# C# Strings

In C#, string is an object of **System.String** class that represent sequence of characters. We can perform many operations on strings such as concatenation, comparision, getting substring, search, trim, replacement etc.

**1. String Declaration and Initialization**

* **Declare a string without initialization:**

string str1;

* **Declare a string with initialization:**

string str2 = "THILLAI";

string str3 = "TAMIZHVANI";

* **Declare a string with an empty string:**

string str4 = string.Empty;

* **Declare a string with null:**

string str5 = null;

* **Declare a string from a character array:**

char[] char\_arr = new char[] { 'T', 'A', 'M', 'I', 'Z', 'H' };

String str6 = new string(char\_arr);

**2. String vs string**

* **In C#, the string keyword is just an alias for String. Both are equivalent, and you can use either to define string variables.**

string str1 = "Hello";

String str2 = "World";

**3. String Immutable**

* **Strings in C# are immutable, meaning once created, they cannot be modified.**

string msg = "welcome";

msg = "welcome to tutlane";

*Explanation: The original "welcome" string is discarded, and a new string "welcome to tutlane" is created.*

**4. String Literals in Regular**

* **Regular literals in C# are sequences of characters enclosed in double quotation marks (" ").**
* **Useful for embedding escape characters like \n, \t, ', ", etc.**

string names = "THILLAI\nTAMILZH\nTAMIZHILLAI";

Console.WriteLine(names);

// Output:

// THILLAI

// TAMILZH

// TAMIZHILLAI

string msg = "Hello \"WORLD\" ";

Console.WriteLine(msg);

// Output: Hello "WORLD"

**5. String Literals in Verbatim**

* **Verbatim literals are indicated by the special character @.**
* **Useful for representing multiline strings or strings with backslash characters.**

string imagepath = @"C:\Users\Thillai\TAMIL.jpg";

Console.WriteLine(imagepath);

// Output: C:\Users\Thillai\TAMIL.jpg

string msg = @"THIS,

IS STRING VERBATIM

FORMAT";

Console.WriteLine(msg);

// Output:

// THIS,

// IS STRING VERBATIM

// FORMAT

string msg2 = @"MY daughter name was ""TAMIZHILLAI"" ";

Console.WriteLine(msg2);

// Output: MY daughter name was "TAMIZHILLAI"

**6. String Format**

* **Format strings are strings whose contents are determined dynamically at runtime.**
* **Use the Format method to embed placeholders in braces, which will be replaced by values at runtime.**

string name = "Thillai Shanmugam";

string location = "Kumbakonam";

string user = string.Format("Name: {0}, Location: {1}", name, location);

Console.WriteLine(user);

// Output: Name: Thillai Shanmugam, Location: Kumbakonam

**7. Access Individual Characters from Strings**

* **Access individual characters in a string using the [] indexer.**

string name = "Thillai Shanmugam";

for (int i = 0; i < name.Length; i++)

{

Console.Write(name[i]);

}

// Output: Thillai Shanmugam

**8. String Properties**

* **Chars - Gets the characters from the current string object based on the specified position.**

string name = "Thillai";

char firstChar = name[0];

Console.WriteLine(firstChar); // Output: T

* **Length - Returns the number of characters in the current string object.**

string name = "Shanmugam";

int length = name.Length;

Console.WriteLine(length); // Output: 9

## ****C# String Methods Example****

This example demonstrates various String methods in C#. Each method is used to manipulate or query the string, with the output clearly displayed to illustrate its functionality.

### ****Code Example****

using System;

namespace StringMethodsExample

{

class Program

{

static void Main(string[] args)

{

string str1 = "Hello, World!";

string str2 = "Hello, C#";

string str3 = " Learn C# Programming ";

// 1. Clone

string clonedString = (string)str1.Clone();

Console.WriteLine($"Cloned String: {clonedString}");

// 2. Compare

int compareResult = String.Compare(str1, str2);

Console.WriteLine($"Compare Result (str1 vs str2): {compareResult}");

// 3. CompareOrdinal

int compareOrdinalResult = String.CompareOrdinal(str1, str2);

Console.WriteLine($"CompareOrdinal Result (str1 vs str2): {compareOrdinalResult}");

// 4. CompareTo

int compareToResult = str1.CompareTo(str2);

Console.WriteLine($"CompareTo Result (str1 vs str2): {compareToResult}");

// 5. Concat

string concatenatedString = String.Concat(str1, " ", str2);

Console.WriteLine($"Concatenated String: {concatenatedString}");

// 6. Contains

bool containsResult = str1.Contains("World");

Console.WriteLine($"Contains 'World': {containsResult}");

// 7. Copy

string copiedString = String.Copy(str1);

Console.WriteLine($"Copied String: {copiedString}");

// 8. CopyTo

char[] charArray = new char[5];

str1.CopyTo(7, charArray, 0, 5);

Console.WriteLine($"CopyTo Result: {new string(charArray)}");

// 9. EndsWith

bool endsWithResult = str1.EndsWith("World!");

Console.WriteLine($"EndsWith 'World!': {endsWithResult}");

// 10. Equals

bool equalsResult = String.Equals(str1, str2);

Console.WriteLine($"Equals (str1 vs str2): {equalsResult}");

// 11. Format

string formattedString = String.Format("Welcome to {0} Programming", "C#");

Console.WriteLine($"Formatted String: {formattedString}");

// 12. GetEnumerator

var enumerator = str1.GetEnumerator();

Console.Write("GetEnumerator Result: ");

while (enumerator.MoveNext())

{

Console.Write(enumerator.Current + " ");

}

Console.WriteLine();

// 13. GetHashCode

int hashCode = str1.GetHashCode();

Console.WriteLine($"GetHashCode: {hashCode}");

// 14. GetType

Type type = str1.GetType();

Console.WriteLine($"GetType: {type}");

// 15. GetTypeCode

TypeCode typeCode = str1.GetTypeCode();

Console.WriteLine($"GetTypeCode: {typeCode}");

// 16. IndexOf

int indexOfResult = str1.IndexOf("World");

Console.WriteLine($"IndexOf 'World': {indexOfResult}");

// 17. Insert

string insertedString = str1.Insert(7, "Beautiful ");

Console.WriteLine($"Inserted String: {insertedString}");

// 18. Intern

string internedString = String.Intern(str2);

Console.WriteLine($"Interned String: {internedString}");

// 19. IsInterned

string isInternedResult = String.IsInterned(str2);

Console.WriteLine($"IsInterned Result: {isInternedResult}");

// 20. IsNormalized

bool isNormalizedResult = str1.IsNormalized();

Console.WriteLine($"IsNormalized: {isNormalizedResult}");

// 21. IsNullOrEmpty

bool isNullOrEmptyResult = String.IsNullOrEmpty(str1);

Console.WriteLine($"IsNullOrEmpty: {isNullOrEmptyResult}");

// 22. IsNullOrWhiteSpace

bool isNullOrWhiteSpaceResult = String.IsNullOrWhiteSpace(str3);

Console.WriteLine($"IsNullOrWhiteSpace: {isNullOrWhiteSpaceResult}");

// 23. Join

string[] words = { "Learn", "C#", "Programming" };

string joinedString = String.Join(" ", words);

Console.WriteLine($"Joined String: {joinedString}");

// 24. LastIndexOf

int lastIndexOfResult = str1.LastIndexOf('o');

Console.WriteLine($"LastIndexOf 'o': {lastIndexOfResult}");

// 25. LastIndexOfAny

char[] chars = { 'H', 'W' };

int lastIndexOfAnyResult = str1.LastIndexOfAny(chars);

Console.WriteLine($"LastIndexOfAny 'H' or 'W': {lastIndexOfAnyResult}");

// 26. Normalize

string normalizedString = str1.Normalize();

Console.WriteLine($"Normalized String: {normalizedString}");

// 27. PadLeft

string padLeftString = str1.PadLeft(20);

Console.WriteLine($"PadLeft: '{padLeftString}'");

// 28. PadRight

string padRightString = str1.PadRight(20);

Console.WriteLine($"PadRight: '{padRightString}'");

// 29. Remove

string removedString = str1.Remove(7);

Console.WriteLine($"Removed String: {removedString}");

// 30. Replace

string replacedString = str1.Replace("World", "Everyone");

Console.WriteLine($"Replaced String: {replacedString}");

// 31. Split

string[] splitArray = str3.Split(' ');

Console.WriteLine("Split Result:");

foreach (string s in splitArray)

{

Console.WriteLine($"'{s}'");

}

// 32. StartsWith

bool startsWithResult = str1.StartsWith("Hello");

Console.WriteLine($"StartsWith 'Hello': {startsWithResult}");

// 33. Substring

string substringResult = str1.Substring(7);

Console.WriteLine($"Substring from index 7: {substringResult}");

// 34. ToCharArray

char[] toCharArrayResult = str1.ToCharArray();

Console.WriteLine("ToCharArray Result:");

foreach (char c in toCharArrayResult)

{

Console.WriteLine(c);

}

// 35. ToLower

string toLowerResult = str1.ToLower();

Console.WriteLine($"ToLower: {toLowerResult}");

// 36. ToLowerInvariant

string toLowerInvariantResult = str1.ToLowerInvariant();

Console.WriteLine($"ToLowerInvariant: {toLowerInvariantResult}");

// 37. ToString

string toStringResult = str1.ToString();

Console.WriteLine($"ToString: {toStringResult}");

// 38. ToUpper

string toUpperResult = str1.ToUpper();

Console.WriteLine($"ToUpper: {toUpperResult}");

// 39. Trim

string trimmedString = str3.Trim();

Console.WriteLine($"Trimmed String: '{trimmedString}'");

// 40. TrimEnd

string trimEndString = str3.TrimEnd();

Console.WriteLine($"TrimEnd: '{trimEndString}'");

// 41. TrimStart

string trimStartString = str3.TrimStart();

Console.WriteLine($"TrimStart: '{trimStartString}'");

}

}

}

### ****Output****

Cloned String: Hello, World!

Compare Result (str1 vs str2): 1

CompareOrdinal Result (str1 vs str2): 1

CompareTo Result (str1 vs str2): 1

Concatenated String: Hello, World! Hello, C#

Contains 'World': True

Copied String: Hello, World!

CopyTo Result: World

EndsWith 'World!': True

Equals (str1 vs str2): False

Formatted String: Welcome to C# Programming

GetEnumerator Result: H e l l o , W o r l d !

GetHashCode: -694847

GetType: System.String

GetTypeCode: String

IndexOf 'World': 7

Inserted String: Hello, Beautiful World!

Interned String: Hello, C#

IsInterned Result: Hello, C#

IsNormalized: True

IsNullOrEmpty: False

IsNullOrWhiteSpace: False

Joined String: Learn C# Programming

LastIndexOf 'o': 8

LastIndexOfAny 'H' or 'W': 7

Normalized String: Hello, World!

PadLeft: ' Hello, World!'

PadRight: 'Hello, World! '

Removed String: Hello,

Replaced String: Hello, Everyone!

Split Result:

''

''

''

'Learn'

'C#'

'Programming'

''

''

''

StartsWith 'Hello': True

Substring from index 7: World!

ToCharArray Result:

H

e

l

l

o

,

W

o

r

l

d

!

ToLower: hello, world!

ToLowerInvariant: hello, world!

ToString: Hello, World!

ToUpper: HELLO, WORLD!

Trimmed String: 'Learn C# Programming'

TrimEnd: ' Learn C# Programming'

TrimStart: 'Learn C# Programming '

### ****Explanation****

* **Cloning**: Clone() creates a copy of the string.
* **Comparison**: Compare(), CompareOrdinal(), and CompareTo() compare two strings based on their sort order.
* **Concatenation**: Concat() joins multiple strings together.
* **Contains**: Contains() checks if a substring exists within the string.
* **Copy**: Copy() and CopyTo() methods create and copy strings.
* **EndsWith**: EndsWith() checks if the string ends with a specific substring.
* **Equality**: Equals() checks if two strings have the same value.
* **Formatting**: Format() replaces placeholders with values.
* **Enumeration**: GetEnumerator() retrieves characters in a string.
* **Hash Code**: GetHashCode() returns the string's hash code.
* **Type Info**: GetType() and GetTypeCode() return type information.
* **Indexing**: IndexOf() and LastIndexOf() locate characters or substrings.
* **Insertion**: Insert() adds a substring at a specific index.
* **Interning**: Intern() and IsInterned() manage string interning.
* **Normalization**: IsNormalized() and Normalize() ensure consistent Unicode formatting.
* **Null/Empty/Whitespace**: IsNullOrEmpty() and IsNullOrWhiteSpace() check for empty or whitespace-only strings.
* **Joining**: Join() combines array elements into a single string.
* **Padding**: PadLeft() and PadRight() align strings with spaces.
* **Removal**: Remove() deletes parts of a string.
* **Replacement**: Replace() substitutes one substring for another.
* **Splitting**: Split() divides a string into an array based on delimiters.
* **Start Check**: StartsWith() checks if a string begins with a specific substring.
* **Substrings**: Substring() extracts a portion of the string.
* **Character Array**: ToCharArray() converts a string to a character array.
* **Case Conversion**: ToLower(), ToLowerInvariant(), and ToUpper() change the case of a string.
* **Trimming**: Trim(), TrimEnd(), and TrimStart() remove whitespace.

### ****C# StringBuilder Class****

The StringBuilder class in C# is used to represent a mutable string of characters. Unlike the string class, which is immutable (cannot be changed once created), the StringBuilder allows for modifications to the string without creating new instances, which can be more efficient for scenarios where frequent changes to the string are required.

The StringBuilder class is part of the System.Text namespace and provides a variety of methods for manipulating strings.

#### ****Why Use StringBuilder?****

* **Efficiency**: Modifying a string using StringBuilder does not create a new object in memory, which reduces overhead and improves performance, especially in loops or when performing many modifications.
* **Mutable Strings**: StringBuilder is mutable, meaning that once an instance is created, the same instance is used for all modifications, reducing memory consumption.

#### ****Common StringBuilder Methods****

1. **Append**: Appends a string to the end of the current StringBuilder.
2. **AppendFormat**: Appends a formatted string (e.g., using placeholders) to the StringBuilder.
3. **Insert**: Inserts a string at a specified index.
4. **Remove**: Removes a specified number of characters starting at a given index.
5. **Replace**: Replaces all occurrences of a specified string with another string.

### ****Example Usage of StringBuilder Methods****

using System;

using System.Text;

namespace StringBuilderExample

{

class Program

{

static void Main(string[] args)

{

// 1. Append: Adding strings to the StringBuilder

StringBuilder sb = new StringBuilder("Suresh");

sb.Append(", Rohini");

sb.Append(", Trishika");

Console.WriteLine(sb);

// Output: Suresh, Rohini, Trishika

// 2. AppendFormat: Adding formatted string

int amount = 146;

StringBuilder sb1 = new StringBuilder("Total");

sb1.AppendFormat(": {0:c}", amount);

Console.WriteLine(sb1);

// Output: Total: $146.00

// 3. Insert: Inserting a string at a specific position

StringBuilder sb2 = new StringBuilder("Welcome Tutlane");

sb2.Insert(8, "to ");

Console.WriteLine(sb2);

// Output: Welcome to Tutlane

// 4. Remove: Removing characters from a specific index

StringBuilder sb3 = new StringBuilder("Welcome to Tutlane");

sb3.Remove(8, 3); // Removing "to "

Console.WriteLine(sb3);

// Output: Welcome Tutlane

// 5. Replace: Replacing occurrences of a string

StringBuilder sb4 = new StringBuilder("Welcome to Tutlane");

sb4.Replace("Tutlane", "C#");

Console.WriteLine(sb4);

// Output: Welcome to C#

}

}

}

### ****Explanation****

1. **Append**: The Append method is used to add strings to the end of the current StringBuilder instance. This is useful for concatenating strings without creating multiple string instances.
   * **Output**: Suresh, Rohini, Trishika
2. **AppendFormat**: The AppendFormat method formats the string using placeholders and appends it to the StringBuilder.
   * **Output**: Total: $146.00
3. **Insert**: The Insert method allows you to insert a string at a specific position in the StringBuilder.
   * **Output**: Welcome to Tutlane
4. **Remove**: The Remove method removes a specified number of characters starting from a given index.
   * **Output**: Welcome Tutlane (removes "to ")
5. **Replace**: The Replace method replaces all occurrences of a specified string with another string in the StringBuilder.
   * **Output**: Welcome to C# (replaces "Tutlane" with "C#")

### ****When to Use StringBuilder?****

* Use StringBuilder when you need to perform multiple operations on a string, such as appending, inserting, or removing characters, especially in loops or when handling large amounts of text.
* For simple, infrequent string manipulations, the string class might still be more straightforward and readable.

## ****String****

### Example:

string str1 = "Welcome";

str1 = "Welcome to Tutlance";

### ****Memory Representation:****

| **RAM** |
| --- |
| **Stack** |
|  |
| str1 -> 0x02345 |
| str1 -> 0x07896 |
| **Heap** |
|  |
| 0x02345: "Welcome" |
| 0x07896: "Welcome to Tutlance" |
|  |
|  |

### ****Explanation:****

* When you initially assign str1 = "Welcome", the string "Welcome" is stored in the heap at address 0x02345.
* When you modify str1 to "Welcome to Tutlance", a new string is created in the heap at address 0x07896.
* The original memory location is discarded, and str1 now points to the new address 0x07896.

### ****Summary:****

* **Strings are immutable**: Every time you modify a string, a new memory location is allocated in the heap, and the reference is updated on the stack.

## ****StringBuilder****

### Example:

csharp

Copy code

StringBuilder sb = new StringBuilder("Welcome");

sb.Append(" to Tutlance");

### ****Memory Representation:****

| **RAM** |
| --- |
| **Stack** |
|  |
| sb -> 0x02345 |
|  |
|  |
|  |
| **Heap** |
|  |
| 0x02345: "Welcome to Tutlance" |
|  |
|  |
|  |

### ****Explanation:****

* When you create the StringBuilder with "Welcome", the string is stored in the heap at address 0x02345.
* When you modify the string by appending " to Tutlance", the same memory location in the heap (0x02345) is updated with the new value "Welcome to Tutlance".
* No new memory allocation occurs, and the reference remains unchanged on the stack.

### ****Summary:****

* **StringBuilder is mutable**: Modifications to the string are done in place, without creating new memory locations in the heap, making it more memory-efficient for repeated operations.