# **Рядки(String Class)**

**Рядок** – це об’єкт типу **string**(System.String в нотації .Net), який зберігає в собі текстовий рядок. Хоча рядки відносять до примітивних типів даних, насправді вони мають **тип за посиланням**. Всередині рядків зберігається колекція символів типу char.

Рядок є **незмінним об’єктом**, це означає, що після створення його не можна змінити, а будь які зміни, до прикладу об’єднання, призводять до створення нового рядку.

Оскільки рядкові змінні дуже часто використовуються в програмування, мова C# містить велику кількість операцій для роботи з рядками.

using System;

namespace CsharpString {

class Test {

public static void Main(string [] args) {

// create string

string str1 = "C# Programming";

string str2 = "Programiz";

// print string

Console.WriteLine(str1);

Console.WriteLine(str2);

Console.ReadLine();

}

}

}

Тут ми створили рядок з ім'ям str і присвоїли йому текст "C# Programming". Для представлення рядків у C# ми використовуємо подвійні лапки.

Рядкова змінна в C# не належить до примітивних типів, таких як int, char тощо. Замість цього вона є об'єктом класу String.

**Операції над рядками**

У мові C# існує безліч методів для виконання різних операцій над рядками. Ми розглянемо деякі з найпоширеніших операцій над рядками.

***1. Отримання довжини рядка Length***

Для знаходження довжини рядка використовується властивість **Length**.

public static void Main(string [] args) {

// create string

string str = "C# Programming";

Console.WriteLine("string: " + str);

// get length of str

int length = str.Length;

Console.WriteLine("Length: "+ length);

Console.ReadLine();

}

**Output**

string: C# Programming

Length: 14

У наведеному вище прикладі властивість **Length** обчислює загальну кількість символів у рядку і повертає її.

***2. Об'єднання рядків Concat()***

З'єднати два рядки в C# можна з допомогою методу **Concat()**, або додати оператори ***“+”*** та ***“+=”***.

*//With Contact() method*

public static void Main(string [] args) {

// create string

string str1 = "C# ";

Console.WriteLine("string str1: " + str1);

// create string

string str2 = "Programming";

Console.WriteLine("string str2: " + str2);

// join two strings

string joinedString = string.Concat(str1, str2);

Console.WriteLine("Joined string: " + joinedString);

Console.ReadLine();

}

**Output**

string str1: C#

string str2: Programming

Joined string: C# Programming

Тут метод **Concat()** об'єднує **str1** і **str2** і присвоює їх змінній **joinedString**.

*//With operators + and +=*

string h = "Hello";

string w = "World";

h += " "; //додаємо пробіл

w = w + "!"; //склеюємо зі знаком оклику

Console.WriteLine(string.Concat(h, w)); //склеює два рядки

***3.Створення рядків new***

Рядок можна створити присвоївши йому значення рядкового літералу, чи за допомогою оператора **new**.

string text1 = null; //рядок без значення

string text2 = ""; //пустий рядок

string text3 = "Some text";

string text4 = new string('x', 4); //xxxx

string text5 = new string(new[] { 'M', 'e', 's', 's', 'a', 'g', 'e' }); //Message

## ***4.Пустий рядок та рядок без значення Empty***

## **Пустий рядок** – це об’єкт типу string, в якому не міститься символів, його довжина рівна нулю. На відміну від порожнього рядка, рядок без значення **(*null-string*)**, не містить посилання на об’єкт.Створити порожній рядок можна одним із способів.

var empty1 = "";

var empty2 = string.Empty;

### ***4.1.IsNullOrEmpty - перевірка на пустоту***

Для перевірки на null і пустоту, використовується метод IsNullOrEmpty, він повертає **true** якщо рядок пустий, або рівний **null**.

string nullString = null;

var b1 = string.IsNullOrEmpty(nullString); //true

var b2 = string.IsNullOrEmpty("test"); // false

### ***4.2.IsNullOrWhiteSpace***

Метод приймає на вхід рядок і повертає **true** якщо рядок рівний **null**, або складається з пробільних символів.

string whiteString = "\t \r\n";

var b1 = string.IsNullOrWhiteSpace(whiteString); //true

var b2 = string.IsNullOrWhiteSpace("123"); // false

## 

## 

## 

## ***5.Зміна регістру ToLower, ToUpper.***

Для перетворення рядків у нижній регістр використовується метод **ToLower**, у верхній – **ToUpper**.

var t = "Abdc";

var lower = t.ToLower(); //abcd

var upper = t.ToUpper(); //ABCD

***6.Порівняння рядків Equals()***

У мові C# можна виконувати порівняння між двома рядками з допомогою методу **Equals()**. Метод **Equals()** перевіряє, чи є два рядки рівними (аналог оператора “==”).

public static void Main(string [] args) {

// create string

string str1 = "C# Programming";

string str2 = "C# Programming";

string str3 = "Programiz";

// compare str1 and str2

Boolean result1 = str1.Equals(str2);

Console.WriteLine("string str1 and str2 are equal: " + result1);

//compare str1 and str3

Boolean result2 = str1.Equals(str3);

Console.WriteLine("string str1 and str3 are equal: " + result2);

Console.ReadLine();

}

**Output**

string str1 and str2 are equal: True

string str1 and str3 are equal: False

У вищенаведеному прикладі ми створили 3 рядки з іменами **str1**, **str2** і **str3**. Тут ми використовуємо метод **Equals()** для перевірки рівності одного рядка іншому. **Equals** може приймати додатковий аргумент **StringComparison**, який визначає налаштування порівняння.

### ***6.1.Compare***

Порівнює значення двох рядків, повертає ціле число:

| **Число** | **Значення** |
| --- | --- |
| **Від’ємне** | Перший рядок менше другого, або має значення null |
| **Нуль** | Рядки рівні |
| **Додатне** | Другий рядок менше першого, чи має значення null |

*порівняння проводиться в порядку сортування, при цьому “A” менше “B”, “D” більше “С”.*

int x = string.Compare( "qwerty", "asdfg"); // > 0

### 

### ***6.2.CompareOrdinal***

Порівняння двох рядків без врахування мови та регіональних параметрів, повертає результат аналогічно попередньому методу.

int y = string.CompareOrdinal( "v", "v"); // == 0

### ***6.3.StartsWith***

Повертає **true** якщо рядок починається з підрядка, і іншому випадку **false**

var st = "0987654321";

bool b = st.StartsWith("09"); //true

### ***6.4.EndsWith***

Повертає **true** якщо рядок закінчується на підрядок, в іншому випадку **false**

var st = "0987654321";

bool b = st.EndsWith("32"); //false

***7.Пошук підрядків***

### ***IndexOf***

Проводить пошук і повертає позицію початку першого входження підрядка(символа), або -1 якщо нічого не знайдено.

int i1 = "twit".IndexOf("t"); //0

### ***LastIndexOf***

Шукає заданий текст в рядку і повертає позицію початку останнього входження підрядку(символу), або -1 якщо нічого не знайдено.

int i2 = "twit".LastIndexOf("t"); //3

## ***7.1.Trim - видалення пробілів***

Методи для видалення символів-розділення з рядка:

* **Trim – видаляє символи з початку і кінця рядка;**
* **TrimStart – тільки на початку;**
* **TrimEnd – тільки з кінця;**

string s = " 1 Test ";

var trim = s.Trim(); // "1 Test"

var start = s.TrimStart(); // "1 Test "

var end = s.TrimEnd(); // " 1 Test"

var custom = s.Trim(' ', 't'); // "1 Tes"

*Ви можете передати в якості аргументу методу, свій набір символів, які потрібно видалити з рядка.*

## ***7.2.Розділення Split***

Для того, щоб розділити рядок на частини по розділювачу(можна задати декілька відразу), використовують метод **Split**.

string[] parts = "10,11,12".Split(","); // 10, 11, 12

Якщо таким чином розділити рядок з декількома розділювачами підряд, то в результуючий масив потрапляють пусті рядки. Для виключення такого варіанту, можна скористатися методом **Split** з параметром **StringSplitOptions.RemoveEmptyEntries**:

string csv = "a;b;c;;d;f;g;";

string[] parts = csv.Split(";", StringSplitOptions.RemoveEmptyEntries);

foreach(var item in parts)

{

Console.WriteLine(item);

}

## ***7.3.Отримання підрядка Substring***

Метод **Substring** дозволяє отримати символ або підрядок з рядка, в якості аргументів задається початковий індекс і довжина(якщо не вказана, то до кінця рядка).

string k = "United Kingdom";

var r1 = k.Substring(0, 6); //United

var r2 = k.Substring(7, 4); //King

var r3 = k.Substring(7); //Kingdom

## ***8.Редагування рядків***

До операцій редагування відносять:

* **пошук та заміна;**
* **вставка підрядка;**
* **видалення частини рядка;**
* **функції доповнення.**

### ***8.1.Заміна Replace***

Для заміни символів чи частини рядка, використовується метод **Replace**, як аргументи він приймає текст для пошуку та текст на який його потрібно замінити(аналог операції знайти і замінити в текстових редакторах):

//заміна слова

string n = "United States";

string m = n.Replace("States", "Kingdom");

Console.WriteLine(m); //United Kingdom

//багатократна заміна символів

var f = "2;4;6;8;10";

Console.WriteLine(f.Replace(";", ", ")); //2, 4, 6, 8, 10

### ***8.2.Вставка Insert***

Метод **Insert** призначений для вставки підрядка з вказаної в якості аргументу позиції.

string v = "Some message";

//вставка слова

string p = v.Insert(5, "text");

Console.WriteLine(p); //Some textmessage

//вставляємо пробіл між словами

Console.WriteLine(p.Insert(9, " ")); //Some text message

### ***8.3.Видалення Remove***

Метод **Remove** дозволяє видалити частину рядка. В якості аргументів приймає позицію з якої починається видалення, та кількість символів, які потрібно видалити(якщо кількість не вказана, видаляються всі символи до кінця рядка):

string text = "London is the Capital of Great Britain";

//видаляємо слово Great

Console.WriteLine(text.Remove(25, 6));

//залишаємо тільки London

Console.WriteLine(text.Remove(6));

### ***8.4.Доповнення рядків***

Для доповнення рядка символами до вказаної довжини, використовуються методи:

* **PadLeft** – повертає новий рядок доповнений зліва вказаними символами;
* **PadRight** – доповнює вказаними символами справа.

Якщо символ доповнення не вказаний, використовується пробіл.

Ці методи зручно використовувати для виводу чисел та тексту з вирівнюванням:

string[] nums = new[] { "10", "1023", "2" };

//виводимо числа з вирівнюванням по правому краю

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

{

Console.WriteLine(nums[i].PadLeft(4));

}

// Результат

// 10

//1023

// 2

//додавання не значущих нулів до двійкових чисел

string[] binNums = new[] {"101", "111010", "11001"};

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

{

Console.WriteLine(binNums[i].PadLeft(8, '0'));

}

//00000101

//00111010

//00011001

//доповнення справа

string tree = "|";

for (int i = 1; i < 6; i++)

Console.WriteLine(tree.PadRight(i, '\*'));

**Послідовності екранування рядків**

**(String Escape Sequences)**

Символ екранування використовується для вилучення деяких символів, присутніх у рядку. Іншими словами, ми використовуємо екрановані послідовності для вставки спеціальних символів всередині рядка.

| **Escape Sequence** | **Character Name** |
| --- | --- |
| **\'** | **single quote**  **(одинарні лапки)** |
| **\"** | **double quote**  **(подвійні лапки)** |
| **\\** | **backslash**  **(зворотний слеш)** |
| **\0** | **null**  **(ноль)** |
| **\n** | **new line**  **(нова лінія)** |
| **\t** | **horizontal tab**  **(горизонтальна вкладка)** |

**Інтерполяція рядка з використанням *“$”***

У C# ми можемо використовувати інтерполяцію рядків для вставки змінних всередині рядка. Для рядкової інтерполяції рядковий літерал повинен починатися з символу **$**.

public static void Main(string [] args)

{

// create string

string name = "Programiz";

// string interpolation

string message = $"Welcome to {name}";

Console.WriteLine(message);

Console.ReadLine();

}

**Output**

Welcome to Programiz

У наведеному вище прикладі ми використовуємо змінну **name** всередині рядка **message**.

Зверніть увагу, якщо рядковий літерал починається з **$,** ім'я змінної розміщується всередині **фігурних дужок {}**