**Регулярные выражения в C#**

***Регулярные выражения*** — это часть небольшой технологической области, невероятно широко используемой в огромном диапазоне программ. Регулярные выражения можно представить себе как мини-язык программирования, имеющий одно специфическое назначение: находить подстроки в больших строковых выражениях.

Это не новая технология, изначально она появилась в среде **UNIX** и обычно используется в языке программирования Perl. Разработчики из Microsoft перенесли ее в Windows, где до недавнего времени эта технология применялась в основном со сценарными языками. Однако теперь регулярные выражения поддерживаются множеством классов .NET из пространства имен *System.Text.RegularExpressions*. Случаи применения регулярных выражений можно встретить во многих частях среды .NET Framework. В частности, вы найдете их в серверных элементах управления проверкой ASP.NET.

**Введение в регулярные выражения**

Язык регулярных выражений предназначен специально для обработки строк. Он включает два средства:

1. Набор управляющих кодов для идентификации специфических типов символов
2. Система для группирования частей подстрок и промежуточных результатов таких действий

С помощью регулярных выражений можно выполнять достаточно сложные и высокоуровневые действия над строками:

* Идентифицировать (и возможно, помечать к удалению) все повторяющиеся слова в строке
* Сделать заглавными первые буквы всех слов
* Преобразовать первые буквы всех слов длиннее трех символов в заглавные
* Обеспечить правильную капитализацию предложений
* Выделить различные элементы в URI (например, имея http://www.professorweb.ru, выделить протокол, имя компьютера, имя файла и т.д.)

Главным преимуществом регулярных выражений является использование ***метасимволов*** — специальные символы, задающие команды, а также управляющие последовательности, которые работают подобно управляющим последовательностям C#. Это символы, предваренные знаком обратного слеша (\) и имеющие специальное назначение.

В следующей таблице специальные метасимволы регулярных выражений C# сгруппированы по смыслу:

|  |  |  |  |
| --- | --- | --- | --- |
| *Метасимволы, используемые в регулярных выражениях C#* | | | |
| **Символ** | **Значение** | **Пример** | **Соответствует** |
| Классы символов | | | |
| [...] | Любой из символов, указанных в скобках | [a-z] | В исходной строке может быть любой символ английского алфавита в нижнем регистре |
| [^...] | Любой из символов, не указанных в скобках | [^0-9] | В исходной строке может быть любой символ кроме цифр |
| . | Любой символ, кроме перевода строки или другого разделителя Unicode-строки |  |  |
| \w | Любой текстовый символ, не являющийся пробелом, символом табуляции и т.п. |  |  |
| \W | Любой символ, не являющийся текстовым символом |  |  |
| \s | Любой пробельный символ из набора Unicode |  |  |
| \S | Любой непробельный символ из набора Unicode. Обратите внимание, что символы \w и \S - это не одно и то же |  |  |
| \d | Любые ASCII-цифры. Эквивалентно [0-9] |  |  |
| \D | Любой символ, отличный от ASCII-цифр. Эквивалентно [^0-9] |  |  |
| **Символы повторения** | | | |
| {n,m} | Соответствует предшествующему шаблону, повторенному не менее n и не более m раз | s{2,4} | "Press", "ssl", "progressss" |
| {n,} | Соответствует предшествующему шаблону, повторенному n или более раз | s{1,} | "ssl" |
| {n} | Соответствует в точности n экземплярам предшествующего шаблона | s{2} | "Press", "ssl", но не "progressss" |
| ? | Соответствует нулю или одному экземпляру предшествующего шаблона; предшествующий шаблон является необязательным | Эквивалентно {0,1} |  |
| + | Соответствует одному или более экземплярам предшествующего шаблона | Эквивалентно {1,} |  |
| \* | Соответствует нулю или более экземплярам предшествующего шаблона | Эквивалентно {0,} |  |
| **Символы регулярных выражений выбора** | | | |
| | | Соответствует либо подвыражению слева, либо подвыражению справа (аналог логической операции ИЛИ). |  |  |
| (...) | Группировка. Группирует элементы в единое целое, которое может использоваться с символами \*, +, ?, | и т.п. Также запоминает символы, соответствующие этой группе для использования в последующих ссылках. |  |  |
| (?:...) | Только группировка. Группирует элементы в единое целое, но не запоминает символы, соответствующие этой группе. |  |  |
| **Якорные символы регулярных выражений** | | | |
| ^ | Соответствует началу строкового выражения или началу строки при многострочном поиске. | ^Hello | "Hello, world", но не "Ok, Hello world" т.к. в этой строке слово "Hello" находится не в начале |
| $ | Соответствует концу строкового выражения или концу строки при многострочном поиске. | Hello$ | "World, Hello" |
| \b | Соответствует границе слова, т.е. соответствует позиции между символом \w и символом \W или между символом \w и началом или концом строки. | \b(my)\b | В строке "Hello my world" выберет слово "my" |
| \B | Соответствует позиции, не являющейся границей слов. | \B(ld)\b | Соответствие найдется в слове "World", но не в слове "ld" |

Это только небольшая часть элементов. Более подробное описание синтаксиса регулярных выражений можно найти на msdn в статье [Элементы языка регулярных выражений — краткий справочник](http://msdn.microsoft.com/ru-ru/library/az24scfc%28v=vs.110%29.aspx).

Теперь посмотрим на некоторые примеры использования. Возьмем первый пример с скороговоркой "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа" и найдем в ней все слова, где встречается корень "губ":

|  |  |
| --- | --- |
| 1  2 | string s = "Бык тупогуб, тупогубенький бычок, у быка губа бела была тупа";  Regex regex = new Regex(@"\w\*губ\w\*"); |

Так как выражение \w\* соответствует любой последовательности алфавитно-цифровых символов любой длины, то данное выражение найдет все слова, содержащие корень "губ".

Второй простенький пример - нахождение телефонного номера в формате 111-111-1111:

|  |  |
| --- | --- |
|  | string s = "456-435-2318";  Regex regex = new Regex(@"\d{3}-\d{3}-\d{4}"); |

Если мы точно знаем, сколько определенных символов должно быть, то мы можем явным образом указать их количество в фигурных скобках:\d{3} - то есть в данном случае три цифры.

Мы можем не только задать поиск по определенным типам символов - пробелы, цифры, но и задать конкретные символы, которые должны входить в регулярное выражение. Например, перепишем пример с номером телефона и явно укажем, какие символы там должны быть:

|  |  |
| --- | --- |
| 1  2 | string s = "456-435-2318";  Regex regex = new Regex("[0-9]{3}-[0-9]{3}-[0-9]{4}"); |

В квадратных скобках задается диапазон символов, которые должны в данном месте встречаться. В итоге данный и предыдущий шаблоны телефонного номера будут эквивалентны.

Также можно задать диапазон для алфавитных символов: Regex regex = new Regex("[a-v]{5}"); - данное выражение будет соответствовать любому сочетанию пяти символов, в котором все символы находятся в диапазоне от a до v.

Можно также указать отдельные значения: Regex regex = new Regex(@"[2]\*-[0-9]{3}-\d{4}");. Это выражение будет соответствовать, например, такому номеру телефона "222-222-2222" (так как первые числа двойки)

С помощью операции | можно задать альтернативные символы: Regex regex = new Regex(@"[2|3]{3}-[0-9]{3}-\d{4}");. То есть первые три цифры могут содержать только двойки или тройки. Такой шаблон будет соответствовать, например, строкам "222-222-2222" и "323-435-2318". А вот строка "235-435-2318" уже не подпадает под шаблон, так как одной из трех первых цифр является цифра 5.

Итак, у нас такие символы, как \*, + и ряд других используются в качестве специальных символов. И возникает вопрос, а что делать, если у нам надо найти, строки, где содержится точка, звездочка или какой-то другой специальный символ? В этом случае нам надо просто экранировать эти символы слешем:

|  |  |
| --- | --- |
| 1  2 | Regex regex = new Regex(@"[2|3]{3}\.[0-9]{3}\.\d{4}");  // этому выражению будет соответствовать строка "222.222.2222" |

**Использование регулярных выражений в C#**

Безуcловно, задачу поиска и замены подстроки в строке можно решить на C# с использованием различных методов [System.String](https://professorweb.ru/my/csharp/charp_theory/level4/4_7.php) и [System.Text.StringBuilder](https://professorweb.ru/my/csharp/charp_theory/level4/4_8.php). Однако в некоторых случаях это потребует написания большого объема кода C#. Если вы используете регулярные выражения, то весь этот код сокращается буквально до нескольких строк. По сути, вы создаете экземпляр объекта **RegEx**, передаете ему строку для обработки, а также само регулярное выражение (строку, включающую инструкции на языке регулярных выражений) — и все готово.

В следующей таблице показана часть информации о перечислении *RegexOptions*, экземпляр которого можно передать конструктору класса RegEx:

|  |  |
| --- | --- |
| *Структура перечисления RegexOptions* | |
| **Член** | **Описание** |
| CultureInvariant | Предписывает игнорировать национальные установки строки |
| ExplicitCapture | Модифицирует способ поиска соответствия, обеспечивая только буквальное соответствие |
| IgnoreCase | Игнорирует регистр символов во входной строке |
| IgnorePatternWhitespace | Удаляет из строки не защищенные управляющими символами пробелы и разрешает комментарии, начинающиеся со знака фунта или хеша |
| Multiline | Изменяет значение символов ^ и $ так, что они применяются к началу и концу каждой строки, а не только к началу и концу всего входного текста |
| RightToLeft | Предписывает читать входную строку справа налево вместо направления по умолчанию — слева направо (что удобно для некоторых азиатских и других языков, которые читаются в таком направлении) |
| Singleline | Специфицирует однострочный режим, в котором точка (.) символизирует соответствие любому символу |

После создания шаблона регулярного выражения с ним можно осуществить различные действия, в зависимости от того, что вам необходимо. Можно просто проверить, существует ли текст, соответствующий шаблону, в исходной строке. Для этого нужно использовать метод **IsMatch()**, который возвращает логическое значение:

**using** System;

**using** System.Text.RegularExpressions;

**class** **Example**

{

**static** **void** **Main**()

{

*// Массив тестируемых строк*

**string**[] test = {

"Wuck World", "Hello world", "My wonderful world"

};

*// Проверим, содержится ли в исходных строках слово World*

*// при этом мы не укажем опции RegexOption*

Regex regex = **new** Regex("World");

Console.WriteLine("Регистрозависимый поиск: ");

**foreach** (**string** str **in** test)

{

**if** (regex.IsMatch(str))

Console.WriteLine("В исходной строке: \"{0}\" есть совпадения!", str);

}

Console.WriteLine();

*// Теперь укажем поиск, не зависимый от регистра*

regex = **new** Regex("World", RegexOptions.IgnoreCase);

Console.WriteLine("РегистроНЕзависимый поиск: ");

**foreach** (**string** str **in** test)

{

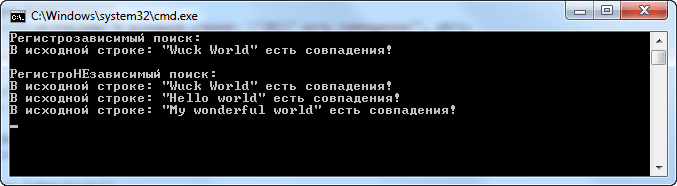
**if** (regex.IsMatch(str))

Console.WriteLine("В исходной строке: \"{0}\" есть совпадения!", str);

}

}

}



Если нужно вернуть найденное соответствие из исходной строки, то можно воспользоваться методом **Match()**, который возвращает объект класса *Match*, содержащий сведения о первой подстроке, которая сопоставлена шаблону регулярного выражения. В этом классе имеется свойство Success, которое возвращает значение true, если найдено следующее совпадение, которое можно получить с помощью вызова метода Match.*NextMatch()*. Эти вызовы метода можно продолжать пока свойство Match.Success не вернет значение false. Например:

**using** System;

**using** System.Text.RegularExpressions;

**class** **Example**

{

**static** **void** **Main**()

{

*// Допустим в исходной строке нужно найти все числа,*

*// соответствующие стоимости продукта*

**string** input = "Добро пожаловать в наш магазин, вот наши цены: " +

"1 кг. яблок - 20 руб. " +

"2 кг. апельсинов - 30 руб. " +

"0.5 кг. орехов - 50 руб.";

**string** pattern = @"\b(\d+\W?руб)";

Regex regex = **new** Regex(pattern);

*// Получаем совпадения в экземпляре класса Match*

Match match = regex.Match(input);

*// отображаем все совпадения*

**while** (match.Success)

{

*// Т.к. мы выделили в шаблоне одну группу (одни круглые скобки),*

*// ссылаемся на найденное значение через свойство Groups класса Match*

Console.WriteLine(match.Groups[1].Value);

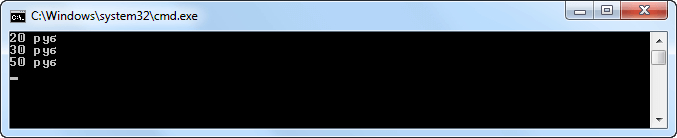
*// Переходим к следующему совпадению*

match = match.NextMatch();

}

}

}



Извлечь все совпадения можно и более простым способом, используя метод *Regex.Matches()*, который возвращает объект класса **MatchCollection**, который, в свою очередь, содержит сведения обо всех совпадениях, которые обработчик регулярных выражений находит во входной строке. Например, предыдущий пример может быть переписан для вызова метода Matches вместо метода Match и метода NextMatch:

**using** System;

**using** System.Text.RegularExpressions;

**class** **Example**

{

**static** **void** **Main**()

{

*// Допустим в исходной строке нужно найти все числа,*

*// соответствующие стоимости продукта*

**string** input = "Добро пожаловать в наш магазин, вот наши цены: " +

"1 кг. яблок - 20 руб. " +

"2 кг. апельсинов - 30 руб. " +

"0.5 кг. орехов - 50 руб.";

**string** pattern = @"\b(\d+\W?руб)";

Regex regex = **new** Regex(pattern);

*// Достигаем того же результата что и в предыдущем примере,*

*// используя метод Regex.Matches() возвращающий MatchCollection*

**foreach** (Match match **in** regex.Matches(input))

{

Console.WriteLine(match.Groups[1].Value);

}

}

}

Наконец, можно не просто извлекать совпадения в исходной строке, но и заменять их на собственные значения. Для этого используется метод *Regex.Replace()*. В качестве замены методу Replace() можно передавать как строку, так и шаблон замены. В следующей таблице показано как формируются метасимволы для замены:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Метасимволы замены в регулярных выражениях C#* | | | | |
| **Символ** | **Описание** | **Пример шаблона** | **Пример шаблона замены** | **Результат (входная -> результирующая строки)** |
| $ number | Замещает часть строки, соответствующую группе number | \b(\w+)(\s)(\w+)\b | $3$2$1 | "один два" -> "два один" |
| $$ | Подставляет литерал "$" | \b(\d+)\s?USD | $$$1 | "103 USD" -> "$103" |
| $& | Замещает копией полного соответствия | (\$\*(\d\*(\.+\d+)?){1}) | \*\*$& | "$1.30" -> "\*\*$1.30\*\*" |
| $` | Замещает весь текст входной строки до соответствия | B+ | $` | "AABBCC" -> "AAAACC" |
| $' | Замещает весь текст входной строки после соответствия | B+ | $' | "AABBCC" -> "AACCCC" |
| $+ | Замещает последнюю захваченную группу | B+(C+) | $+ | "AABBCCDD" -> "AACCDD" |
| $\_ | Замещает всю входную строку | B+ | $\_ | "AABBCC" -> "AAAABBCCCC" |

Давайте рассмотрим метод Regex.Replace() на примере:

**using** System;

**using** System.Text.RegularExpressions;

**class** **Example**

{

**static** **void** **Main**()

{

*// Допустим в исходной строке нужно заменить "руб." на "$",*

*// а стоимость переместить после знака $*

**string** input = "Добро пожаловать в наш магазин, вот наши цены: \n" +

"\t 1 кг. яблок - 20 руб. \n" +

"\t 2 кг. апельсинов - 30 руб. \n" +

"\t 0.5 кг. орехов - 50 руб. \n";

Console.WriteLine("Исходная строка:\n {0}", input);

*// В шаблоне используются 2 группы*

**string** pattern = @"\b(\d+)\W?(руб.)";

*// Строка замены "руб." на "$"*

**string** replacement1 = "$$$1"; *// Перед первой группой ставится знак $,*

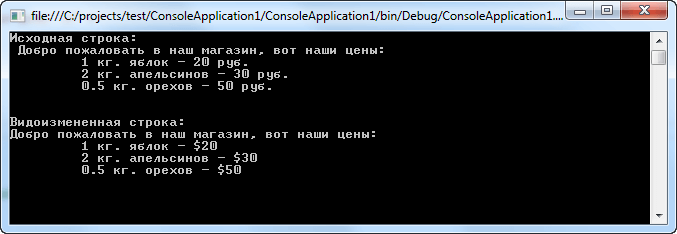
*// вторая группа удаляется без замены*

input = Regex.Replace(input, pattern, replacement1);

Console.WriteLine("\nВидоизмененная строка: \n" +input);

}

}



Для закрепления темы давайте рассмотрим еще один пример использования регулярных выражений, где будем искать в исходном тексте слово «сериализация» и его однокоренные слова, при этом выделяя в консоли их другим цветом:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Text.RegularExpressions;

**namespace** **ConsoleApplication1**

{

**class** **Program**

{

**static** **void** **Main**(**string**[] args)

{

**string** myText = @"Сериализация представляет собой процесс сохранения объекта на диске.

В другой части приложения или даже в совершенно отдельном приложении может производиться

десериализация объекта, возвращающая его в состояние, в котором он пребывал до сериализации.";

**const** **string** myReg = "со";

MatchCollection myMatch = Regex.Matches(myText,myReg);

Console.WriteLine("Все вхождения строки \"{0}\" в исходной строке: ",myReg);

**foreach** (Match i **in** myMatch)

Console.Write("\t"+i.Index);

*// Усложним шаблон регулярного выражения*

*// введя в него специальные метасимволы*

**const** **string** myReg1 = @"\b[с,д]\S\*ериализац\S\*";

MatchCollection match1 = Regex.Matches(myText,myReg1,RegexOptions.IgnoreCase);

findMyText(myText,match1);

Console.ReadLine();

}

**static** **void** **findMyText**(**string** text, MatchCollection myMatch)

{

Console.WriteLine("\n\nИсходная строка:\n\n{0}\n\nВидоизмененная строка:\n",text);

*// Реализуем выделение ключевых слов в консоли другим цветом*

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

{

**foreach** (Match m **in** myMatch)

{

**if** ((i >= m.Index) && (i < m.Index+m.Length))

{

Console.BackgroundColor = ConsoleColor.Green;

Console.ForegroundColor = ConsoleColor.Black;

**break**;

}

**else**

{

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.White;

}

}

Console.Write(text[i]);

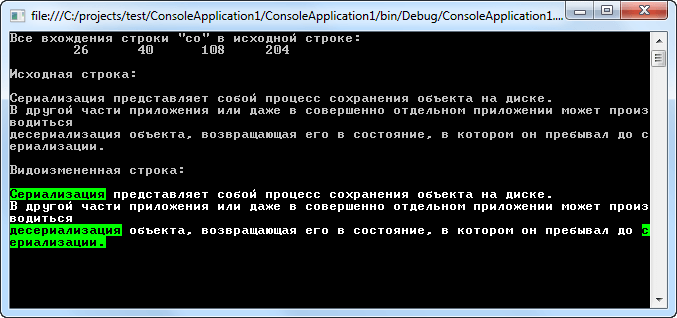
}

}

}

}

Результат работы данной программы:



Для проверки гибкости работы регулярных выражений, подставьте в исходный текст еще несколько слов «сериализация», вы увидите, что они будут автоматически выделены зеленым цветом в консоли.

**Лабораторная работа №7**

*Вариант 1*

1. Используя регулярные выражения, написать программу, которая проверит ввел ли пользователь телефон в формате +375(код)номер. Например, +375(44)9996699. Учтите, что 32 и т.д. не правильный мобильный код для нашей страны.
2. Составить регулярное выражение для поиска в тексте повторяющихся слов, расположенных подряд и разделенных произвольным количеством пробелов, независимо от регистра. Вывести их на экран консоли.

*Вариант 2*

1. Используя регулярные выражения, написать программу, которая проверит ввел ли пользователь e-mail адрес в соответствующем формате.
2. Задан текст с некоторыми словами, заключенными в фигурные скобки. Используя регулярные выражения, создать программу для получения всех подстрок между круглыми скобками.

*Вариант 3*

1. Используя регулярные выражения, написать программу, которая следующую задачу. Время имеет формат часы:минуты. И часы, и минуты состоят из двух цифр, пример: 09:00. Напишите регулярное выражение для поиска времени в строке: «Завтрак в 09:00». Учтите, что «37:98» – некорректное время.
2. Задан текст, в котором находятся номера телефонов в формате ххх-хх-хх. Используя регулярные выражения, найти и вывести эти номера.

*Вариант 4*

1. Используя регулярные выражения, написать программу, которая проверит ввел ли пользователь телефон в формате 8-(код)номер. Например, 8-(29)9996699. Учтите, что 32 и т.д. не правильный мобильный код для нашей страны.
2. Текст состоит из слов, чисел и знаков препинания. Составить регулярное выражение для поискав нем слов. Вывести их на экран консоли.

*Вариант 5*

1. Используя регулярные выражения, написать программу, которая проверит ввел ли пользователь IP-адрес. Адрес состоит из четырех групп цифр, разделенных точками. Каждая группа может включать от одной до трех цифр. Примеры: 212.46.197.69, 212.194.5.106.
2. Текст состоит из слов, чисел и знаков препинания. Составить регулярное выражение для поискав нем чисел. Вывести их на экран консоли.

*Вариант 6*

1. Используя регулярные выражения, написать программу, которая проверит ввел ли пользователь HTML-цвет. Цвет задается в формате #ABCDEF, то есть # и содержит затем 6 шестнадцатеричных символов.
2. Задан текст, который состоит из слов, чисел и знаков препинания. Заменить каждое вхождение слова «кот» на слово «котик». Вывести результат на экран консоли.

*Вариант 7*

1. Используя регулярные выражения, написать программу, которая решает следующую задачу. Арифметическое выражение состоит из двух чисел и операции между ними, например: 1 + 2, 1.2 \*3.4, -3/ -6, -2-2. Список операций: «+», «-», «\*» и «/». Также могут присутствовать пробелы вокруг оператора и чисел. Напишите регулярное выражение, которое найдёт как всё арифметическое действие, так и два числа-участника операции.
2. Задан текст, который состоит из слов, чисел и знаков препинания. Заменить каждое вхождение слова «лаба» на пробел. Вывести результат на экран консоли.

*Вариант 8*

1. Используя регулярные выражения, написать программу, которая решает задачу проверки имени заливаемого графического объекта на хостинг.
2. Используя регулярные выражения, написать программу, которая удаляет из строки все, что находится после символа #. Вывести результат на экран консоли.

*Вариант 9*

1. Используя регулярные выражения, написать программу, которая решает задачу проверки верно ли пользователь ввел адрес сайта.
2. Задан текст, в котором записаны ФИО в формате (Иванов П.П.). Используя регулярные выражения, вывести все ФИО в таком формате.

*Вариант 10*

1. Используя регулярные выражения, написать программу, которая решает задачу проверки верно ли пользователь ввел адрес id сайта Вконтакте (в старом формате, например, id78195751).
2. Найти в тексте, состоящем из слов, цифр и знаков препинания, все слова, длина которых больше 7 символов. Вывести результат на экран консоли.