

# Робота з рядками

## 1. Класи для роботи з рядками

У мові C # строкові значення представляє тип `string`, а вся функціональність роботи з даним типом зосереджена в класі `System.String`. Власне `string` є псевдонімом для класу `System.String`. Об'єкти цього класу представляють текст як послідовність символів Unicode. Максимальний розмір об'єкта `String` може становити в пам'яті 2 ГБ, або близько 1 мільярда символів.

Створювати рядки можна, як використовуючи змінну типу `string` і привласнюючи їй значення, так і застосовуючи один з конструкторів класу `String`:

```
string s1 = "hello";
string s2 = null;

string s3 = new String('a', 6); // результатом будет строка "aaaaaa"
string s4 = new String(new char[]{'w', 'o', 'r', 'l', 'd'});
```

Не дивлячись на те, що `String` є об'єктним типом, оператори `==` та `!=` визначені для порівняння вмісту об'єктів типу рядок, а не посилань як для інших класів. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class Program {
    static void Main(string[] args) {
        string a = "hello";
        string b = "h";
        // Append to contents of 'b'
        b += "ello";
        Console.WriteLine(a == b);
        Console.WriteLine((object)a == (object)b);
        Console.ReadKey();
    }
}
```

Виведе на екран:

```
True
False
```

Що показує, те що вміст рядків однаковий, хоча об'єкти різні.

Для рядків `String` визначено оператори `+` та `+=` для додавання.

Проте слід зазначити, що ці об'єкти являються незмінними: після створення їх неможливо змінити. Все методи `String` і оператори `C#`, які, начебто змінюють рядок, насправді повертають в результаті новий об'єкт-рядок.

```
using System;

class Program {
    static void Main(string[] args) {
        string a, b = "h";
        a = b;          //a і b посилаються на один об'єкт
        Console.WriteLine((object)a == (object)b);
        b += "ello";    //тепер це різні об'єкти
        Console.WriteLine((object)a == (object)b);
        Console.WriteLine("a==" + a);
        Console.WriteLine("b==" + b);
        Console.ReadKey();
    }
}
```

На екрані буде:

```
True
False
a==h
b==hello
```

Тобто спочатку посилання `a` і `b` були на один об'єкт, а після виконання оператора `+=` змінна `b` стала вказувати на новий об'єкт.

Так само, і оператор `[]` служить тільки для доступу для читання окремих символів, та не може використовуватись для зміни значень.

```
using System;

class Program {
    static void Main(string[] args) {
        string a = "hello";
        Console.WriteLine(a[0]);
        Console.WriteLine(a[1]);
        Console.WriteLine(a[4]);
        Console.ReadKey();
    }
}
```

Рядкові константи можна задавати в двох варіантах:

- в подвійних лапках ;
- в подвійних лапках з `@` на початку.

В першому варіанті, використовуються `Escape` – послідовності, а в

другому ні. Тому другий варіант зручний, щоб вказувати шлях до файлу.

Таблиця 1. Escape – послідовності

Escape послідовність	Опис
\a	Дзвоник
\b	Повернення на одну позицію
\f	Перехід на нову сторінку
\n	Перехід на новий рядок
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\0	null
\'	Символ - '
\"	Символ – "
\\	Символ - \
\u0000 або \x0000	Представляє знак Юнікоду де dddd – шістнадцяткове число

Наприклад:

```
using System;
class Program {
    static void Main(string[] args) {
        Console.WriteLine("\u0041BC\n\\");
        Console.WriteLine(@"C:\Temp\file1.txt");
        Console.ReadKey();
    }
}
```

Виведе на екран:

```
ABC
\
C:\Temp\file1.txt
```

Таблиця 2. Основні властивості та поля класу **String**

Назва	Опис
Empty	Статичне поле доступне тільки для читання, повертає пустий рядок
Length	Властивість, що повертає довжину рядка

Таблиця 3 – Основні методи класу **String**

Назва	Опис
Compare	Статичний метод для порівняння рядків
Contains	Перевіряє, чи міститься підрядок у рядку
Copy	Статичний метод. Створює новий об'єкт, що є копією даного рядка
CopyTo	Копіює заданий діапазон символів в з рядка в масив символів
EndsWith	Перевіряє, чи завершується рядок заданим підрядком
Format	Статичний метод. Дозволяє створити новий рядок зі значень різних виразів за допомогою форматування
IndexOf	Повертає індекс входження підрядка або набору символів в рядок
IndexOfAny	Повертає індекс входження будь-якого з набору символів в рядок
Insert	Вставляє підрядок в рядок
IsNullOrEmpty	Статичний метод. Перевіряє чи є рядок пустим або null
Join	Статичний метод. Створює рядок з масиву рядків використовуючи заданий роздільник
LastIndexOf	Повертає індекс останнього входження підрядка або набору символів в рядок
LastIndexOfAny	Повертає індекс останнього входження будь-якого з набору символів в рядок
Remove	Видаляє з рядка задану кількість символів
Replace	Замінює входження символу або підрядка новим символом або підрядком
Split	Створює з рядку масив підрядків за вказаним набором символів роздільників
StartsWith	Перевіряє, чи починається рядок з заданого підрядка
Substring	Створює новий рядок з підрядка поточного рядку
ToCharArray	Створює масив символів з підрядка
ToLower	Переводить рядок до нижнього регістру
ToUpper	Переводить рядок до верхнього регістру
Trim	Видаляє всі входження вказаних символів на початку та в кінці рядку
TrimEnd	Видаляє всі входження вказаних символів в кінці рядку
TrimStart	Видаляє всі входження вказаних символів на початку рядку

## 2. Операції з рядками

### Конкатенація

Конкатенація рядків або об'єднання може проводитися як за допомогою операції `+`, так і за допомогою методу `Concat`:

```
string s1 = "hello";
string s2 = "world";
string s3 = s1 + " " + s2; // результат: строка "hello world"
string s4 = String.Concat(s3, "!!!"); // результат: строка "hello world!!!"

Console.WriteLine(s4);
```

Для об'єднання рядків також може використовуватися метод `Join`:

```
string s5 = "apple";
string s6 = "a day";
string s7 = "keeps";
string s8 = "a doctor";
string s9 = "away";
string[] values = new string[] { s5, s6, s7, s8, s9 };

String s10 = String.Join(" ", values);
// результат: строка "apple a day keeps a doctor away"
```

### Порівняння рядків

Для порівняння рядків застосовується статичний метод `Compare`:

```
string s1 = "hello";
string s2 = "world";

int result = String.Compare(s1, s2);
if (result < 0)
{
    Console.WriteLine("Строка s1 перед строкой s2");
}
else if (result > 0)
{
    Console.WriteLine("Строка s1 стоит после строки s2");
}
else
{
    Console.WriteLine("Строки s1 и s2 идентичны");
}
// результатом будет "Строка s1 перед строкой s2"
```

Дана версія методу `Compare` приймає два рядки і повертає число. Якщо перший рядок за алфавітом стоїть вище другий, то повертається число менше нуля. В іншому випадку повертається число більше нуля. І третій випадок - якщо

рядки рівні, то повертається число 0. В даному випадку так як символ **h** за алфавітом стоїть вище символу **w**, то і перший рядок буде стояти вище.

### Пошук в рядку

За допомогою методу `IndexOf` можемо визначити індекс першого входження окремого символу або підрядка в рядку:

```
string s1 = "hello world";
char ch = 'o';
int indexOfChar = s1.IndexOf(ch); // равно 4
Console.WriteLine(indexOfChar);

string subString = "wor";
int indexOfSubstring = s1.IndexOf(subString); // равно 6
Console.WriteLine(indexOfSubstring);
```

Подібним чином діє метод `LastIndexOf`, тільки знаходить індекс останнього входження символу або підрядка в рядок.

Ще одна група методів дозволяє дізнатися починається або закінчується рядок на певний підрядок. Для цього призначені методи `StartsWith` і `EndsWith`. Наприклад, у нас є завдання видалити з папки всі файли з розширенням exe:

```
string path = @"C:\SomeDir";

string[] files = Directory.GetFiles(path);

for (int i = 0; i < files.Length; i++)
{
    if(files[i].EndsWith(".exe"))
        File.Delete(files[i]);
}
```

### Поділ рядків

За допомогою функції `Split` можемо розділити рядок на масив підрядків. Як параметр функція `Split` приймає масив символів або рядків, які і будуть служити роздільниками. Наприклад, підрахуємо кількість слів у терміні, розділивши її з пробільних символів:

```
string text = "И поэтому все так произошло";

string[] words = text.Split(new char[] { ' ' });

foreach (string s in words)
{
    Console.WriteLine(s);
}
```

Це не кращий спосіб поділу по пробілу, так як у вхідному рядку могло б бути кілька підряд пробілів і в підсумковий масив також би потрапили пропуски, тому краще використовувати іншу версію методу:

```
string[] words = text.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
```

### Обрізання рядку

Для обрізання початкових або кінцевих символів використовується функція `Trim`:

```
string text = " hello world ";

text = text.Trim(); // результат "hello world"
text = text.Trim(new char[] { 'd', 'h' }); // результат "ello worl"
```

Функція `Trim` без параметрів обрізає початкові і кінцеві пробіли і повертає обрізану рядок. Щоб явно вказати, які початкові та кінцеві символи слід обрізати, ми можемо передати в функцію масив цих символів.

Ця функція має часткові аналоги: функція `TrimStart` обрізає початкові символи, а функція `TrimEnd` обрізає кінцеві символи. Обрізати певну частину рядка дозволяє функція `Substring`:

```
string text = "Хороший день";
// обрезаем начиная с третьего символа
text = text.Substring(2);
// результат "роший день"
Console.WriteLine(text);
// обрезаем сначала до последних двух символов
text = text.Substring(0, text.Length - 2);
// результат "роший де"
Console.WriteLine(text);
```

Функція `Substring` також повертає обрізану рядок. Як параметр перша використана версія застосовує індекс, починаючи з якого треба обрізати рядок.

Друга версія застосовує два параметра - індекс початку обрізки і довжину вирізується частини рядка.

### Вставка

Для вставки одного рядка в іншу застосовується функція `Insert`:

```
string text = "Хороший день";  
string subString = "замечательный ";  
  
text = text.Insert(8, subString);  
Console.WriteLine(text);
```

Першим параметром в функції `Insert` є індекс, за яким треба вставляти підрядок, а другий параметр - власне підрядок.

### Видалення рядків

Видалити частину рядка допомагає метод `Remove`:

```
string text = "Хороший день";  
// индекс последнего символа  
int ind = text.Length - 1;  
// вырезаем последний символ  
text = text.Remove(ind);  
Console.WriteLine(text);  
  
// вырезаем первые два символа  
text = text.Remove(0, 2);
```

Перша версія методу `Remove` приймає індекс в рядку, починаючи з якого треба видалити всі символи. Друга версія приймає ще один параметр - скільки символів треба видалити.

### Заміна

Щоб замінити один символ або підрядок на іншу, застосовується метод `Replace`:

```
string text = "хороший день";  
  
text = text.Replace("хороший", "плохой");  
Console.WriteLine(text);  
  
text = text.Replace("о", "");  
Console.WriteLine(text);
```

У другому випадку застосування функції `Replace` рядок з одного символу "о" замінюється на порожній рядок, тобто фактично видаляється з



тексту. Подібним способом легко видаляти якийсь певний текст в рядках.

### Зміна регістра

Для приведення рядка до верхнього і нижнього регістру використовуються відповідно функції `ToUpper ()` і `ToLower ()`:

```
string hello = "Hello world!";

Console.WriteLine(hello.ToLower()); // hello world!
Console.WriteLine(hello.ToUpper()); // HELLO WORLD!
```

Якщо методів класу **String** не достатньо, для перетворень текстової інформації можна використовувати клас – **StringBuilder** (простір імен **System.Text**), що дозволяє змінювати окремі символи та має свій набір методів.

Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class Program {
    static void Main(string[] args) {
        StringBuilder sb = new StringBuilder("kyiv");
        sb[0] = Char.ToUpper(sb[0]);
        Console.WriteLine(sb.ToString());
        Console.ReadKey();
    }
}
```

Ця програма замінює першу літеру слова **kyiv** на заголовну, і виводить на екран наступне:

Kyiv

## 3 Форматування і інтерполяція рядків

При виведенні рядків в консолі за допомогою методу **Console.WriteLine** можемо застосовувати форматування замість конкатенації:

```

class Program
{
    static void Main(string[] args)
    {
        Person person = new Person { Name = "Tom", Age = 23 };

        Console.WriteLine("Имя: {0} Возраст: {1}", person.Name, person.Age);
        Console.Read();
    }
}

class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}

```

У рядку "Ім'я: {0} Вік: {1}" на місце {0} та {1} будуть вставлятися в порядку проходження person.Name та person.Age. Те ж саме ми можемо зробити за допомогою методу **String.Format**:

```

string output = String.Format("Имя: {0} Возраст: {1}", person.Name, person.Age);
Console.WriteLine(output);

```

У методі **Format** можуть використовуватися різні специфікатори, які дозволяють налаштувати вивід даних.

Всі формати:

<b>C / c</b>	Задає формат грошової одиниці, вказує кількість десяткових розрядів після коми
<b>D / d</b>	Цілочисельний формат, вказує мінімальну кількість цифр
<b>E / e</b>	Експоненціальне уявлення числа, вказує кількість десяткових розрядів після коми
<b>F / f</b>	Формат дробових чисел з фіксованою точкою, вказує кількість десяткових розрядів після коми
<b>G / g</b>	Задає коротший з двох форматів: F або E
<b>N / n</b>	Також задає формат дробових чисел з фіксованою точкою, визначає кількість розрядів після коми
<b>P / p</b>	Задає відображення знаку відсотків поруч з числом, вказує кількість десяткових розрядів після коми
<b>X / x</b>	Шістнадцятковий формат числа

## форматування валюти

```
double number = 23.7;
string result = String.Format("{0:C}", number);
Console.WriteLine(result); // $ 23.7
string result2 = String.Format("{0:C2}", number);
Console.WriteLine(result2); // $ 23.70
```

## форматування цілих чисел

```
int number = 23;
string result = String.Format("{0:d}", number);
Console.WriteLine(result); // 23
string result2 = String.Format("{0:d4}", number);
Console.WriteLine(result2); // 0023
```

## форматування дробових чисел

```
int number = 23;
string result = String.Format("{0:f}", number);
Console.WriteLine(result); // 23,00

double number2 = 45.08;
string result2 = String.Format("{0:f4}", number2);
Console.WriteLine(result2); // 45,0800

double number3 = 25.07;
string result3 = String.Format("{0:f1}", number3);
Console.WriteLine(result2); // 25,1
```

## формат відсотків

```
decimal number = 0.15345m;
Console.WriteLine("{0:P1}", number); // 15.3%
```

## настроюваний формат

Використовуючи знак #, можна налаштувати формат виведення. Наприклад, нам треба вивести деяке число в форматі телефону + x (xxx) xxx-xx-xx:

```
long number = 19876543210;
string result = String.Format("{0:+# (###) ###-##-##}", number);
Console.WriteLine(result); // +1 (987) 654-32-10
```

## метод ToString

Метод **ToString()** не тільки отримує рядковий опис об'єкта, а й може здійснювати форматування, як в методі **Format**:

```
long number = 19876543210;
Console.WriteLine(number.ToString("+# (###) ###-##-##")); // +1 (987) 654-32-10

double money = 24.8;
Console.WriteLine(money.ToString("C2")); // $ 24,80
```

## Інтерполяція рядків

Починаючи з версії мови C # 6.0 була додана така функціональність, як інтерполяція рядків. Ця функціональність покликана замінити форматування рядків. Знак долара перед рядком вказує, що буде здійснюватися інтерполяція рядків. У середині рядка знову ж використовуються плейсхолдери {...}, тільки всередині фігурних дужок вже можна безпосередньо писати ті вирази, які хочемо вивести. Інтерполяція по суті являє більш лаконічне форматування. При цьому всередині фігурних дужок можемо вказувати не тільки властивості, але і різні вирази мови C #:

```
int x = 8;
int y = 7;
string result = $"{x} + {y} = {x + y}";
Console.WriteLine(result); // 8 + 7 = 15
```

```
long number = 19876543210;
Console.WriteLine($"{number:+# ### ### ## ##}"); // +1 987 654 32 10
```

```
Person person = new Person { Name = "Tom", Age = 23 };

Console.WriteLine($"Имя: {person.Name}  Возраст: {person.Age}");
```

```
Console.WriteLine($"Имя: {person.Name, -5} Возраст: {person.Age}"); // пробелы после
Console.WriteLine($"Имя: {person.Name, 5} Возраст: {person.Age}"); // пробелы до
```