overloading program

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication3 {
   class Method_overloading {
      public int Addition(int a, int b) {
         int x;
         return x = a + b;
      }
      public int Addition(int a, int b, int c) {
         int y;
         return y = a + b + c;
      }
      public float Addition(float a, float b) {
         float u;
         return u = a + b;
      }
      public float Addition(float a, float b, float c) {
         float v;
         return v = a + b + c;
      }
   }
   //Now you can use those Addition method four types
   class hub {
      public static void Main(String[] args) {
         Method_overloading mthover = new Method_overloading();
         Console.WriteLine("Addition of two integers::::::::::::::::" + mthover.Addition(2, 5));
         Console.WriteLine("Addition of two double type values::::::" + mthover.Addition(0.40 f, 0.50
f));
         Console.WriteLine("Addition of three integers::::::::::::::" + mthover.Addition(2, 5, 5));
         Console.WriteLine("Addition of three double type values::::" + mthover.Addition(0.40 f, 0.50 f,
0.60 f));
         Console.ReadLine();
      }
   }
}
```

overriding program

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

namespace Hello_Word
{
    class baseClass
    {
        public virtual void Greetings()
        {
            Console.WriteLine("baseClass Saying Hello!");
        }
    }
    class subClass : baseClass
    {
        public override void Greetings()
        {
            base.Greetings();
            Console.WriteLine("subClass Saying Hello!");
        }
    }
    class Program
```

```
    {
        static void Main(string[] args)
        {
            baseClass obj1 = new subClass();
            obj1.Greetings();
            Console.ReadLine();
        }
    }
}
```

boxing and unboxing

```
class Test
{
static void Main()
{
int i = 1;
object o = i; // boxing
int j = (int) o; // unboxing
}
}
```

An int value can be converted to object and back again to int.

This example shows both boxing and unboxing. When a variable of a value type needs to be converted to a reference type, an object box is allocated to hold the value, and the value is copied into the box.

Unboxing is just the opposite. When an object box is cast back to its original value type, the value is copied out of the box and into the appropriate storage location.

.....string....

The .NET Framework allows strings to be created using simple assignment, and also overloads a class constructor to support string creation using a number of different parameters. The .NET Framework also provides several methods in the System.String class that create new string objects by combining several strings, arrays of strings, or objects.

Creating Strings Using Assignment
The easiest way to create a new String object is simply to assign a string literal to a String object.

Creating Strings Using a Class Constructor
You can use overloads of the String class constructor to create strings from character arrays. You can also create a new string by duplicating a particular character a specified number of times.

Methods that Return Strings
The following table lists several useful methods that return new string objects.

Method name	Use
String.Format	Builds a formatted string from a set of input objects.
String.Concat	Builds strings from two or more strings.
String.Join	Builds a new string by combining an array of strings.
String.Insert	Builds a new string by inserting a string into the specified index of an existing string.
String.CopyTo	Copies specified characters in a string into a specified position in an array of characters.

concat.....two string

```
string helloString1 = "Hello";
string helloString2 = "World!";
Console.WriteLine(String.Concat(helloString1, ' ', helloString2));
// The example displays the following output:
//      Hello World!
```

Join two string
The String.Join method creates a new string from an array of strings and a separator string. This method is useful if you want to concatenate multiple strings together, making a list perhaps separated by a comma.

The following example uses a space to bind a string array.

C#

Copy

```
string[] words = {"Hello", "and", "welcome", "to", "my" , "world!"};
Console.WriteLine(String.Join(" ", words));
// The example displays the following output:
//      Hello and welcome to my world!
```

insert command

```
string sentence = "Once a time.";
 Console.WriteLine(sentence.Insert(4, " upon"));
 // The example displays the following output:
 //      Once upon a time.
```

Here is the complete example that shows how to use stings in C# and .NET.

```
using System;
namespace CSharpStrings
{
   class Program
   {
      static void Main(string[] args)
      {
         // Define .NET Strings
         // String of characters
         System.String authorName = "Mahesh Chand";

         // String made of an Integer
         System.String age = "33";

         // String made of a double
         System.String numberString = "33.23";

         // Write to Console.
         Console.WriteLine("Name: {0}", authorName);
         Console.WriteLine("Age: {0}", age);
         Console.WriteLine("Number: {0}", numberString);
         Console.ReadKey();
      }
   }
}
```

What is different between String and System.String?

.NET defines all data types as a class. The System.String class represents a collection of Unicode characters also known as a text. The System.String class also defines the properties and methods to work with string data types.

The String class is equivalent to the System.String in C# language. The string class also inherits all the properties and methods of the System.String class.

Create a string

There are several ways to construct strings in C# and .NET.
Create a string using a constructor
Create a string from a literal
Create a string using concatenation
Create a string using a property or a method
Create a string using formatting
Create a string using its constructor

The String class has several overloaded constructors that take an array of characters or bytes. The following code snippet creates a string from an array of characters.

```
char[] chars = { 'M', 'a', 'h', 'e', 's', 'h' };
string name = new string(chars);
Console.WriteLine(name);
```

Create a string from a literal

This is the most common ways to instantiate a string.

You simply define a string type variable and assign a text value to the variable by placing the text value without double quotes. You can put almost any type of characters within double quotes accept some special character limitations.

The following code snippet defines a string variable named firstName and then assigns text value Mahesh to it.

```
string firstName;
firstName = "Mahesh";
```

Alternatively, we can assign the text value direct to the variable.

```
string firstName = "Mahesh";
```

Here is a complete sample example of how to create strings using literals.

```
using System;
namespace CSharpStrings
{
   class Program
   {
```

```
        static void Main(string[] args)
        {
            string firstName = "Mahesh";
            string lastName = "Chand";
            string age = "33";
            string numberString = "33.23";
            Console.WriteLine("First Name: {0}", firstName);
            Console.WriteLine("Last Name: {0}", lastName);
            Console.WriteLine("Age: {0}", age);
            Console.WriteLine("Number: {0}", numberString);
            Console.ReadKey();
        }
    }
}
```

Create a string using concatenation

Sting concatenation operator (+) can be used to combine more than one string to create a single string. The following code snippet creates two strings. The first string adds a text Date and current date value from the DateTime object. The second string adds three strings and some hard coded text to create a larger string.

```
string nowDateTime = "Date: " + DateTime.Now.ToString("D");
string firstName = "Mahesh";
string lastName = "Chand";
string age = "33";
string authorDetails = firstName + " " + lastName + " is " + age + " years old.";

Console.WriteLine(nowDateTime);
Console.WriteLine(authorDetails);
```

Create a string using a property or a method

Some properties and methods of the String class returns a string object such as SubString method. The following code snippet takes one sentence string and finds the age within that string. The code returns 33.

```
string authorInfo = "Mahesh Chand is 33 years old.";
int startPosition = sentence.IndexOf("is ") + 1;
string age = authorInfo.Substring(startPosition +2, 2 );
Console.WriteLine("Age: " + age);
```

Create a string with Format

The String.Format method returns a string. The following code snippet creates a new string using the Format method.

```
string name = "Mahesh Chand";
int age = 33;
string authorInfo = string.Format("{0} is {1} years old.", name, age.ToString());
Console.WriteLine(authorInfo);
```

Create a string using ToString Method

The ToString method returns a string. We can apply ToString on pretty much any data type that can be converted to a string. The following code snippet converts an int data type to a string.

```
string name = "Mahesh Chand";
int age = 33;
string authorInfo = string.Format("{0} is {1} years old.", name, age.ToString());
Console.WriteLine(authorInfo);
```

Get all characters of a string using C#

A string is a collection of characters.

The following code snippet reads all characters of a string and displays on the console.

```
string nameString = "Mahesh Chand";
for (int counter = 0; counter <= nameString.Length - 1; counter++)
Console.WriteLine(nameString[counter]);
```

Size of string

The Length property of the string class returns the number of characters in a string including white spaces.

The following code snippet returns the size of a string and displays on the console.

```
string nameString = "Mahesh Chand";
Console.WriteLine(nameString);
Console.WriteLine("Size of string {0}", nameString.Length);
```

Number of characters in a string

We can use the string.Length property to get the number of characters of a string but it will also count an empty character. So, to find out exact number of characters in a string, we need to remove the empty character occurrences from a string.

The following code snippet uses the Replace method to remove empty characters and then displays the non-empty characters of a string.

```
string name = "Mahesh Chand";

string name = "Mahesh Chand";

// Get size of string
Console.WriteLine("Size of string: {0}", name.Length );

// Remove all empty characters
string nameWithoutEmptyChar = name.Replace(" ", "");

// Size after empty characters are removed
Console.WriteLine("Size of non empty char string: {0}", nameWithoutEmptyChar.Length);
```

```
// Read and print all characters
for (int counter = 0; counter <= nameWithoutEmptyChar.Length - 1; counter++)
Console.WriteLine(nameWithoutEmptyChar[counter]);
```

Convert String to Char Array

ToCharArray method converts a string to an array of Unicode characters. The following code snippet converts a string to char array and displays them.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
char[] charArr = sentence.ToCharArray();
foreach (char ch in charArr)
{
  Console.WriteLine(ch);
}
```

Empty String

An empty string is a valid instance of a System.String object that contains zero characters. There are two ways to create an empty string. We can either use the string.Empty property or we can simply assign a text value with no text in it.

The following code snippet creates two empty strings.

```
string empStr = string.Empty;
string empStr2 = "";
```

Both of the statements above generates the same output.

An empty string is sometimes used to compare the value of other strings. The following code snippet uses an empty string to compare with the name string.

```
string name = "Mahesh Chand";
if (name != empStr)
{
  Console.WriteLine(name);
}
```

In real world coding, we will probably never create an empty string unless you plan to use it somewhere else as a non-empty string. We can simply use the string.Empty direct to compare a string with an empty string.

```
if (name != string.Empty)
{
  Console.WriteLine(name);
}
```

Here is a complete example of using an empty string.

```
string empStr = string.Empty;
string empStr2 = "";
string name = "Mahesh Chand";
if (name != empStr)
{
```

```
  Console.WriteLine(name);
}
if (name != string.Empty)
{
  Console.WriteLine(name);
}
```

C# STRING FUNCTION

String FunctionsDefinitions

Clone() Make clone of string.

CompareTo() Compare two strings and returns integer value as output. It returns 0 for true and 1 for false.

Contains() The C# Contains method checks whether specified character or string is exists or not in the string value.

EndsWith() This EndsWith Method checks whether specified character is the last character of string or not.

Equals()The Equals Method in C# compares two string and returns Boolean value as output.

GetHashCode() This method returns HashValue of specified string.

GetType() It returns the System.Type of current instance.

GetTypeCode() It returns the Stystem.TypeCode for class System.String.

IndexOf() Returns the index position of first occurrence of specified character.

ToLower() Converts String into lower case based on rules of the current culture.

ToUpper() Converts String into Upper case based on rules of the current culture.

Insert() Insert the string or character in the string at the specified position.

IsNormalized() This method checks whether this string is in Unicode normalization form C.

LastIndexOf() Returns the index position of last occurrence of specified character.

Length It is a string property that returns length of string.

Remove() This method deletes all the characters from beginning to specified index position.

Replace() This method replaces the character.

Split() This method splits the string based on specified value.

StartsWith() It checks whether the first character of string is same as specified character.

Substring() This method returns substring.

ToCharArray() Converts string into char array.

Trim() It removes extra whitespaces from beginning and ending of string.

.....

Write a program in C# Sharp to find maximum and minimum element in an array.....

C# Sharp Code:

```
using System;
public class Exercise9
{
   public static void Main()
{
   int[] arr1= new int[100];
   int i, mx, mn, n;


     Console.Write("\n\nFind maximum and minimum element in an array :\n");
     Console.Write("-------------------------------------------------\n");

     Console.Write("Input the number of elements to be stored in the array :");
            n= Convert.ToInt32(Console.ReadLine());

     Console.Write("Input {0} elements in the array :\n",n);
     for(i=0;i<n;i++)
      {
               Console.Write("element - {0} : ",i);
                      arr1[i] = Convert.ToInt32(Console.ReadLine());
             }


   mx = arr1[0];
   mn = arr1[0];

   for(i=1; i<n; i++)
   {
      if(arr1[i]>mx)
      {
         mx = arr1[i];
      }


      if(arr1[i]<mn)
      {
         mn = arr1[i];
      }
   }
   Console.Write("Maximum element is : {0}\n", mx);
   Console.Write("Minimum element is : {0}\n\n", mn);
 }
}
```

Copy

Sample Output:

Find maximum and minimum element in an array :

Input the number of elements to be stored in the array :2
Input 2 elements in the array :
element - 0 : 20
element - 1 : 25
Maximum element is : 25
Minimum element is : 20
.....
Write a programin C# Sharp to separate odd and even integers in separate arrays

```
using System;

public class Program
{
   public static void Main()
   {
      int[] values ={ 1, 22, 13, 44, 52, 66, 71, 88, 99, 100 };

      foreach (var result in values)
      {
         if (result % 2 == 0)
         {
            Console.WriteLine(result + " is Even Value");
         }
         else
         {
            Console.WriteLine(result + " is Odd Value");
         }
      }
   }
}
```

Output

1 is Odd Value
22 is Even Value
13 is Odd Value
44 is Even Value
52 is Even Value
66 is Even Value
71 is Odd Value
88 is Even Value
99 is Odd Value

100 is Even Value

.....

Write a program in C# Sharp to sort elements of array in ascending order.

```
using System;
class Program
{
   static void Main()
   {
      int i;
      int[] a = new int[30]; // Array Declaration in C#
      Console.Write("Enter the Number of values to be Sort : ");
      // read the string value (by default) and convert it in to integer
      int n = Convert.ToInt16(Console.ReadLine());
      //Reading the values one by one
      for (i = 1; i <= n; i++)
      {
         Console.Write("Enter the No " + i + ":");
         // read the string value (by default) and convert it in to integer
         a[i] = Convert.ToInt16(Console.ReadLine());
      }
      //Sorting the values
      for (i = 1; i <= n; i++)
      {
         for (int j = 1; j <= n - 1; j++)
         {
            if (a[j] > a[j + 1])
            {
               int temp = a[j];
               a[j] = a[j + 1];
               a[j + 1] = temp;
            }
         }
      }
      //Display the Ascending values one by one
      Console.Write("Ascending Sort : ");
      for (i = 1; i <= n; i++)
      {
         Console.Write(a[i]+" ");
      }
      //Waiting for output
      Console.ReadKey();
   }
}
```

.....

Write a program in C# Sharp for addition of two Matrices of same size

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

namespace Add_Two_Matrix
{
   class Program
   {
      static void Main(string[] args)
      {
         int[,] a = {{1, 2, 3 }, { 4, 5, 6 }, { 7,8,9} };
         int[,] b ={ { 4, 8, 7 }, { 6,5,4}, {3,2,1 } };
         int[,] c = new int[3,3];
         int f = c.Length;
         int i, m = 0;
         int j = 0;
         int n = 0;
         for ( i = 0;i<3;i++)
         {
            Console.WriteLine(" ");

            for (j = 0; j < 3; j++)
            {
               Console.Write(" " + a[i, j]);
            }

         }
         Console.Write("\n");
         for ( m = 0; m < 3; m++)
         {
            Console.WriteLine(" ");

            for ( n = 0; n < 3; n++)
            {
               Console.Write(" " + b[m, n]);
            }

         }
         Console.Write("\n");
         for (int k = 0; k < 3; k++)
         {
            Console.WriteLine("");
            for (int l = 0; l < 3; l++)
            {
```

```
            Console.Write(a[k, l] + b[k, l] + "\t");
           }
         }
         Console.WriteLine("Yup Its Solved Its very easy welcome your all time."+"\n"+"RAK Groups");
           Console.ReadLine();
      }
   }
}
```

.....

Write a program in C# Sharp for threading.....

```
using System.Threading;

namespace ThreadTest
{
    class Program
   {
       static voidMain(string[] args)
       {
          //Creating thread object to strat it
          Thread th= newThread(ThreadB);
          Console.WriteLine("Threads started :");
          // Start thread B
         th.Start();
          //Thread A executes 10 times
          for (inti=0; i<10; i++)
         {
             Console.WriteLine("Thread : A");
         }

          Console.WriteLine("Threads completed");
          Console.ReadKey();
       }
       public staticvoidThreadB()
       {
         //Executes thread B 10 times
        for(inti=0;i<10;i++)
       {
          Console.WriteLine("Thread : B");
       }
       }
   }
}
```

.....thread sunchronization.....

Synchronization is handled with the following four categories:

The following are the four categories to handle Synchronization mechanism:
Blocking Methods
Locking Construct
Signaling
No Blocking Synchronization
Blocking Methods

In this technique, one thread wait for the another thread to finish for a particular period of time. Here, the execution is paused for some reason. The following are some Blocking methods in C#:
Sleep
Join
Task.Wait
Sleep

It pauses the execution of a program for a defined time. It does not utilize CPU during the pause time. It is useful for waiting on an external Task.
Thread.Sleep(300)
Join

It is also a part of blocking mechanism in the thread synchronization. It is similar to Sleep but it does not pauses all threads. It pauses the calling thread until the thread whose join method is called has completed.

Example:

```
class Program
{
   static void Main(string[] args)
   {
      Thread thread1 = new Thread(Method1);
      thread1.Start();

      Thread thread2 = new Thread(Method2);
      thread2.Start();

      thread1.Join();
```

```
        Console.WriteLine("After Thread1");

        thread2.Join();
        Console.WriteLine("After Thread2");
        Console.ReadKey();

    }

    private static void Method2(object obj)
    {
        Console.WriteLine("Thread1 Executed.");
    }

    private static void Method1(object obj)
    {
        Console.WriteLine("Thread2 Executed");
    }


}
```

Output:

```
Thread2 Executed.
Thread1 Executed.
After Thread1
After Thread2
```

Task.Wait

It is a blocking Synchronization method which allows the calling thread to wait until the current task has been completed.

Example:

```
class Program
{
    static void Main(string[] args)
    {
        Task task = Task.Run(() =>
        {
            Random randomNumbers = new Random();
            long sum = 0;
            int count = 1000000;
            for (int i = 1; i <= count; i++)
            {
                int randomNumber = randomNumbers.Next(0, 101);
```

```
            sum += randomNumber;
        }

        Console.WriteLine("Total:{0}", sum);
        Console.WriteLine("Count: {0}", count);
    });
    task.Wait();

    Console.ReadKey();
  }
}
```

Output:

```
Total: 50028434
Count: 1000000
```

Locking

Locking is also a synchronization mechanism. It limits the access to a resource in multiple thread. Exclusive locking mechanism is used for it.

The following are the two main Locking mechanism:
Lock
Mutex
Lock

It locks the critical section of code so that only one thread can execute the critical section of code at a time. If another thread tries to enter into critical section of code then it is prevented and blocked and then it will wait until the object is released from the using thread.

Example:

```
 class Program
 {
    decimal totalBalance =50000;
    private Object myLock = new Object();

    static void Main(string[] args)
    {
       Program program = new Program();
       program.WithDraw(5000);
       Console.ReadKey();
    }

    public  void WithDraw(decimal amount)
    {
```

```
        lock (myLock)
        {
            if (amount > totalBalance)
            {
                Console.WriteLine("Insufficient Amount.");
            }

            totalBalance -= amount;
            Console.WriteLine("Total Balance {0}",totalBalance);
        }
    }

 }
```

Output:

Total Balance 45000

Note: Avoid lock on a public type, or instances beyond the control of code.