Chairbear
Digital Learning

C#
Strings

Written 3rd July 2020

Table Of Contents

Table Of Contents 1

Introduction 3

Prerequisite C# 4

- Case Sensitivity 4
- Line Spacing 4
- Ending Statements 5
- Comments 5

Strings 5

- Properties Reference Table 5
  - Properties Examples 6
    - String Chars 6
    - String Length 6
- Methods Reference Table 7
  - Method Examples 12
    - Clone 12
    - Compare & Compare In Depth Explanation 12
    - CompareTo 14
    - Concat 14
    - Contains 15
    - Copy 15
    - CopyTo 15
    - EndsWith 15
    - Equals 15
    - Format 16
    - GetHashCode 16
    - GetType 16
    - GetTypeCode 16
    - IndexOf 16
    - Insert 16
    - Join 17
    - LastIndexOf 17
    - PadLeft 17
    - PadRight 17
    - Remove 17

Replace 17
Split 17
StartsWith 18
Substring 19
ToCharArray 19
ToLower 19
ToUpper 20
Trim 20
String Combination 20
Strings & Escape Characters 22

Introduction

This book was written for educational purposes, by Jade Lei ©Chairbear, for use by any individual or organization under the BY-NC-ND creative commons licence (Attribution + Noncommercial + NoDerivatives).

This book assumes you understand the basics of C#, however, does iterate over some of the basic syntax and rules of the language in the prerequisite section. If you are not familiar with C#, we would recommend reading our C# Essentials book.

This book covers the properties and methods applicable to strings, as well as additional information on strings in C#. It is a guide to use as a reference, and includes examples of usage.

We acknowledge that not everyone learns the same way, and so suggest you find additional resources that accommodate your learning style.

Jade Lei
SEO Technical Specialist,

Junior Software Developer (2020)

Prerequisite C#

Here are some of the fundamental C# rules and syntax that you should know before continuing to explore the rest of this book.

```
using System;



namespace fruits

{

 class Program

 {

   static void Main(string[] args)

   {

       var Fruit = "apple";

      var FRUIT = "apple";

      var fruit = "apple";

      Console.WriteLine(FRUIT);

   }

 }

}
```

Case Sensitivity

C# is case sensitive. The variables ‘fruit’, ‘FRUIT’ and ‘Fruit’ are all different variables.

Line Spacing

C# ignores additional spacing, like the above space after `using System;` but just because you can use lots of spaces and tabs doesn't mean you should, the best practise is to include a space

on either side of operator and assignment signs (such as the = sign), to make the code more human readable.

For this same reason, avoid code lines longer than 80 characters. If a C# statement does not fit on one line, the best place to break it is after an operator.

Ending Statements

All C# statements should end with a semicolon, as best practice.

Comments

Not all C# statements are executed, code after double slashes (//) or between /* and */ is treated as a comment. Comments are useful for adding notes to bits of code to make it easier to understand when revisiting it later.

```
//this is a single line comment

/* this is

a  multi line comment */
```

Strings

A string in C# is treated as an object, meaning there are properties and methods that can perform certain operations on strings.

Properties Reference Table

Property	***Meaning***	***Return Value***
str[1][positionIndex[2]]	**String Chars** You can access the individual characters in a string by referring to the characters index number inside square brackets. C# is a zero-based indexing system, meaning that it starts counting from zero	**Character** Returns the character from the position specified [positionIndex]

[1] str - string variable name
[2] positionIndex - the index at which the wanted character is sitting to return

str.Length	**String Length** Gets the number of characters in the current String object. Spaces count towards character length	**Number** Returns the number of characters in a string

Properties Examples

String Chars

```
string name = "Jem";

Console.WriteLine(name[1]);     //e
```

```
string alphabet = "abcdefghijklmnop";

char a = alphabet[0];

//a - Gets the character at position 0 (a) and assigns it to the variable called 'a'

Console.WriteLine(alphabet[alphabet.Length-1]);

//p - To get the last character, you can subtract one from the length
```

String Length

```
string name = "Jem";

Console.WriteLine(name.Length);     //3
```

```
string greeting = "Welcome to C#";

Console.WriteLine(greeting.Length); //13
```

Methods Reference Table

Method	*Meaning*	*Return Value*
str.Clone()	**String Clone** Returns a reference to this instance of the string	**Object** Returned value is not an independent copy of string instance
string[3].Compare(strOne[4],strTwo[5])	**String Compare** Used to compare the first string with the second string lexicographically[6]	**Integer** If both strings are equal, it returns 0. If the first string is greater (bigger) than the second string, it returns 1, otherwise it returns -1
strOne.CompareTo(strTwo)	**String Compare** Used to compare the first string with the second string lexicographically	**Integer** Similar to .Compare
string.Concat(strOne,strTwo)	**String Concat** Used to concatenate multiple string objects	**String** Returns the concatenated string
strOne.Contains(strTwo)	**String Contains** Used to check if specified substring(strTwo) occurs within this string(strOne) or not	**Boolean** Returns true if the specified substring is found in the string, else returns false
string.Copy(str)	**String Copy** Used to create a new instance of string with the same value as a specified string(str)	**String** Returns a copy of specified string(str)
str.CopyTo(index[7], destination[8],	**String Copy To**	**Char Array**

[3] Literally type string.
[4] strOne - first string to compare
[5] strTwo - second string to compare
[6] "is a generalisation of the way words are alphabetically ordered based on the alphabetical order of their component letters" - https://en.wikipedia.org/wiki/Lexicographical_order
[7] The index of the first character in the string instance (str) to copy
[8] An char array to which characters in the string instance are copied to

destinationIndex[9], count[10])	Used to copy a specified number of characters(count) from the specified position(index) in the string. It copies the characters of this string into a char array(destination) starting at a specified position(destinationIndex)	Returns modified char array
strOne.EndsWith(strTwo)	**String Ends With** Used to check whether the specified string (strOne) ends with the string specified in the brackets (strTwo)	**Boolean** Returns true if the string ends in the specified string, else returns false
strOne.Equals(strTwo)	**String Equals** Used to check whether two specified strings have the same value	**Boolean** If both strings have same value, it returns true, else it returns false
string.Format(str,object)	**String Format** Used to replace one+ format items in the specified string (str) with the string representation of a specified object (object)	**String** Returns formatted string
str.GetEnumerator()	**String Get Enumerator** Retrieves an object that can iterate through the individual characters in a string	**CharEnumerator** Returns a System.CharEnumerator
str.GetHashCode()	**String Get Hash Code** Used to get hash code of string	**Integer** Returns the hash value of a string
str.GetType()	**String Get Type** Used to get the type of the current object (str.)	**Data Type** Returns System.String for strings
str.GetTypeCode()	**String Get Type Code** Used to get the type of the current object (str.)	**Data Type** Returns String for strings
str.IndexOf('char')	**String Index Of**	**Integer**

[9] The index in *destination* at which the copy operation starts
[10] The number of characters in this string instance (str) to copy to *destination*

	Used to get index of the specified character ('char') from the string	Returns index of specified character in string
str.IndexOfAny(char[])	**String Index Of Any** Used to get index of the first occurrence of any character in a specified char array (char[]) from the string	**Integer** Returns index of specified character in string. Return -1 if no match was found
str.Insert(index,"str")	**String Insert** Used to insert the specified string ("str") at specified index number (index)	**String** Returns modified string
string.IsNullOrEmpty(str)	**String Is Null Or Empty** Is used to test whether the specified string is null or empty	**Boolean** Returns true if string is null or empty, else returns false
string.IsNullOrWhiteSpace(str)	**String Is Null,Empty Or Contains Only Whitspace** Is used to test whether the specified string is null, empty or contains only whitespace	**Boolean** Returns true if string is null or empty, else returns false
string.Join("Separator", array)	**String Join** Concatenates the elements in a array, separating the elements by the char separator specified, and returns them as a string	**String** Returns a string that consists of the elements of the specified array, delimited by the specified separator character
str.LastIndexOf(str,StartPosition)	**String Last Index Of** Used to get index of the specified string from the string, where searching starts at a specified character position (StartPosition) and proceeds backward toward the beginning of the string	**Integer** Returns starting position index of the specified string (str) in string (str.). If the string is not found, -1 is returned
str.LastIndexOfAny(char[],StartPosition)	**String Last Index Of Any** Used to get an index of the last occurrence of any character in a specified char array (char[]) from the string. The searching starts at the specified character position (StartPosition) and proceeds backward toward the	**Integer** Returns index of specified character in string. Return -1 if no match was found

	beginning of the string	
str.PadLeft(TotalLength[11], 'char padder')	**String Pad Left** Is used to create a new string that right-aligns the characters in the string (str.) by padding them on the left with a specified unicode character (char padder), to give a specified total length (TotalLength)	**String** It returns a modified string
str.PadRight(TotalLength[12], 'char padder')	**String Pad Right** Is used to create a new string that left-aligns the characters in the string (str.) by padding them on the right with a specified unicode character (char padder), to give a specified total length (TotalLength)	**String** It returns a modified string
str.Remove(Index)	**String Remove** Is used to create a new string where the specified characters in the current string have been deleted - beginning at a specified position (Index)	**String** It returns a modified string
str.Replace("Oldstr","Newstr")	**String Replace** Used to replace the old specified string instances("Oldstr") in str. with the new specified string instances ("Newstr")	**String** It returns a modified string
str.Split(Separator)	**String Split** Used to split a string into an string array, separating elements based on the separator provided	**String Array** Returns a string array
strOne.StartsWith(strTwo)	**String Starts With** Used to check whether the specified string (strOne) starts	**Boolean** Returns true if the string starts with the specified string, else returns false

[11] TotalLength - The number of characters in the resulting string = (the number of original characters + the additional padding characters)

[12] TotalLength - The number of characters in the resulting string = (the number of original characters + the additional padding characters)

	with the string specified in the brackets (strTwo)	
str.Substring(Index)	**String Substring** Is used to create a substring from the string (str). The substring starts at a specified character position (Index) and continues to the end of the string.	**String** It returns a modified string
str.ToCharArray(Index,count)	**String To Char Array** Is used to turn a string into a char array, starting at the specified index position (Index) and for the specified length (count)	**Char Array** Returns a char array
str.ToLower()	**String To Lowercase** Used to convert the string to all lowercase letters	**String** Returns a copy of the string converted to lowercase
str.ToLowerInvariant()	**String To Lower Invariant** Used to convert the string to all lowercase letters using the casing rules of the invariant culture	**String** Returns a copy of the string converted to lowercase
str.ToUpper()	**String To Uppercase** Used to convert the string to all uppercase letters	**String** Returns a copy of the string converted to uppercase
str.ToUpperInvariant()	**String To Uppercase** Used to convert the string to all uppercase letters using the casing rules of the invariant culture	**String** Returns a copy of the string converted to uppercase
str.Trim()	**String Trim** Is used to remove remove all leading and trailing white-space characters from the current string	**String** Returns a modified string

Method Examples

Clone

```
string greeting = "Welcome to C#";

Console.WriteLine(greeting);                    //Welcome to C#

string greeting2 = (string) greeting.Clone();

Console.WriteLine(greeting2);                   //Welcome to C#

greeting2 = "Hello";

Console.WriteLine(greeting2);                   //Hello

Console.WriteLine(greeting);                    //Welcome to C#
```

Compare & Compare In Depth Explanation

In life, to put items in order you must be able to compare them. Strings use the lexicographical order to compare and sort strings. This LO is the dictionary order, ei a comes before b, and c before d and so on, however with strings uppercase letters precede (come before) lowercase letters.

Two strings are lexicographically equal if they are the same length, and contain the same characters in the same position. The Compare() method returns 0 for this result.

If stringOne comes before stringTwo then a negative value (-1) is returned.

If stringTwo comes first then a positive value (1) is returned.

Another way to think of this is by assigning numeric values to the strings characters based on alphabetical order. See the example below.

When only working with lowercase letters we can use the below table.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	y	x	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

We can then replace letter characters in a string with a comma separated list of numbers that represent said character. This makes for much easier comparison. Like below where ‘hello’’s lexicographical order is decoded into `8,5,12,12,15.` We can then compare the two ‘string’s number by number until we come across a non matching number, in which case we then check

to see which number is higher. If it is the one in stringOne then .Compare will return 1, else if the higher number is in stringTwo then .Compare returns -1.

```
string str1 = "hello";  //8,5,12,12,15

string str2 = "hello";  //8,5,12,12,15

string str3 = "csharp"; //3,19,1,18,16

string abc = "abc";     //1,2,3

string def = "def";     //4,5,6



Console.WriteLine(string.Compare(str1,str2));  // 0 (8 is equal to 8)

Console.WriteLine(string.Compare(str2,str3));  // 1 (8 is greater than 3)

Console.WriteLine(string.Compare(abc,def)); // -1 (1 is not greater than 4)
```

Here is another example of comparing the 'string's number by number.

```
string str1 = "hello";  //8,5,12,12,15

string str2 = "helo";  //8,5,12,15

Console.WriteLine(string.Compare(str1,str2));  // -1 (12 is not greater than 15)
```

When working with capitals and lowercase letters in a string we can refer to the below table.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	Y	X	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**y**	**x**	**z**
27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52

We still do the same as what we'd do with all lowercase letters - decode into numbers for easier comparison.

```
string firstPersonFirstName = "Steve";   //19,46,31,48,31 ('Steve' LO )

string firstPersonLastName = "Rich";    //18,35,29,34 ('Rich' lexicographical order )
```

```
Console.WriteLine(string.Compare(firstPersonFirstName,firstPersonLastName)); // 1 (19
is greater than 18)
```

```
string fpFn = "Steve";      //19,46,31,48,31 ('Steve' lexicographical order )

string fpLn = "Sich";        //19,35,29,34 ('Sich' lexicographical order )

Console.WriteLine(string.Compare(fpFn,fpLn)); // 1 (46 is greater than 35)
```

CompareTo

```
string str1 = "hello";  //8,5,12,12,15

string str2 = "helo";  //8,5,12,15

Console.WriteLine(str1.CompareTo(str2));  // -1 (12 is not greater than 15)



string str11 = "hello";  //8,5,12,12,15

string str22 = "hello";  //8,5,12,12,15

string str33 = "csharp"; //3,19,1,18,16

string abc = "abc";     //1,2,3

string def = "def";     //4,5,6



Console.WriteLine(str11.CompareTo(str22));  // 0 (8 is equal to 8)

Console.WriteLine(str2.CompareTo(str33));  // 1 (8 is greater than 3)

Console.WriteLine(abc.CompareTo(def)); // -1 (1 is not greater than 4)
```

Concat

```
string name = "Jem";

string greeting = "Hello ";

string greeting2 = "! Welcome to C#";
```


```
Console.WriteLine(string.Concat(greeting, name, greeting2));

//Hello Jem! Welcome to C#
```

Contains

```
string name = "Jem";

string greeting = "Hello ";

Console.WriteLine(name.Contains(greeting)); //false
```

Copy

```
string name = "Jem";

string greeting = string.Copy(name);

Console.WriteLine(greeting); //Jem
```

CopyTo

```
string name = "Jem";

char[] Destination = {'h','e','l','l','o','t','o','m'};

name.CopyTo(0,Destination,5,3);

Console.WriteLine(Destination); //helloJem
```

EndsWith

```
string dog = "Fluffy Poodle";

string dogBreed = "Poodle";

Console.WriteLine(dog.EndsWith(dogBreed)); //true
```

Equals

```
string dog = "Fluffy Poodle";

string dogBreed = "Poodle";

Console.WriteLine(dog.Equals(dogBreed)); //false
```


Format

```
string CurrentDate = "Todays date is : {0}";

DateTime now = DateTime.Now;

string Formatted_Date = string.Format(CurrentDate,now);

Console.WriteLine(Formatted_Date);  // Todays date is : 7/17/2020 1:51:10 PM



string greeting = "Welcome to C#, {0}, I hope you are {1}";

Console.WriteLine(string.Format(greeting,"Jem", "ready to learn.")); //Welcome to
C#, Jem, I hope you are ready to learn.
```

GetHashCode

```
string dog = "Fluffy Poodle";

Console.WriteLine(dog.GetHashCode()); //1314283923
```

GetType

```
string dog = "Fluffy Poodle";

Console.WriteLine(dog.GetType()); //System.String
```

GetTypeCode

```
string dog = "Fluffy Poodle";

Console.WriteLine(dog.GetTypeCode()); //String
```

IndexOf

```
string dog = "Fluffy Poodle";

Console.WriteLine(dog.IndexOf('P')); //7
```

Insert

```
string dog = "Fluffy Poodle";

Console.WriteLine(dog.Insert(7,"Sh")); // Fluffy ShPoodle
```


Join

```
string[] fruit = {"Apple","Pear","Peach","Zitrone"};

Console.WriteLine(string.Join(", ",fruit));  //Apple, Pear, Peach, Zitrone
```

LastIndexOf

```
string fruit = "Apple,Pear,Peach,Zitrone";

Console.WriteLine(fruit.LastIndexOf("Pear"));  //6
```

PadLeft

```
string fruit = "Apple";

Console.WriteLine(fruit.PadLeft(10,'.'));  //.....Apple
```

PadRight

```
string fruit = "Apple";

Console.WriteLine(fruit.PadRight(10,'.'));   //Apple.....
```

Remove

```
string greeting = "Hello Jem. How Are You?";

Console.WriteLine(greeting.Remove(10)); //Hello Jem.
```

Replace

```
string greeting = "Hello Jem. How Are You?";

Console.WriteLine(greeting.Replace("Jem","Lorenzo")); //Hello Lorenzo. How Are You?
```

Split

```
string fruit = "Apple,Pear,Peach,Zitrone";

char separator = ',';

string[] strlist = fruit.Split(separator);

foreach (string I in strlist)

         {
```


```
           Console.WriteLine(I);

          }

/*

Apple

Pear

Peach

Zitrone

*/

string animals = "Bear & Dog & Cat & Rabbit";

string[] animalArray = animals.Split('&');

foreach (string I in animalArray)

         {

          Console.WriteLine(I);

         }

/*

Bear

Dog

Cat

Rabbit

*/
```

StartsWith

```
string greeting = "Hello Jem. How Are You?";

Console.WriteLine(greeting.StartsWith("Hello")); //true
```

Substring

```
string greeting = "Hello Jem. How Are You?";

Console.WriteLine(greeting.Substring(10)); // How Are You?
```

ToCharArray

```
string greeting = "Hello Jem. How Are You?";

char[] greetingArray = greeting.ToCharArray(10,13);

foreach (char I in greetingArray){

Console.WriteLine(I);

 /*

 H

 o

 w

 A

 r

 e

 Y

 o

 u

 ?

 */
```

ToLower

```
string name = "Jem";

Console.WriteLine(name.ToLower());  //jem
```


ToUpper

```
string name = "Jem";

Console.WriteLine(name.ToUpper());  //JEM
```

Trim

```
string fruit = "    Apple,Pear,Peach,Zitrone     ";

string cleanedFruit =   fruit.Trim();

Console.WriteLine(cleanedFruit); //Apple,Pear,Peach,Zitrone
```

String Combination

C# provides many ways to combine two+ strings into a single string.

String concatenation is the process of combining two or more strings together to create a bigger string. The best method to do this is called string interpolation, which uses placeholders in a string, which will then be replaced with values of variables later.

With interpolation the string is prepended with a dollar sign($), and the placeholders within the string are surrounded by curly braces.

```
string name = "Jem";

string greeting = $"Hello {name}! Welcome to C#";

Console.WriteLine(greeting);  //Hello Jem! Welcome to C#
```

An additional benefit of the interpolation is that you don't have to be mindful of spaces like the below methods.

Another way to concatenate strings in C# is with the + operator.

```
string name = "Jem";

string greeting = "Hello " + name + "! Welcome to C#";

Console.WriteLine(greeting);  //Hello Jem! Welcome to C#
```


Did you notice that we had to add a space after ‘hello’, if we didn’t the outcome would have been ‘HelloJem! Welcome to C#’. When you combine strings with the concatenation(+) method, you're literally placing them next to each other. So if you need a space, you have to add it.

Another way is with the addition assignment method.

```
string name = "Jem";

string greeting = "Hello " + name;

greeting += "! Welcome to C#";

Console.WriteLine(greeting); //Hello Jem! Welcome to C#
```

The final method is with the string.Concat() method.

```
string name = "Jem";

string greeting = "Hello ";

string greeting2 = "! Welcome to C#";

Console.WriteLine(string.Concat(greeting, name, greeting2));

//Hello Jem! Welcome to C#
```

In C#, the plus (+) operator is used for both addition and concatenation. The difference between the usage is that when used with numbers it is interpreted as addition, whilst when used with strings it is concatenation.

```
int x = 10, y = 20;

int z = x + y;  //expected output : 30

Console.WriteLine(z);

string a = "10", b = "20";

string c = a + b; //expected output : 1020

Console.WriteLine(c);
```

Typically, if you add a number and a string, the result will be a string concatenation. However, as C# interprets from left to right, it depends on the placement of numbers in the expression.

```
int x = 10;

int y = 20;

var z = "X and Y added is: " + x + y;

Console.WriteLine(z);

/*expected output : 'X and Y added is: 1020' x and y are not added

as C# hits the string first, and so treats numbers after it as strings too*/
```

```
var a = 10;

var b = 20;

var c = "10";

var d = a + b + c;

Console.WriteLine(d);

/*expected output : 3010 as C# hits the numbers first, and so adds them correctly,

however, once it hits the string, it changes from addition to concatenation,

and outputs the string '3010'. */
```

Strings & Escape Characters

Because strings must be written within quotes, C# will misunderstand additional double quote marks within, and cut the string short. The solution to avoid this problem, is to use the backslash escape character, which turns special characters into string characters.

Escape Sequence	***Meaning***
\'	Single Quote
\"	Double Quote
\\	Backslash
\b	Backspace

\n	New Line
\t	Horizontal Tabulator

```
string mood = “I\’m ok”;
```