name

Index

Utsav

Ram

Shyam

Lab 18:

Title: Tag Helper (HtmlForm)

Introduction:

The objective of this lab report is to introduce HTML.Form in Tag Helper, a powerful tool that simplifies the creation of interactive and dynamic forms in HTML. With its intuitive and user-friendly syntax, HTML.Form in Tag Helper enables developers to generate form elements effortlessly while incorporating essential functionalities such as validation, data binding, and styling. This lab report aims to explore the features and capabilities of HTML.Form in Tag Helper, showcasing its ability to streamline form development and enhance the user experience.

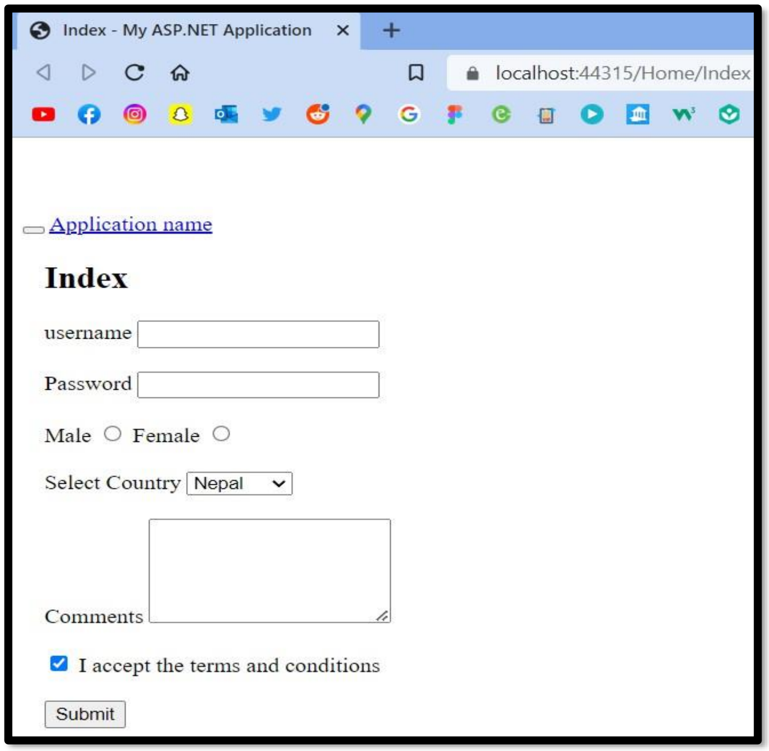
Controller Code:

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

namespace Lab_18.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

View Code:

```
<!--Html.Form-->
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2> @{
     Html.BeginForm("Index", "H ome");
}
<!--Html.Username-->
@H tml.Label("username")&nbsp;@H tml.TextBox("Username")
<br />
<br />
<!--Html.password-->
@H t ml.Label("flassword")&nbsp;@Html.Password(“Password”)
 <br />
<br />
<!--Html.RadioButton-->
@H tml.Label("Male")&nbsp;@H tml.RadioButton("Gender", "Male")
@Html.Label("Female")&nbsp;@Html.RadioButton("Gender", "Female")
<br />
<br />
<!--Html.RadioButton-->
@Html.Label("Select
Country")finbsp;@H tml.DropDownList("Country List", new
SelectList(new[] { "Nepal", "India", "Pakistan", "Srilanka" },
"choose"))
<br />
<br />
<!--Html.TextArea-->
@Html.Label("Comments")&nbsp;@Html.TextArea("CommentArea", "", 5, 20, new
{ @class = "form-control" })
<br />
<br/>
<!—Html.TextArea-->
@Html.CheckBox("Acceptterms",true) @Html.Label("I accept the terms and conditions")
@Html.Hidden("Id")
<br/>
<br/>
<!—Html.SubmitButton-->
<! – Not in Html Helper -->
<input type=”submit” value=”Submit”/>
@{
Html.EndForm();
}
```

Output:


Index - My ASP.NET Application

localhost:44315/Home/Index

Application name

Index

username

Password

Male ○ Female ○

Select Country Nepal

Comments

✓ I accept the terms and conditions

Submit

Lab 19:

Title: Data Annotation

Introduction:

Data annotation is a critical component of data processing and analysis, and its significance extends to various programming languages, including C#. In the realm of C# development, data annotation plays a crucial role in ensuring data integrity, validation, and overall quality. By leveraging data annotation techniques, developers can enhance the reliability and usability of their C# applications.

One of the primary uses of data annotation in C# is in validating user input. By applying annotations to properties or fields within classes, developers can specify constraints and rules that the data must adhere to. For example, using attributes such as Required, StringLength, or RegularExpression, developers can enforce that certain fields must be filled, set maximum or minimum length limits, or ensure that data matches a specific pattern. These annotations help prevent invalid or inconsistent data from entering the system, improving data quality and minimizing errors.

Controller Code:

```
using Lab_19.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Lab_19.Controllers
{

    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
        [H ttpflost]
        public ActionResult Index(Employee e)
        {
            if(ModelState.IsValid==true)
            {
                ViewData["SuccessMessage"] =
                "<script>alert('Data has been submitted!!!')</script>";
                ModelState.Clear();
            }
            return View();
        }}}
```

Model Code:

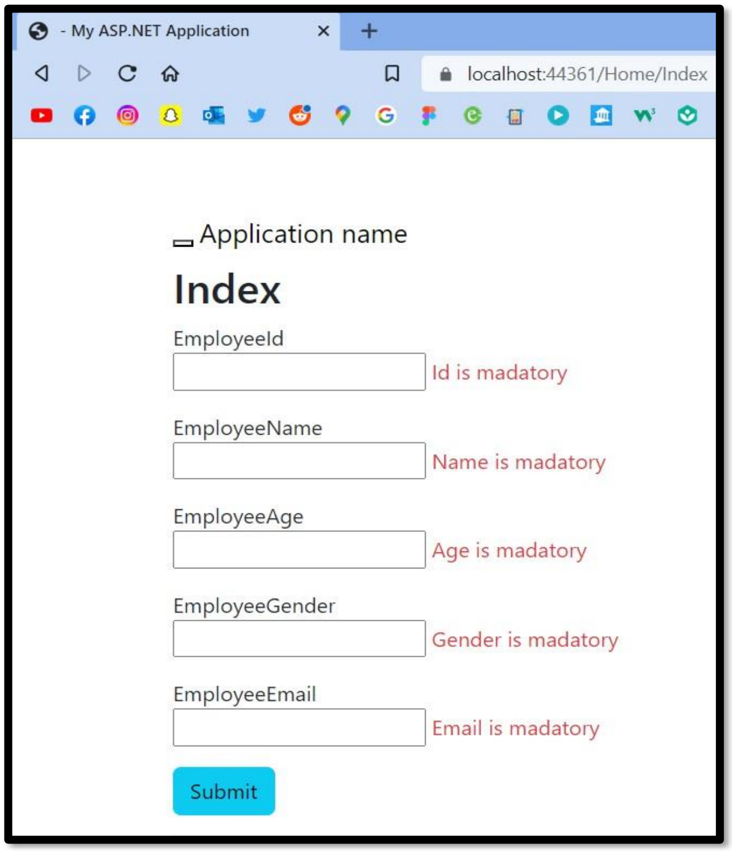
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web; namespace Lab_19.Models
{
    public class Employee{
        [Required(ErrorMessage ="Id is madatory")]
        public int EmployeeId { get; set; }
        [Required(ErrorMessage = "Name is madatory")]
        public string Employee ame { get; set; }
        [Required(ErrorMessage = "Age is madatory")]
        public int EmployeeAge { get; set; }
        [Required(ErrorMessage = "Gender is madatory")]
        public string EmployeeGender { get; set; }
        [Required(ErrorMessage = "Email is madatory")]
         public string EmployeeEmail { get; set; }
    }
}
```

View Code:

```
  @model Lab_19.Models.Employee
@{
    ViewBag.Tilte = "Index";
}
<h2>Index</h2>
@using (Html.BeginForm())
{
    <p>
        @Html.LabelFor(model  =>  model.EmployeeId)&nbsp;<br
/>@H tml.TextBoxFor(model => model.EmployeeId, "", new
{@class="format-control"})
        @Html.ValidationMessageFor(Model =>
Model.EmployeeId, "", new { @class = "text-danger" })
    </p>

   <p>
       @Html.LabelFor(model  =>
model.EmployeeName)&nbsp;<br />@H tml.TextBoxFor(model
=> model.EmployeeName, "", new { @class = "format-
control" })
        @Html.ValidationMessageFor(Model =>
Model.EmployeeName, "", new { @class = "text-danger" })
    </p>
    <p>
        @Html.LabelFor(model =>
model.EmployeeAge)&nbsp;<br />@H tml.TextBoxFor(model  =>
model.EmployeeAge, "", new { @class = "format-control" })
```

```
          @Html.ValidationMessageFor(Model =>
Model.EmployeeAge, "", new { @class = "text-danger" })
     </p>
     <p>
          @Html.LabelFor(model =>
model.EmployeeGender)&nbsp;<br />@ H tml.TextBoxFor(model
=> model.EmployeeGender, "", new { @class = "format-
control"
})
          @Html.ValidationMessageFor(Model =>
Model.EmployeeGender, "", new{@class = "text-danger"})
     </p>
     <p>
          @H tml.LabelFor(model =>
model.EmployeeEmail)&nbsp;<br />@ Html.TextBoxFor(model
=> model.EmployeeEmail, "", new { @class = "format-control"
})
          @Html.ValidationMessageFor(Model =>
Model.EmployeeEmail,"",new { @class = "text-danger" })
     </p>
     <input type="Submit" value="Submit" class="btn btn-
info"/>
     @Html.Raw(ViewData["SuccessMessage"])
}
```

Output:



Index

EmployeeId
Id is madatory

EmployeeName
Name is madatory

EmployeeAge
Age is madatory

EmployeeGender
Gender is madatory

EmployeeEmail
Email is madatory

Lab 20:

Title: URL Routing

Introduction:

URL routing is a critical aspect of web development that ensures seamless navigation and functionality within web applications. When a user enters a URL or clicks on a link, the URL routing mechanism directs the request to the appropriate destination within the application. This destination can be a specific web page, a resource such as an image or file, or a handler that executes a particular action. To achieve this, developers establish a set of rules and patterns that map incoming URLs to their corresponding endpoints.

URL routing offers numerous benefits in web application development. Firstly, it enables developers to create dynamic and user-friendly applications by allowing users to access different pages or perform specific actions with ease. Instead of relying solely on complex query strings or hard-coded URLs, URL routing provides a more intuitive and readable way to interact with the application. Additionally, by organizing the application's functionality into distinct routes, developers can efficiently manage and maintain the codebase, enhancing the application's scalability and ease of future updates.

Home Controller

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

namespace Lab_20.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
```

```
          public ActionResult Student()
          {
               return View();
          }
     }
}
```

Index.cshtml

```
@{
     Layout = null;
}

<!DOCTYPE html>

<html>
<head>
     <meta  name="viewport"  content="width=device-width"  />
     <title>Index</title>
</head>
<body>
     <div>
          <h1>Student Index Action for URL Routing</h1>
     </div>
</body>
</html>
```

Student.cshtml

```
  @{
     Layout = null;
}

<!DOCTYPE html>

<html>
<head>
     <meta  name="viewport"  content="width=device-width"  />
     <title>Student</title>
</head>
<body>
     <div>
          <h1>Hello This is Student action for routing2</h1>
     </div>
</body>
</html>
```

StudentController.cs

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

namespace UrlRouting.Controllers
{
    public class StudentController : Controller
    {
        // GET: Student
        public ActionResult StudentIndex()
        {
            return View();
        }
        public ActionResult Index(into id)
        {
            return Content(id.ToString());
        }
 }
```

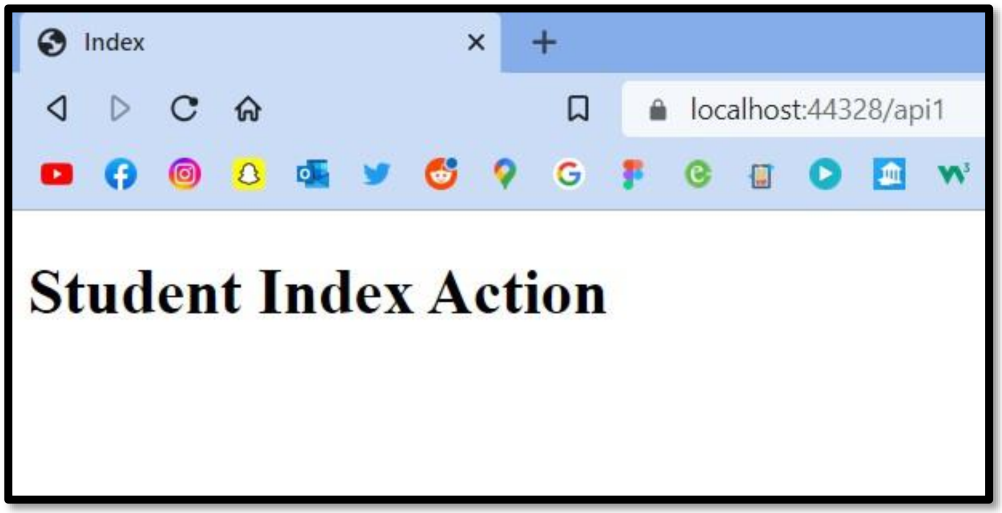
Studentindex.cshtml

```
@{
        Layout = null;
   }

   <!DOCTYPE html>

   <html>
   <head>
        <meta name="viewport" content="width=device-width" />
        <title>Index</title>
   </head>
   <body>
        <div>
            <h1>Student Index Action</h1>
        </div>
   </body>
   </html>
```

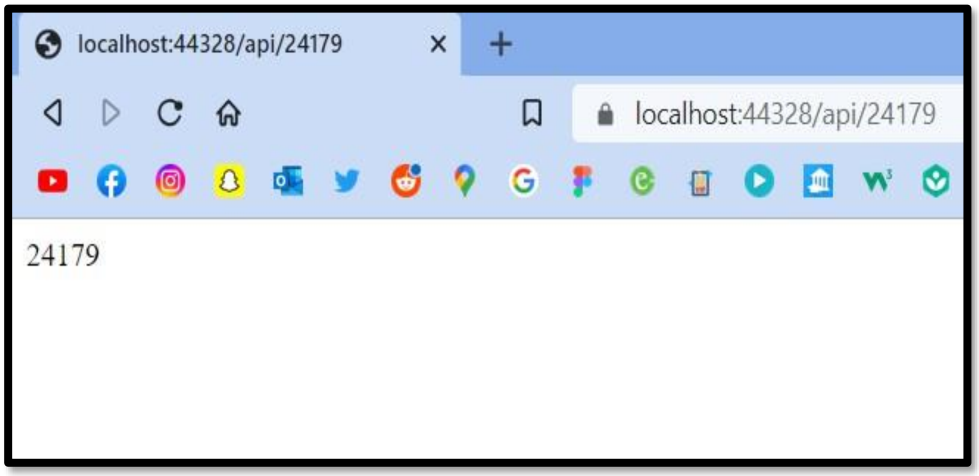
RouteConfig.cs

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

namespace Lab_20
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection
routes)
        {

routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

                routes.MapRoute(
                    name: "Default2",
                    url: "api1/",
                    defaults: new { controller = "Student",
                    action = "studentindex",
id=Urlflarameter.Optional }
               );
               routes.MapRoute(
                name: "Default1",
               url: "api/{id}",
                defaults: new { controller = "Student", action = "Index",
       id = Urlflarameter.Optional },
                 constraints: new { id = @"\d+" }
                    );
               routes.MapRoute(
                name: "Default",
                   url: "{controller}/{action}/{id}",
                   defaults: new { controller = "Student",
action = "Index", id = UrlParameter.Optional }
                   );
        }
    }
}
```

Output:



Student Index Action



```
24179
```

Lab 21:

Title: Attribute Based Routing

Introduction:

Attribute-based routing is a powerful and flexible approach to routing and message handling in modern software applications. It enables developers to define routing rules based on specific attributes or properties of incoming messages, allowing for dynamic and granular control over how messages are processed and dispatched. By decoupling routing logic from the underlying infrastructure, attribute-based routing offers increased modularity, extensibility, and adaptability, making it an invaluable tool in building scalable and resilient systems.

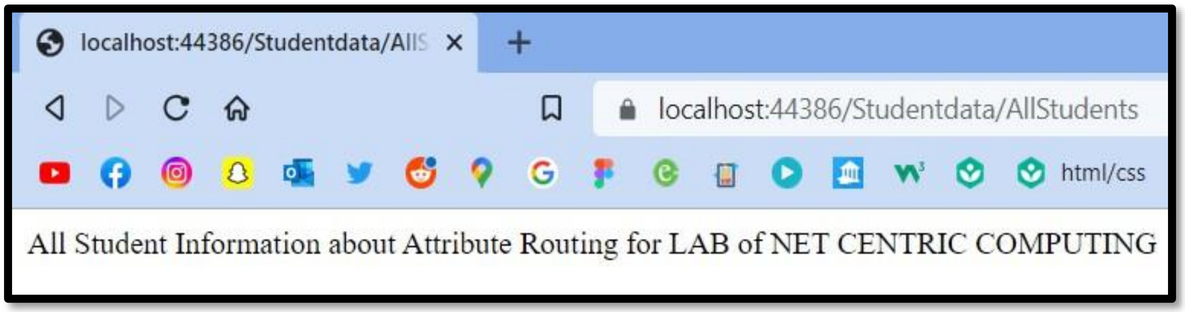
Controller Code:

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

namespace Lab_21.Controllers
{
    [RoutePrefix("StudentData")]
    public class StudentController : Controller
    {
        // GET: Student
        [Route("AllStudents")]
        public ActionResult Index()
        {
            return Content("All Student Information about Attribute
Routing for LAB of  NET CENTRIC COMPUTING");
        }
        [Route("StudentByRoll/{rolln}")]
        public ActionResult studentRollNo(int rolln)
        {
            if(rolln>0)
            {
                return Content("Valid Roll Number");
            }
            else
            {
                return Content("Invalid Roll Number");
            }
        }
}}
```

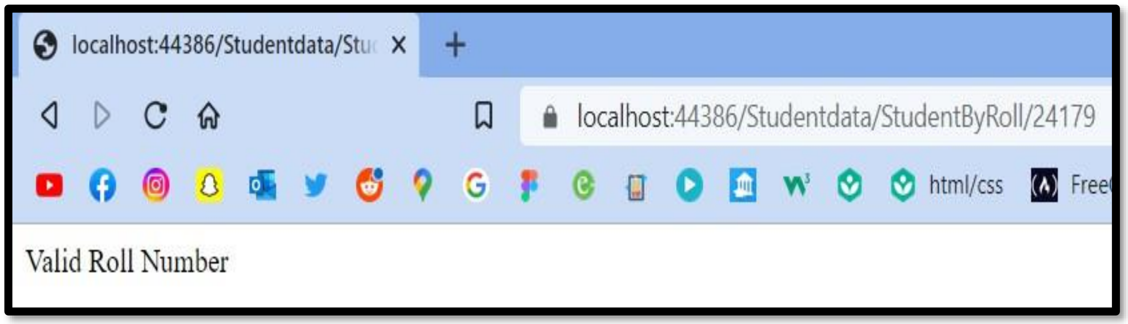
RouteConfig Code:

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

namespace Lab_21
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection
routes)
        {

routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapMvcAttributeRoutes();
            routes.MapRoute(
                name: "Default",
                url:  "{controller}/{action}/{id}",
                defaults: new { controller = "Home",
                action= "Index", id = Urlflarameter.Optional }
            );
        }
    }
}
```

Output


localhost:44386/Studentdata/AllS
localhost:44386/Studentdata/AllStudents
html/css

All Student Information about Attribute Routing for LAB of NET CENTRIC COMPUTING


localhost:44386/Studentdata/Stu
localhost:44386/Studentdata/StudentByRoll/24179
html/css
Free

Valid Roll Number

Lab 22:

Title: Temp Data

Introduction:

In modern web development, it is crucial to handle and transfer data effectively between different components of an application. One of the key challenges is maintaining data across multiple requests and actions. ASP.NET MVC provides a powerful mechanism called TempData to address this requirement.

The purpose of this lab report is to showcase the implementation of TempData in an MVC Web API. TempData is a container provided by ASP.NET MVC that allows developers to store and retrieve data between requests. It is particularly useful when we need to pass data from one action to another during a user's session.

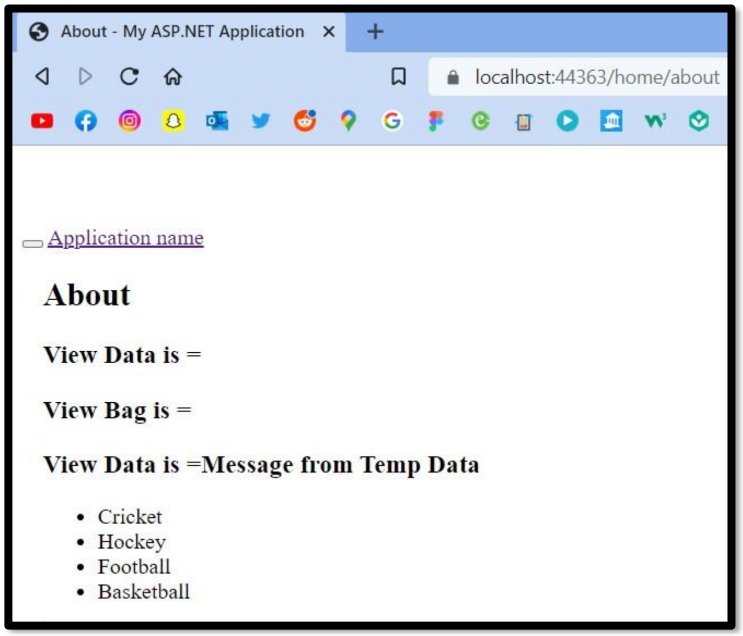
Controller Code:

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

namespace Lab_22.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            ViewData["Var1"] = "Message from View Data";
             ViewBag.Var2 = "Message from View Bag";
            TempData["Var3"] = "Message from Temp Data";
            string[] games = { "Cricket", " H ockey","Football",
            "Basketball" };
            TempData["GamesArray"] = games;
            return View();
        }
        public ActionResult About()
        {
            return View();
        }
    }
}
```

View Code:

```
@{
    ViewBag.Title = "About";
}

<h2>About</h2>
<h3>View Data is =@ViewData["Var1"]</h3>
<h3>View Bag is =@ViewBag.Var2</h3>
<h3>View Data is =@TempData["Var3"]</h3>
<ul>
    @{
        foreach(string i in (string[])
TempData["GamesArray"])
    {
        <li>@i</li>
         }
         }
     </ul>
```

Output:



Application name

About

View Data is =

View Bag is =

View Data is =Message from Temp Data

- Cricket
- Hockey
- Football
- Basketball