Синтаксис C#
(работа со строками)

Артём Трофимушкин

Строки в C#

Тип string представляет последовательность, состоящую из нуля или более символов в кодировке Юникод. string является псевдонимом для типа System.String в .NET

Сравнение строк:

- == (проверка на равенство)
- != (проверка на неравенство)
- Метод Equals (проверка на равенство с дополнительными настройками

```
string a = "test";
string b = "Test";
Console.WriteLine(a == b);      // false
Console.WriteLine(a.Equals(b)); // false

// but this will return true
Console.WriteLine(a.Equals(b, StringComparison.InvariantCultureIgnoreCase));
```

Escape-последовательности (\uXXXX)

Escape-последовательности и соответствующие им символы

- `\'` Одинарная кавычка
- `\"` Двойная кавычка
- `\\` Обратная косая черта
- `\0` Null
- `\a` Предупреждение
- `\b` Backspace
- `\f` Перевод страницы
- `\n` Новая строка
- `\r` Возврат каретки
- `\t` Горизонтальная табуляция
- `\u` Escape-последовательность Юникода, например \u0041 = "A"

```
var test = "a\tb\nc";
Console.WriteLine(test);
// a    b
// c
```

Буквальные строковые литералы (@)

Буквальные строки используются для удобства и читабельности, если текст строки содержит символы обратной косой черты, например в путях к файлам:

```
Console.WriteLine("c:\\path\\to\\file1.txt"); // c:\path\to\file1.txt
Console.WriteLine(@"c:\path\to\file2.txt");   // c:\path\to\file2.txt
```

Так как они сохраняют символы новой строки как часть текста строки, их можно использовать для инициализации многострочных строк.

```
    string multiline = @"Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.";
    Console.WriteLine(multiline);
```

Конкатенация строк

Объединение подразумевает добавление одной строки к концу другой. Вы можете сцеплять строки с помощью оператора +.

Следующий пример показывает использование сцепки для разделения длинного строкового литерала на строки меньшего размера, чтобы повысить удобочитаемость исходного кода. Эти части объединяются в одну строку во время компиляции. Количество строк не влияет на производительность во время выполнения:

```
string longText = "Lorem ipsum dolor sit amet, consectetur adipiscing "
  + "elit, sed do eiusmod tempor incididunt ut labore et dolore magna "
  + "aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco "
  + "laboris nisi ut aliquip ex ea commodo consequat.";

Console.WriteLine(longText);
```

Конкатенация строк

Объединение строк с переменными других типов также возможно с помощью оператора +. При объединении переменные других типов будут автоматически приведены к строке с помощью метода ToString().

```
int i = 15;
int j = 3;
string result1 = i + " divided by " + j + " equals to " + i / j;
Console.WriteLine(result1); // 15 divided by 3 equals to 5

DateTime now = DateTime.Now;
string result2 = String.Format("Now is {0:dd.MM.yyyy HH:mm}", now);
Console.WriteLine(result2); // Now is 05.02.2019 12:00
```

Конкатенация строк

Объединение строк с переменными других типов также возможно с помощью оператора +. При объединении переменные других типов будут автоматически приведены к строке с помощью метода ToString().

```
int i = 15;
int j = 3;
string result1 = i + " divided by " + j + " equals to " + i / j;
Console.WriteLine(result1); // 15 divided by 3 equals to 5

DateTime now = DateTime.Now;
string result2 = String.Format("Now is {0:dd.MM.yyyy HH:mm}", now);
Console.WriteLine(result2); // Now is 05.02.2019 12:00
```

Форматирование строк

Форматирование позволяет сформировать строку используя шаблон и объекты с возможностью указать формат. Для этого используется метод String.Format(...).

Вместо переменных значений используют шаблоны с порядковыми номерами в круглых скобках: {0}, {1}, и т.д.:

```
double i = 15;
double j = Math.PI;
string s = String.Format("{0} divided by {1} equals to {2}", i, j, i / j);
Console.WriteLine(s);

// Output:
// 15 divide by 3.14159265358979 equals to 4.77464829275686
```

Форматирование строк

Можно также указывать формат для некоторых переменных, например можно указать сколько выводить знаков после запятой или в каком формате указывать дату и время.

Формат указывается через двоеточие после индекса переменной: {0:#.###}, {1:dd.MM.yyyy HH:mm}, и т.д.:

```
double i = 15;
double j = Math.PI;
var s = String.Format("{0} divided by {1:#.##} equals to {2:#.##}", i, j, i/j);
Console.WriteLine(s); // 15 divided by 3.14 equals to 4.77

DateTime now = DateTime.Now;
string result = String.Format("Now is {0:dd.MM.yyyy HH:mm}", now);
Console.WriteLine(result); // Now is 29.01.2019 14:00
```

Все виды возможных форматов для различных типов данных можно найти здесь: https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/formatting-types

Интерполяция строк

Специальный знак $ в начале строки делает интерполированную строку.

При вычислении интерполированной строки в результирующую элементы заменяются строковыми представлениями результатов выражений.
Эта возможность доступна в C# 6 и более поздних версиях.

```
double i = 15;
double j = Math.PI;
var s = $"{i} divided by {j:#.##} equals to {i / j:#.##}";
Console.WriteLine(s); // 15 divided by 3.14 equals to 4.77

DateTime now = DateTime.Now;
string result = $"Now is {now:dd.MM.yyyy HH:mm}";
Console.WriteLine(result); // Now is 29.01.2019 14:00
```

Здесь также подходят все виды форматов доступных по URL:
https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/formatting-types

Самостоятельная работа

Написать приложение, которое будет запрашивать у пользователя 2 вещественных (дробных) числа, а затем будет выводит результат математических действий над ними. Строки вывода должны формироваться различными методами:

- перемножение – конкатенацией
- сложение – форматированием
- вычитание – интерполяцией

Проверку ошибок входных данных не делаем (будем рассчитывать на корректный ввод данных).

Пример выполнения с результатом

```
Enter two real numbers to multiply them:
12.6 /это ввод пользователя/
3.14 /это ввод пользователя/
12.6 * 3.14 = 39.564
12.6 + 3.14 = 15.74
12.6 - 3.14 = 9.46
```

Поиск по строкам

Содержит ли строка текст?

- Contains (содержит ли ...?)
- StartsWith (начинается ли с ...?)
- EndsWith (заканчивается ли на ...?)

```
var test = "test string";
Console.WriteLine(test.Contains(" "));       // true
Console.WriteLine(test.StartsWith("te"));    // true
Console.WriteLine(test.EndsWith("s"));       // false
```

Где искомый текст находится в строке?

- IndexOf (позиция первого символа найденной последовательности или -1)
- LastIndexOf (позиция последнего символа найденной последовательности или -1)

```
Console.WriteLine(test.IndexOf("s"));        // 2
Console.WriteLine(test.LastIndexOf("s"));    // 5
```

Модификация строк

Методы модификации строк

- Replace (найти и заменить с … на …)
- Substring (вырезать подстроку указанной длины начиная с указанной позиции)
- Split (разделить строку на массив подстрок используя символ-разделитель)
    - Split(' ', StringSplitOptions.RemoveEmptyEntries);

```
var test = "my test string";
Console.WriteLine(test.Replace("test", "best"));  // my best string
Console.WriteLine(test.Substring(8, 3));          // str

string[] words = test.Split(' ');
foreach (string word in words)
    Console.WriteLine(word);
// my
// test
// string
```

Модификация строк

Методы изменения регистра

- ToLower, ToLowerInvariant (привести буквенные символы к нижнему регистру)
- ToUpper, ToUpperInvariant (привести буквенные символы к верхнему регистру)

```
Console.Write("Enter cardholder name: ");
string cardholder = Console.ReadLine();

Console.WriteLine($"Valid cardholder name: {cardholder.ToUpper()}");

Console.WriteLine("Press any key to exit...");
Console.ReadKey();

// Enter cardholder name: Andrei Golyakov
// Valid cardholder name: ANDREI GOLYAKOV
// Press any key to exit...
```

Очистка строк и проверка на пустоту

Методы очистки строк

- Trim (удалить все начальные и конечные вхождения заданных символов или пробелы)
- TrimStart (удалить все начальные вхождения заданных символов или пробелы)
- TrimEnd (удалить все конечные вхождения заданных символов или пробелы)

```
var test = " \t my test string \t ";
Console.WriteLine(test.Trim());
Console.WriteLine(test.TrimStart());
Console.WriteLine(test.TrimEnd());
```

Методы проверки строк

- IsNullOrEmpty (строка не определена или равна string.Emtpy)
- IsNullOrWhiteSpace (строка не определена или содержит лишь знаки отступов)

```
Console.WriteLine(string.IsNullOrWhiteSpace("    \t    "));   // true
Console.WriteLine(string.IsNullOrEmpty(""));                  // true
Console.WriteLine(string.IsNullOrEmpty(null));                // true
```

Построение строк (string.Join)

Методы создания строк

- string.Join (склеить несколько элементов, используя указанную подстроку в промежутках)

```
var test = "my test string";

// split strings into string array
string[] words = test.Split(' ');

// rebuild string taking the first and the third words only
string result = string.Join("_", words[0], words[2]);

Console.WriteLine(result);
// my_string
```

Самостоятельная работа

Дана строка с “грязными” пробелами, например:

```
string text = "   lorem    ipsum    dolor      sit   amet  ";
```

Необходимо произвести над ней следующие операции:

1. “Очистить” исходную строку от лишних пробелов в начале, в конце строки, а также между словами, а также поднять регистр второго слова:
   - “ lorem ipsum dolor sit amet ” → “lorem IPSUM dolor sit amet”
2. Удалить из исходной строки последнее слово и пробелы перед ним:
   - “ lorem ipsum dolor sit amet ” → “ lorem ipsum dolor sit”

Результаты по каждому пункту вывести отдельной строкой. Пример:

```
lorem IPSUM dolor sit amet
   lorem    ipsum    dolor      sit
```

Самостоятельная работа (решение)

```
string text = "   lorem    ipsum    dolor      sit   amet  ";

// part 1:

string[] words = text.Split(' ', StringSplitOptions.RemoveEmptyEntries); // split string by words

words[1] = words[1].ToUpperInvariant();      // uppercase for the second word

Console.WriteLine(string.Join(' ', words));  // join words using single space char


// part 2:

string textClean = text.TrimEnd();                    // remove the spaces at the end of the line

int lastSpaceIndex = textClean.LastIndexOf(' ');      // looking for the position of the last space char

textClean = textClean.Substring(0, lastSpaceIndex);   // cutting string from 0 to lastSpaceIndex

textClean = textClean.TrimEnd();                      // clean up the tail

Console.WriteLine(textClean);                         // this is it :)
```

Построение строк (StringBuilder)

Класс StringBuilder позволяет эффективно с точки зрения памяти создавать и модифицировать длинные строки.

Располагается в области видимости System.Text. Основные члены класса:

- Append() – добавляет аргумент, при необходимости конвертированный в строку, к концу внутренней строчки
- AppendFormat() – добавляет шаблонизированную строку к концу внутренней строки, позволяет передать сразу шаблон форматирования и дополнительные параметры. Является просто сокращенной записью от Append(string.Format())
- Insert() – позволяет вставить в произвольное место внутренней строки переданную параметром строку или переменную, приведенную к строке.
- Remove() – позволяет удалить заданное количество символов внутренней строки начиная с любого места.
- Replace() – аналогично string.Replace – ищет и в случае нахождения изменяет искомую строку заданной.
- ToString() – строит и возвращает внутреннюю строку, когда создание завершено.
- Length - возвращает текущую длину внутренней строки.

Построение строк (StringBuilder)

```
// Create a StringBuilder that expects to hold 50 characters.
// Initialize the StringBuilder with "ABC".
StringBuilder sb = new StringBuilder("ABC", 50);

// Append three characters (D, E, and F) to the end of the StringBuilder.
sb.Append(new char[] { 'D', 'E', 'F' });

// Append a format string to the end of the StringBuilder.
sb.AppendFormat("GHI{0}{1}", 'J', 'k');

// Display the number of characters in the StringBuilder and its string.
Console.WriteLine("{0} chars: {1}", sb.Length, sb.ToString());

// Insert a string at the beginning of the StringBuilder.
sb.Insert(0, "Alphabet: ");
// Replace all lowercase k's with uppercase K's.
sb.Replace('k', 'K');

// Display the number of characters in the StringBuilder and its string.
Console.WriteLine("{0} chars: {1}", sb.Length, sb.ToString());
```

Домашнее задание 1

Написать консольное приложение, которое запрашивает строку и выводит количество слов, начинающихся на букву А.
Программа должна спрашивать исходную строку до тех пор, пока пользователь не введёт хотя бы 2 слова.

Пример работы программы:

```
> Введите строку из нескольких слов:
> тест /это ввод пользователя/
> Слишком мало слов :( Попробуйте ещё раз:
> Антон купил арбуз. Алый мак растет среди травы. /это ввод пользователя/
> Количество слов, начинающихся с буквы 'А': 3.
> Нажмите любую клавишу для выхода...
```

Не забывать обрабатывать все предсказуемые исключения.

Домашнее задание 2

Написать консольное приложение, которое запрашивает строку, а затем выводит все буквы приведенные к нижнему регистру в обратном порядке. Программа должна спрашивать исходную строку до тех пор, пока пользователь не введет строку, содержащую печатные символы.

Пример работы программы:

```
> Введите непустую строку:
>       /это ввод пользователя/
> Вы ввели пустую строку :( Попробуйте ещё раз:
> Не до логики, голоден /это ввод пользователя/
> недолог ,икигол од ен
> Нажмите любую клавишу для выхода...
```

Не забывать обрабатывать все предсказуемые исключения.

Спасибо за внимание.