

Tribhuvan University
Bhaktapur Multiple Campus
Doodhpati-17, Bhaktapur, Nepal



Lab report II of
NET centric computing
(CSC 367)

Submitted by:

Saini Thapa Magar

B.Sc. CSIT 6th semester

Roll no.: 44

Submitted to:

Ramesh Kharbuja

C# Exception Handling

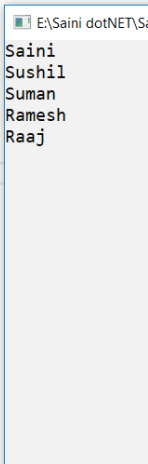
When executing C# code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things. When an error occurs, C# will normally stop and generate an error message. The technical term for this is: C# will throw an exception (throw an error).

C# try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block. The try and catch keywords come in pairs.

Output

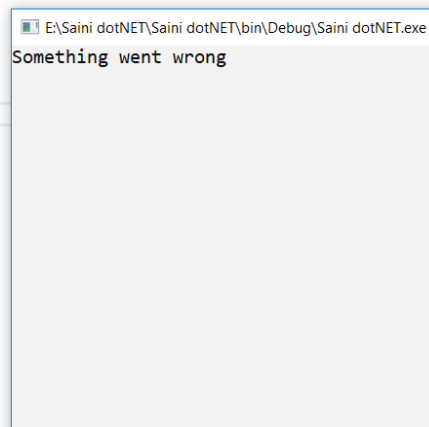
```
0 references
static void Main(string[] args)
{
    try
    {
        string[] Names = new string[5];
        Names[0] = "Saini";
        Names[1] = "Sushil";
        Names[2] = "Suman";
        Names[3] = "Ramesh";
        Names[4] = "Raaj";
        //Names[5] = "Santosh";
        foreach (string name in Names)
        {
            Console.WriteLine(name);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Something went wrong");
    }
    Console.ReadKey();
}
```



The screenshot shows a console window titled "E:\Saini dotNET\S..." with the following output:

```
Saini
Sushil
Suman
Ramesh
Raaj
```

```
0 references
static void Main(string[] args)
{
    try
    {
        string[] Names = new string[5];
        Names[0] = "Saini";
        Names[1] = "Sushil";
        Names[2] = "Suman";
        Names[3] = "Ramesh";
        Names[4] = "Raaj";
        Names[5] = "Santosh";
        foreach (string name in Names)
        {
            Console.WriteLine(name);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Something went wrong");
    }
    Console.ReadKey();
}
```



The screenshot shows a console window titled "E:\Saini dotNET\Saini dotNET\bin\Debug\Saini dotNET.exe" with the following output:

```
Something went wrong
```

C# Polymorphism

Polymorphism means “many forms”, and it occurs when we have many classes that are related to each other by inheritance. Polymorphism can be static or dynamic.

➤ Static Polymorphism

Static Polymorphism is the linking of a function with an object during compile time is called static. It is also called static binding. C# provides two techniques to implement static polymorphism i.e. Function overloading and Operator overloading.

➤ Dynamic Polymorphism

In static polymorphism, the response to a function is determined at the compile time. In dynamic polymorphism, it is decided at run-time. Dynamic polymorphism is implemented by abstract classes and virtual functions.

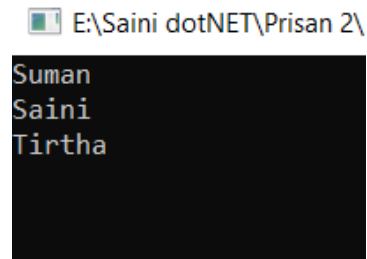
C# Indexer

An indexer allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([]).

```
namespace Prisan_2
{
    2 references
    class Indexer
    {
        private string[] names= new string[3];
        4 references
        public string this[int i]
        {
            get { return names[i]; }
            set { names[i] = value; }
        }
    }
    delegate int Calculate(int x, int y);

    0 references
    internal class Program
    {
        0 references
        static int Add(int x, int y)
        { return x + y; }
        0 references
        static void Main(string[] args)
        {
            //Indexer
            Indexer i = new Indexer();
            i[0] = "Suman";
            i[1] = "Saini";
            i[2] = "Tirtha";
            for (int element = 0; element < 3; element++)
            {
                Console.WriteLine(i[element]);
            }
            Console.ReadKey();
        }
    }
}
```

Output



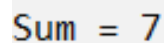
```
E:\Saini dotNET\Prisan 2\  
Suman  
Saini  
Tirtha
```

C# Delegate

C# delegates are similar to pointers to functions, in C or C++. A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime. Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

```
//Delegate  
Calculate calculator = Add;  
int result = calculator(3, 4);  
Console.WriteLine("Sum = " + result);  
Console.ReadKey();
```

Output



```
Sum = 7
```

C# ArrayList

The ArrayList class implements a collection of objects using an array whose size is dynamically increased as required. The ArrayList class is very useful if you are working with arrays and need to add, remove, index, or search for data in a collection.

Code

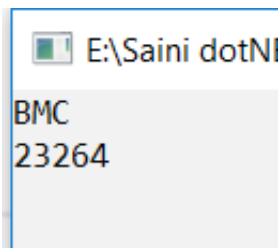
```
using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Prisan_2  
{  
    internal class Program
```

```

{
    static void Main(string[] args)
    {
        ArrayList mylist = new ArrayList();
        mylist.Add("BMC");
        mylist.Add(23264);
        foreach (var item in mylist)
        {
            Console.WriteLine(item);
        }
        Console.ReadKey();
    }
}

```

Output



C# Hash Table

The Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key. It uses the key to access the elements in the collection. A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Code

```

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

namespace Prisan_2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Hashtable mylist = new Hashtable();
            mylist.Add("College", "BMC");
            mylist.Add("Contact", 9821914747);
            mylist.Add("ismarried", false);
            foreach (var item in mylist.Keys)

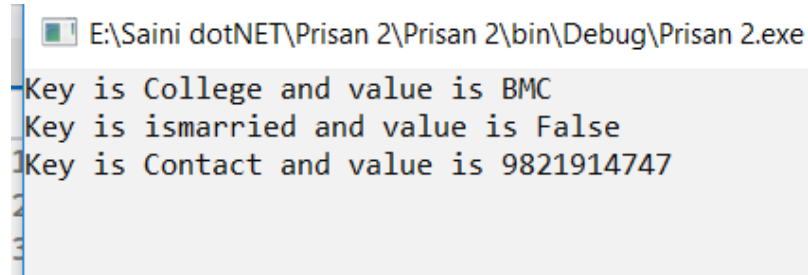
```

```

        {
            Console.WriteLine("Key is " + item + " and value is " +
mylist[item]);
        }
        Console.ReadKey();
    }
}
}

```

Output



```

E:\Saini dotNET\Prisan 2\Prisan 2\bin\Debug\Prisan 2.exe
Key is College and value is BMC
Key is ismarried and value is False
Key is Contact and value is 9821914747

```

C# List

List<T> class represents the list of objects which can be accessed by index. It comes under the System.Collections.Generic namespace. List class can be used to create a collection of different types like integers, strings etc. List<T> class also provides the methods to search, sort, and manipulate lists.

Code:

```

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

namespace practical12
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> rollno = new List<int>();
            rollno.Add(36);
            rollno.Add(37);
            rollno.Add(38);
            rollno.Add(39);
            rollno.Add(40);
            rollno.Add(41);
            foreach (var item in rollno)
            {
                Console.WriteLine("Roll Num is " + item);
            }
            Console.ReadKey();
        }
    }
}

```

Output

```
Roll Num is 35
Roll Num is 37
Roll Num is 38
Roll Num is 39
Roll Num is 40
Roll Num is 41
```

C# Files

Working With Files

The File class from the System.IO namespace, allows us to work with files. The File class has many useful methods for creating and getting information about files. For example:

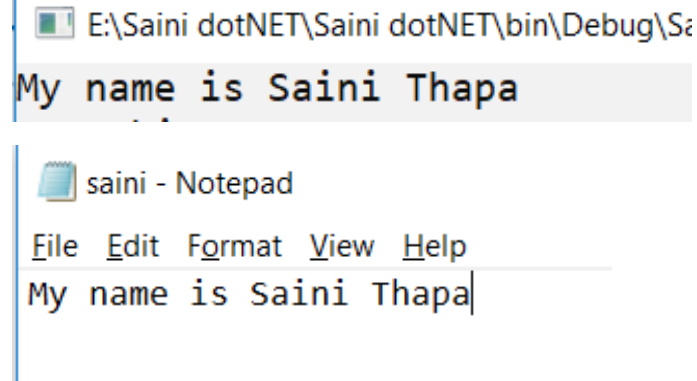
Methods	Description
AppendText()	Appends text at the end of an existing file
Copy()	Copies a file
Create()	Creates or overwrites a file
Delete()	Deletes a file
Exists()	Tests whether the file exists
ReadAllText()	Reads the contents of a file
Replace()	Replaces the contents of a file with the contents of another file
writeAllText()	Creates a new file and writes the contents to it. If the file already exists, it will be overwritten

Code

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

namespace practical13
{
    internal class Program
    {
        static void Main(string[] args)
        {
            string pathName = @"suraj.txt";
            string writeText = "My name is Saini Thapa";
            File.WriteAllText("saini.txt", writeText);
            string readText = File.ReadAllText("saini.txt");
            Console.WriteLine(readText);
            Console.ReadKey();
        }
    }
}
```

Output



Asynchronous Programming in C#

Asynchronous programming in C# is an efficient approach towards activities blocked or access is delayed. If an activity is blocked like this in a synchronous process, then the complete application waits and it takes more time. The application stops responding. Using the asynchronous approach, the applications continue with other tasks as well. The `async` and `await` keywords in C# are used in async programming. Using them, you can work with .NET Framework resources, .NET Core, etc. Asynchronous methods defined using the `async` keyword are called async methods.

Code

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

namespace ASync
{
    internal class Program
    {
        public static async Task Function1()
        {
            await Task.Run(() =>
            {
                for(int i= 0; i < 5; i++)
                {
                    Console.WriteLine($"{i} number from function1");
                }
            });
        }
        public static async Task function2()
        {
            for(int i= 0;i<5; i++)
            Console.WriteLine("Counter from function 2");
        }
        static async Task Main()
    }
```



```

{
    Console.WriteLine("Starting program...");

    // Call a method asynchronously
    await Function1();
    function2();
    //await DoSomethingAsync();


    Console.WriteLine("Program finished.");
    Console.ReadKey();
}

static async Task DoSomethingAsync()
{
    Console.WriteLine("Starting async method...");

    // Simulate some work that takes 1 second
    await Task.Delay(1000);
    Console.WriteLine("Executing.....");
    await Task.Delay(1000);
    Console.WriteLine("3.....");
    await Task.Delay(800);
    Console.WriteLine("2.....");
    await Task.Delay(800);
    Console.WriteLine("1.....");
    await Task.Delay(1000);
    Console.WriteLine("done!");
    await Task.Delay(1000);
    Console.WriteLine("Async method finished.");
}
}

```

Output

 E:\Saini dotNET\ASync\ASync\bin\Debug\ASync.exe

```

Starting program...
0 number from function1
1 number from function1
2 number from function1
3 number from function1
4 number from function1
Counter from function 2
Counter from function 2
Counter from function 2
Counter from function 2
Counter from function 2
Program finished.

```