Programming Strings using C#

Mahesh Chand

June 2012, Garnet Valley PA

Strings in C#

In any programming language, to represent a value, we need a data type. The Char data type represents a character in .NET. In .NET, the text is stored as a sequential read-only collection of Char objects. There is no null-terminating character at the end of a C# string; therefore a C# string can contain any number of embedded null characters ('\0').

The System.String data type is used to represent a string. A string in C# is an object of type System.String.

The string class in C# represents a string.

The following code creates three strings with a name, number and double values.

```
// 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";
```

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();
        }
    }
}
```

String Class

The string class defined in the .NET base class library represents text as a series of Unicode characters. The string class provides methods and properties to work with strings.

The string class has methods to clone a string, compare strings, concatenate strings, and copy strings. This class also provides methods to find a substring in a string, find the index of a character or substring, replace characters, spilt a string, trim a string, and add padding to a string. The string class also provides methods to convert a string characters to uppercase or lowercase.

Check out these links to learn about a specific operation or functionality of strings.

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.

1. Create a string using a constructor
2. Create a string from a literal
3. Create a string using concatenation
4. Create a string using a property or a method
5. 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 using a formatting

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";

// 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.

```
Console.WriteLine("Start" + empStr + "End");
Console.WriteLine("Start" + empStr2 + "End");
```

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);
}
```

Null String

A null string is a string variable that has not been initialized yet and has a null value. If you try to call any methods or properties of a null string, you will get an exception. A null string valuable is exactly same as any other variable defined in your code.

A null string is typically used in string concatenation and comparison operations with other strings.

The following code example shows how to use a null string.

```
string nullStr = null;
string empStr = string.Empty;

string name = "Mahesh Chand";

if ((name != nullStr) || (name != empStr))
```

```
{
    Console.WriteLine(name + " is neither null nor empty");
}
```

What is the difference between a null and an 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.

```
Console.WriteLine("Start" + empStr + "End");
Console.WriteLine("Start" + empStr2 + "End");
```

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);
}
```

A null string is a string variable that has not been initialized yet and has a null value. If you try to call any methods or properties of a null string, you will get an exception. A null string valuable is exactly same as any other variable defined in your code.

A null string is typically used in string concatenation and comparison operations with other strings.

The following code example shows how to use a null string.

```
string nullStr = null;
string empStr = string.Empty;

string name = "Mahesh Chand";

if ((name != nullStr) || (name != empStr))
```

```
{
    Console.WriteLine(name + " is neither null nor empty");
}
```

Immutable String

Strings in .NET are immutable. Once a string is created, it cannot be modified. It may appear to you that you are reassigning a new value to the same string object but that's not true. Every time you assign a new value to an existing string object, a new object is being created and old object is being released by the CLR. That means you must be careful when dealing with string operations in C# and .NET. If you are doing lot of string operations, the StringBuilder class is recommended.

The following code snippet defines two string instances and then we change the value of the first string with the value of the second string. You would think, you are still using same old name variable but actually the name in the third line is a new object.

```
string name = "Mahesh Chand";
string lastName = "Something else";
name = lastName;
Console.WriteLine(name);
```

Clone a string

The Clone method of string class is used to clone a string. The Clone method returns the reference of the same string. The following code snippet clones a string.

```
string firstName = "Mahesh";
string lastName = "Chand";
string authorName = firstName + " " + lastName ;

string cloneAuthor = authorName.Clone().ToString();

Console.WriteLine(authorName);
Console.WriteLine(cloneAuthor);
```

Compare strings

The string class has three compare methods with several overloaded forms – Compare, CompareOrdinal, and CompareTo.

The Compare method compares two strings and returns an integer value that indicates their relative position in the sort order. The return value of Compare method can be less than zero, greater than zero or equals to zero depending on if one string comes before or after in the sort order. The Compare method has 8 overloaded forms. It is recommended to use the overloaded method with StringComparison. Check out best practices for using strings in .NET.

The following code compares two strings and return results on the System console.

```
string name = "Mahesh Chand";
string name2 = "Mojesh Chand";
int comp = string.Compare(name, name2);
if (comp > 0)
{
    Console.WriteLine(name +" comes after " + name2);
}
else if (comp < 0)
{
    Console.WriteLine(name + " comes before " + name2);
}
else
{
    Console.WriteLine(name + " and  " + name2 + " are same." );
}
```

The CompareOrdinal method compares two strings by evaluating the numeric values of the corresponding character in each string.

The following code compares two strings using the CompareOrdinal method and return results on the System console.

```
String name1 = "MAHESH";
String name2 = "mahesh";
String str;
int result;
result = String.CompareOrdinal(name1, name2);
str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" : "equal to"));
Console.Write("String '{0}' is ", name1);
Console.Write("{0} ", str);
Console.WriteLine("String '{0}'.", name2);
```

The CompareTo method is an instance method. It compares a value (either a string or on object) with a string instance. Return values of this method are same as the Compare method. The following source code compares two strings.

The following code compares two strings using the CompareTo method and return results on the System console.

```
String name1 = "MAHESH";
String name2 = "mahesh";
String str;
int result;
result = name1.CompareTo(name2);
str = ((result < 0) ? "less than" : ((result > 0) ? "greater than" : "equal to"));
Console.Write("String '{0}' is ", name1);
Console.Write("{0} ", str);
Console.WriteLine("String '{0}'.", name2);
```

Copy a string

The string class provides two methods to copy strings – Copy and CopyTo.

The Copy method copies the contents of a string to another. The Copy method is a static method and takes a string as input and returns another string with the same contents as the input string. For example, the following code copies authorName string to copyAuthor string.

```
string firstName = "Mahesh";
string lastName = "Chand";
string authorName = firstName + " " + lastName;
Console.WriteLine("Original String: {0}", authorName);
string copyAuthor = string.Copy(authorName);
Console.WriteLine("Copied String: {0}", copyAuthor);
```

The CopyTo method copies a specified number of characters from a specified position in this instance to a specified position in an array of characters. For example, the following example copies contents of str1 to an array of characters. You can also specify the starting character of a string and the number of characters you want to copy to the array.

```
string firstName = "Mahesh";
string lastName = "Chand";
string authorName = firstName + " " + lastName;
char[] copiedString = { 'n', 'e', 'e', 'l', ' ', 'b', 'e', 'n', 'i', 'w', 'l' };

Console.WriteLine("Original String: {0}", authorName);
authorName.CopyTo(2, copiedString, 2, 6);
Console.WriteLine(copiedString);
```

Concatenate strings

The Concat method of the string class concatenates one or more strings and objects and a combination of all.

The following code snippet shows different forms of sting and object concatenation.

```
int number = -123;
bool b = false;
Object numberObject = number;
object boolObject = b;
string author = "Mahesh Chand";
Object[] objs = new Object[] { 555, 444, 999 };

Console.WriteLine("Concatenate 1: {0}", String.Concat(numberObject));
Console.WriteLine("Concatenate multiple: {0}", String.Concat(numberObject, boolObject, author));
Console.WriteLine("Concatenate an array: {0}", String.Concat(objs));
```

Ends With

The EndsWith method returns true if a strings ends with a specified string. The following code snippet checks if a string ends with "C# Corner".

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
string findString = "C# Corner";
bool b = sentence.EndsWith(findString);
Console.WriteLine("String ends with {0}: {1}", findString, b);
```

Starts With

The StartsWith method returns true if a strings starts with a specified string. The following code snippet checks if a string starts with "Mahesh".

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
string findString = "Mahesh";
bool b = sentence.StartsWith(findString);
Console.WriteLine("String ends with {0}: {1}", findString, b);
```

Format a C# String

The string.Format method is used to create formatted strings and concatenate multiple strings representing multiple objects. The Format method automatically converts any passed object into a string.

For example, the following code uses an integer, a floating number and string values and formats them into a string using the Format method.

```
int val = 7;
string name = "Mr. John";
float num = 45.06f;
string str = String.Format("Days Left : {0}. Current DataTime: {1:u}. \n String: {2}, Float: {3}", val, DateTime.Now, name,
num);
Console.WriteLine(str);
```

Learn more about string formatting - Composite Formatting in .NET

Find number of characters in a string

The Length property of the String class returns the number of characters within a string. The following code snippet in returns the number of characters in a string.

```
string authorName = "Mahesh Chand Beniwal";
Console.WriteLine("Number of characters: " + authorName.Length);
```

Read all characters in a string

The Chars property of a string represents all characters in a string. This property is an indexer that means we do not need to specify the property explicitly.

The following code snippet in loops through all characters of a string and displays on the console.

```
String authorName = "Mahesh Chand Beniwal";
for (int i = 0; i < authorName.Length; i++)
        Console.WriteLine(authorName[i]);
```

Find a character in a string

The IndexOf and LastIndexOf methods can be used to find an index of a character within a string. The IndexOf method returns the 0 based index of the first occurrence of a character. If a character is found, it returns the index of the character; otherwise returns -1;

This method has several overloaded forms and hence can be applied on the entire string or a portion of the string by specifying the start and end positions of characters in the string.

The LastIndexOf method returns the 0 based index of the last occurrence of a character found in a string.

The following code snippet uses various forms of IndexOf and LastIndexOf methods. As you can see from this code, we can even specify what part of a string you want to search for. The second parameter of IndexOf and LastIndexOf takes the starting index and third parameter is the number of characters after the starting index.

```
String authorName = "Mahesh Chand Beniwal";
Console.WriteLine("Index of first 'a': " + authorName.IndexOf('a'));
Console.WriteLine("Index of 'a' after 5th char: " + authorName.IndexOf('a', 5));
Console.WriteLine("Index of 'a' after 10th char till next 5: " + authorName.IndexOf('a', 10, 5));

Console.WriteLine("Last index of 'a': " + authorName.LastIndexOf('a'));
Console.WriteLine("Last index of 'a': " + authorName.LastIndexOf('a', 5));
Console.WriteLine("Last index of 'a': " + authorName.LastIndexOf('a', 10, 5));
```

Find any character in a string

The IndexOfAny and LastIndexOfAny methods can be used to find an index of any character of an array of characters within a string. The IndexOfAny method returns the 0 based index of the first occurrence of a character. If a character is found, it returns the index of the character; otherwise returns -1.

This method has several overloaded forms and hence can be applied on the entire string or a portion of the string by specifying the start and end positions of characters within a string.

The LastIndexOfAny method returns the 0 based index of the last occurrence of a character found in a string.

The following code snippet uses various forms of IndexOfAny and LastIndexOfAny methods. As you can see from this code, we can even specify what part of a string you want to search for. The second parameter of IndexOf and LastIndexOf takes the starting index and third parameter is the number of characters after the starting index.

```
String authorName = "Mahesh Chand Beniwal";

char[] findChars = new char[] { 'a', 'e', 'i', 'o', 'u'};
Console.WriteLine(authorName.IndexOfAny(findChars));
Console.WriteLine(authorName.IndexOfAny(findChars, 5));
Console.WriteLine(authorName.IndexOfAny(findChars, 5, 5));

Console.WriteLine(authorName.LastIndexOfAny(findChars));
Console.WriteLine(authorName.LastIndexOfAny(findChars, 5));
Console.WriteLine(authorName.LastIndexOfAny(findChars, 5, 5));
```

Removing String in C#

The Remove method deletes a specified number of characters from a string at a specified position. This method returns the result as a string.

For example, the following code removes 13 characters from index starting at 16th position.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
sentence = sentence.Remove(16, 13);
Console.WriteLine("Final String");
Console.WriteLine(sentence);
```

Replace String in C#

The Replace method replaces all occurrences of a specified character in a string. For example, the following source code replaces all occurrences of string "author" with string "writer" in the string.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
sentence = sentence.Replace("author", "writer");
Console.WriteLine("Final String");
Console.WriteLine(sentence);
```

Split String in C#

The Split method separates strings by a specified set of characters and places these strings into an array of strings. The Split method takes an array of separator strings. For example, if you want to separate a string with a space, you would pass space as a separator. The separated strings are returned in an array of strings.

The following code snippet splits sentence string based on spaces and stores all separated strings in an array.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
string[] stringSeparators = new string[] { " " };
string[] result = sentence.Split(stringSeparators, StringSplitOptions.None);
Console.WriteLine("Final Strings:");
foreach (string s in result)
{
    Console.Write("'{0}' ", String.IsNullOrEmpty(s) ? "<>" : s);
}
```

Uppercase a string in C#

The ToUpper method of a string converts a string to uppercase . The following code shows how to create an uppercase string.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
Console.WriteLine("Uppercase: {0}", sentence.ToUpper());
Console.WriteLine("Lowercase: {0}", sentence.ToLower());
```

Lowercase a string in C#

The ToLower method of a string converts a string to lowercase . The following code shows how to create an lowercase string.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
Console.WriteLine("Uppercase: {0}", sentence.ToUpper());
Console.WriteLine("Lowercase: {0}", sentence.ToLower());
```

Padding C# String

The string class provides PadLeft and PadRight methods to pad a string.

Left padding a string in C#

The PadLeft method of the string class is used apply left padding on a string. The following code snippet uses PadLeft method to pad a string.

```
string name = "Mahesh Chand";
char pad = '*';
Console.WriteLine(name.PadLeft(15, pad));
Console.WriteLine(name.PadLeft (2));
```

Right padding a string in C#

The PadRight method of the string class is used apply left padding on a string. The following code snippet uses PadRight method to pad a string.

```
string name = "Mahesh Chand";
char pad = '*';
Console.WriteLine(name.PadLeft(15, pad));
Console.WriteLine(name.PadRight(2, pad));
```

Trim C# String

Trimming a string is removing spaces and/or special characters from a string. A sting can be trimmed from the left side, the right side and also both sides.

The TrimStart method is used to trim a string from the starting and TrimEnd method is used to trip as string from the end.

Here is a complete example of

```
Console.WriteLine("TrimString");

string firstName = "Mahesh";
string lastName = "Chand";
string age = "33";
string authorInfo = " Author "+ firstName + " " + lastName + " is "
    + age.ToString() + " years old.  ";
Console.WriteLine("Original string:{0}", authorInfo);
Console.WriteLine("Lengh:{0}", authorInfo.Length);

string trimmedString = authorInfo.Trim();
Console.WriteLine("Trimmed string:{0}", trimmedString);
Console.WriteLine("Lengh:{0}", trimmedString.Length);

// Trim End
string trimmedLeftString = authorInfo.TrimStart();
Console.WriteLine("Trimmed left string:{0}", trimmedLeftString);
Console.WriteLine("Lengh:{0}", trimmedLeftString.Length);

// Trim Start
string trimmedRightString = authorInfo.TrimEnd();
Console.WriteLine("Trimmed right string:{0}", trimmedRightString);
Console.WriteLine("Lengh:{0}", trimmedRightString.Length);

char[] charsToTrim = { '*', ' '};
string charString = " ** Author " + firstName + " "
    + lastName + " is " + age.ToString() + " years old. *** ";
string charTrimmed = charString.Trim(new char[] {'*', ' '});
Console.WriteLine("Char Trimmed string:{0}", charTrimmed);
Console.WriteLine("Lengh:{0}", charTrimmed.Length);
```

Trim a string in C#

The Trim method trims a string from both end ends by removing white spaces from the start and the end of a string.

The following code snippet shows how to trim a string.

```
string firstName = "Mahesh";
string lastName = "Chand";
string age = "33";
string authorInfo = " Author "+ firstName + " " + lastName + " is "
    + age.ToString() + " years old.  ";
Console.WriteLine("Original string:{0}", authorInfo);
Console.WriteLine("Lengh:{0}", authorInfo.Length);

string trimmedString = authorInfo.Trim();
Console.WriteLine("Trimmed string:{0}", trimmedString);
Console.WriteLine("Lengh:{0}", trimmedString.Length);
```

Trim left string in C#

The TrimStart method trims a string from the begging by removing white spaces from the start of a string.

The following code snippet shows how to trim a string from the start.

```
string firstName = "Mahesh";
string lastName = "Chand";
string age = "33";
string authorInfo = " Author "+ firstName + " " + lastName + " is "
    + age.ToString() + " years old.  ";
Console.WriteLine("Original string:{0}", authorInfo);
Console.WriteLine("Lengh:{0}", authorInfo.Length);

// Trim Start
string trimmedRightString = authorInfo.TrimEnd();
Console.WriteLine("Trimmed right string:{0}", trimmedRightString);
Console.WriteLine("Lengh:{0}", trimmedRightString.Length);
```

Trim right string in C#

The TrimEnd method trims a string from the end by removing white spaces from the end of a string.

The following code snippet shows how to trim a string from the end.

```
string firstName = "Mahesh";
string lastName = "Chand";
string age = "33";
string authorInfo = " Author "+ firstName + " " + lastName + " is "
    + age.ToString() + " years old.  ";
Console.WriteLine("Original string:{0}", authorInfo);
Console.WriteLine("Lengh:{0}", authorInfo.Length);

string trimmedString = authorInfo.Trim();
Console.WriteLine("Trimmed string:{0}", trimmedString);
Console.WriteLine("Lengh:{0}", trimmedString.Length);

// Trim End
string trimmedLeftString = authorInfo.TrimStart();
Console.WriteLine("Trimmed left string:{0}", trimmedLeftString);
Console.WriteLine("Lengh:{0}", trimmedLeftString.Length);
```

C# Substrings

A substring is a part of a string. The Substring method retrieves a substring from a string starting at a specified position. The Substring method has two overloaded forms. The first form takes just an integer as the starting position of the substring and returns rest of the string. The second form takes first integer as the starting position and second integer as the length of the substring.

The following code snippet gets two substrings from a string.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner";
Console.WriteLine("Original String: {0}", sentence);
Console.WriteLine("Substring starting at 20: {0}", sentence.Substring(20));
Console.WriteLine("Substring starting at 20, next 10 chars: {0}", sentence.Substring(20, 10));
```

Sub String Exists

The Contains method returns true if a substring is a part a string. The following code snippet checks if a substring exists in a string.

```
string sentence = "Mahesh Chand is an author and founder of C# Corner.";
Console.WriteLine("Original String: {0}", sentence);
string findString = "author";
bool b = sentence.Contains(findString);
Console.WriteLine("String exists: {0}", b);
```