1.6 Manipulating String

String

String is an object of **System.String** class that represent sequence of characters enclosed with double quotes.

We can perform many operations on strings such as concatenation, comparision, getting substring, search, trim, replacement etc.

string vs String

In C#, *string* is keyword which is an alias for *System.String* class. That is why string and String are equivalent. We are free to use any naming convention.

```
string s1 = "hello";          //creating string using string keyword
String s2 = "welcome";        //creating string using String class
```

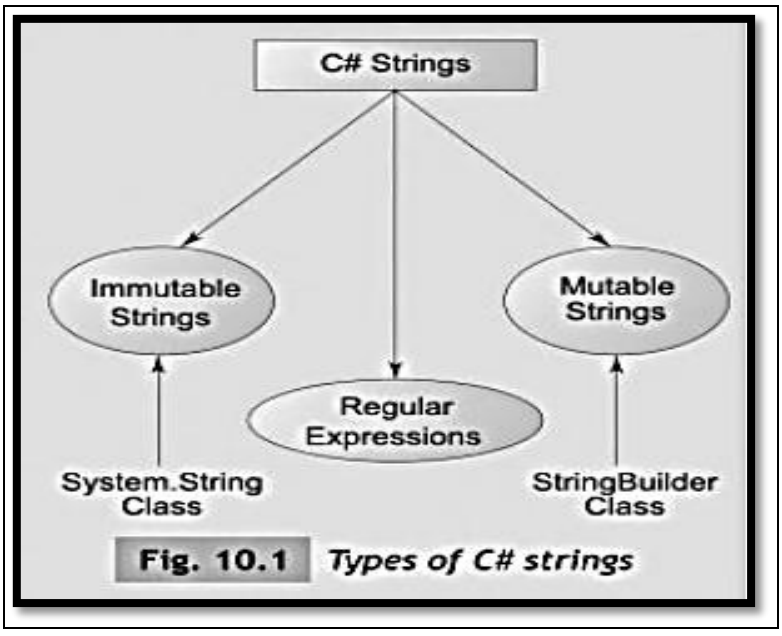

Fig. 10.1 *Types of C# strings*

Immutable string

String objects are immutable, meaning that cannot modify the characters contained in them.

Mutable string

Mutable string that are modifiable can be created using StringBuilder class.

Assigning String literals

The most common way to create a string is to assign a quoted string of characters known as *string literal* to a string object. For example:

```
string s1;----//declaring a string object
s1 = "abc";--  //assigning string literal
```

Both these statements may be combined into one as follows:

```
string s1 = "abc"; //declaring and assigning
```

Copying Strings

We can also create new copies of existing strings. This can be accomplished in two ways:

- Using the overloaded = operator
- Using the static **Copy** method

Example:

```
string s2 = s1;                  //assigning
string s2 = string.Copy(s1);     //copying
```

Both these statements would accomplish the same thing, namely, copying the contents of **s1** into **s2.**

Concatenating Strings

We may also create new strings by concatenating existing strings. There are a couple of ways to accomplish this.

- Using the overloaded + operator
- Using the static **Concat** method

Examples:

```
string s3 = s1 + s2; //s1 and s2 exist already
string s3 = string.Concat(s1, s2)
```

If s1 = 'abc' and s2 = 'xyz', then both the statements will store the string 'abcxyz' in s3. Note that the

Reading from the Keyboard

It is possible to read a string value interactively from the keyboard and assign it to a string object.

```
string s = Console.ReadLine( );
```

ToString Method

Another way of creating a string is to call the **ToString** method on an object and assign the result to a string variable.

```
int number = 123;
string numStr = number.ToString( );
```

This statement converts the number 123 to a string “123” and then assigns the string value to the string variable **numStr**

Verbatim String

Strings can also be created using what are known as *verbatim strings*. Verbatim strings are those that start with the @ symbol. This symbol tells the compiler that the string should be used verbatim even if it includes escape characters.

```
string s1 = @ "\EBG\Csharp\string.cs";
```

In order to obtain the same output without using the symbol @, the input string should be written as follows:

```
string s1 = "\\EBG\\Csharp\\string.cs";
```

We may also use the escape characters such as \n and \t in @ - quoted strings.

String Methods

String objects are ***immutable***, meaning that we cannot modify the characters contained in them. However, since the string is an alias for the predefined **System.String** class in the Common Language Runtime (CLR), there are many built-in operations available that work with strings. All operations produce a modified version of the string rather than modifying the string on which the method is called.

Method	*Operation*
Compare ( )	Compares two strings
CompareTo ( )	Compares the current instance with another instance
ConCat ( )	Concatenates two or more strings
Copy( )	Creates a new string by copying another
CopyTo ( )	Copies a specified number of characters to an array of Unicode characters
EndsWith ( )	Determines whether a substring exists at the end of the string
Equals( )	Determines if two strings are equal
IndexOf ( )	Returns the position of the first occurrence of a substring
Insert ( )	Returns a new string with a substring inserted at a specified location
Remove ( )	Deletes characters from the string
Replace ( )	Replaces all instances of a character with a new character
Split ( )	Creates an array of strings by splitting the string at any occurrence of one
StartsWith ( )	Determines whether a substring exists at the beginning of the string
Substring ( )	Extracts a substring
ToLower ( )	Returns a lower-case version of the string
ToUpper ( )	Returns an upper-case version of the string

Insert() Method

In C#, ***Insert()*** method is a String method. It is used to return a new string in which a specified string is inserted at a specified index position in the current string instance.

```
using System.IO;
using System;

class Program
{
   static void Main()
   {
      string s1="Lean";
      string s2=s1.Insert(3,"r");
      string s3=s2.Insert(5,"er");
      Console.WriteLine(s2);
     Console.WriteLine(s3);
   }
}
```

Output

```
Learn
Learner
```

Comparing Strings

String class supports overloaded methods and operators to compare whether two strings are equal or not. They are:

- Overloaded **Compare( )** method
- Overloaded **Equals( )** method
- Overloaded **==** operator

Compare() method

C# Compare() method is used to compare first string with second string lexicographically. It returns an integer value.

If both strings are equal, it returns 0. If first string is greater than second string, it returns 1 else it returns -1.

```
s1==s2 returns 0
s1>s2 returns 1
s1<s2 returns -1
```

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "hello";
        string s2 = "hello";
        string s3 = "csharp";
        string s4 = "mello";
       Console.WriteLine(string.Compare(s1,s2));
        Console.WriteLine(string.Compare(s2,s3));
        Console.WriteLine(string.Compare(s3,s4));
      }
   }
```

Output:

```
0
1
-1
```

Equal() method

The **string** class supports an overloaded **Equals** method for testing the equality of strings. There are again two versions of **Equals** method. They are implemented as follows:

bool b1 = s2.Equals(s1);

bool b2 = string.Equals (s2, s1);

These methods return a Boolean value **true** if s1 and s2 are equal, otherwise **false.**

```
using System;
 class Program
{
 public static void Main(string[] args)
    {
       string s1 = "ProGeek 2.0";
       string s2 = "ProGeek 2.0";
        Console.WriteLine(s1.Equals(s2));
    }
}
```

Output:

```
True
```

The = = Operator

A simple and natural way of testing the equality of strings is by using the overloaded = = operator.

Example:

```
bool b3 = (s1 == s2); //b3 is true if they are equal
```

We very often use such statements in decision statements, like:

```
if (s1 == s2)
    Console.WriteLine("They are qual");
```

Finding Substring

It is possible to extract substrings from a given string using the overloaded **Substring** method available in **String** class. There are two version of **Substring:**

- s.Substring(n)
- s.Substring(n1, n2)

Examples:

```
string s1 = "NEW YORK";
string s2 = s1.Substring(5);
string s3 = s1.Substring(0,3);
string s4 = s1.Substring(5,8);
```

When executed, the string variables will contain the following substrings:

```
s2:     YORK
s3:     NEW
s4:     YORK
```

StartsWith() Method

In C#, **StartsWith()** is a string method. This method is used to check whether the beginning of the current string instance matches with a specified string or not.

```
using System.IO;
using System;

class Program
{
   static void Main()
   {
      string s1="Hello";
       bool x=s1.StartsWith("He");

       Console.WriteLine(x);
   }
}
```

Output

```
True
```

EndsWith() Method

In C#, **EndsWith()** is a string method. This method is used to check whether the ending of the current string instance matches with a specified string or not.

```
using System.IO;
using System;

class Program
{
   static void Main()
   {
      string s1="Hello";
       bool x=s1.EndsWith("lo");

       Console.WriteLine(x);
   }
}
```

Output

```
True
```

ToUpper() Method

In C#, ***ToUpper()*** is a string method. It converts every character to Uppercase

```
using System.IO;
using System;

class Program
{
   static void Main()
   {
      string s1="Hello";
       string s2=s1.ToUpper();

       Console.WriteLine(s2);
   }
}
```

Output

```
HELLO
```

ToLower() Method

In C#, ***ToLower()*** is a string method. It converts every character to lowercase

```
using System.IO;
using System;

class Program
{
   static void Main()
   {
      string s1="Hello";
       string s2=s1.ToLower();

       Console.WriteLine(s2);
   }
}
```

Output

```
hello
```

Mutable string

Mutable strings that are modifiable can be created using the **StringBuilder** class. *Examples:*

```
StringBuilder str1 = new StringBuilder("abc");
StringBuilder str2 = new StringBuilder ( );
```

- The string object str1 is created with the initial size of three characters and str2 is created as an empty.
- They can grow dynamically as more characters are added to them.
- They can grow either unbounded or up to a configurable maximum.
- Mutable strings are also called as dynamic string.

The **StringBuilder** class supports many methods that are useful for manipulating dynamic strings.

Table 10.2 *Some useful stringBuilder methods*

METHOD	OPERATION
Append ( )	Appends a string
AppendFormat ( )	Appends strings using a specific format
EnsureCapacity ( )	Ensures sufficient size
Insert ( )	Inserts a string at a specified position
Remove ( )	Removes the specified characters
Replace ( )	Replaces all instances of a character with a specified one

C# also supports some special functions known as *properties*. They are listed in Table 10.3

Table 10.3 *StringBuilder properties*

PROPERTY	PURPOSE
Capacity	To retrieve or set the number of characters the object can hold
Length	To retrieve or set the length
MaxCapacity	To retrieve the maximum capacity of the object
[ ]	To get or set a character at a specified position

```
using System;
using System.Text;

public class HelloWorld
{
   public static void Main(string[] args)
   {
       StringBuilder s1= new StringBuilder("Object ");
       Console.WriteLine(s1);
      Console.WriteLine(s1.Length);

      s1.Append("language");
      Console.WriteLine(s1);

       s1.Insert(7,"oriented ");
       Console.WriteLine(s1);
   }
}
```

Output

```
Object
7
Object language
Object oriented language
```

Arrays of strings

We can also create and use arrays that contain strings. The statement

```
string[] itemArray = new string[3];

string[] itemArray ={“Java”,”C++”,”Csharp”};
```

```
using System;
using System.Text;

public class HelloWorld
{
   public static void Main(string[] args)
   {
       string[] countries={"India","Germany","America","France"};
       int n=countries.Length;
       Array.Sort(countries);
       for(int i=0;i<n;i++)
       {
          Console.WriteLine(countries[i]);
       }
       Console.WriteLine();
       Array.Reverse(countries);
       for(int i=0;i<n;i++)
       {
          Console.WriteLine(countries[i]);
       }

   }
}
```

Output

```
America
France
Germany
India

India
Germany
France
America
```

Regular Expression

- Regular Expression is a pattern which is used to parse and check whether the given input text is matching with the given pattern or not.
- In C#, Regular Expressions are generally termed as C# ***Regex***.
- Patterns may consist of any character literals, operators or constructors.
- C# provides a class termed as Regex which can be found in ***System.Text.RegularExpression*** namespace.

```
using System;
using System.Text.RegularExpressions;

public class HelloWorld
{
   public static void Main(string[] args)
   {
    string s1= @"support@gmail.com";
    string s2= @"^[a-z]{7,15}@[a-z]{5,8}.(com|.ac.in)$";

     Regex re=new Regex(s2);

     if(re.IsMatch(s1))
     {
        Console.WriteLine("Successful");
     }
     else
     {
        Console.WriteLine("UnSuccessful");
     }
   }
}
```

Output

```
Successful
```