**Общие сведения о регулярных выражениях**

Регулярные выражения предоставляют мощный, гибкий и эффективный способ обработки текста. Комплексная нотация сопоставления шаблонов регулярных выражений позволяет быстро анализировать большие объемы текста в следующих целях:

* валидация данных;
* сбор данных (особенно веб-скрапинг, поиск страниц, содержащих определённый набор слов в определённом порядке);
* обработка данных (преобразование сырых данных в нужный формат);
* парсинг (например, достать все GET параметры из URL или текст внутри скобок);
* замена строк по шаблону;
* подсветка синтаксиса, переименование файла, анализ пакетов и многие другие задачи, где нужно работать со строками.

Библиотеки для использования регулярных выражений, как правило, встроены в большинство современных языков программирования, однако даже без них многие текстовые редакторы (Notepad++, Visual Studio) позволяют производить поиск и обработку текста с помощью регулярных выражений.

Регулярное выражение — это описание шаблона символов для поиска в строке. Самый простой шаблон, который мы можем описать — это непосредственно та строка символов, которую необходимо найти. Например, если нужно проверить, содержит ли строка слово «world», мы можем использовать эту же строку в регулярном выражении. В более сложных случаях нужно использовать специальные символы, которые можно условно разделить на 3 группы:

* метасимволы, отвечающие за класс символа (таблица 1) – это специальные обозначения, которые позволяют заменить собой несколько символов обычного алфавита. Например, регулярное выражение, проверяющее наличие в строке любой из цифр, будет описываться символом \d ;
* метасимволы, указывающие кратность символа или группы символов (таблица 2) позволяют задать, сколько раз должен встретиться символ. Например, если мы хотим найти все трехзначные цифры, регулярное выражение будет иметь вид \d{3}
* метасимволы, отвечающие за группировку символов, частей шаблона (таблица 3) используются в случаях, когда шаблон регулярного выражения может и должен быть разбит на отдельные составляющие, которые могут потребовать раздельной обработки (например поиск повторений заданной группы символов, извлечение значений из отдельных групп одного выражения).

*Таблица 1 – метасимволы, отвечающие за задание класса символа*

|  |  |
| --- | --- |
| **Символ** | **Значение** |
| [ ] | Любой из символов, указанных в скобках.  Пример:  [a-d] – регулярное выражение задает любой из символов в диапазоне с ‘a’ по ‘d’ (включительно)  [abc] – любой из символов: 'a’, ‘b’, ‘c’ |
| [^ ] | Любой из символов, не указанных в скобках  Пример:  [^0-5] – любой символ кроме цифр от 0 до 5 (включительно)  [!$qwe] – любой символ кроме ‘!’, ‘$’, ‘q’, ‘w’, ‘e’ |
| . | Любой символ, кроме символов-разделителей. |
| \w | Любой текстовый символ, не являющийся пробелом, символом табуляции и т.п. |
| \W | Любой символ, не являющийся текстовым символом. |
| \s | Пробельный символ. |
| \S | Любой непробельный символ из набора Unicode. Обратите внимание, что символы \w и \S — это не одно и то же. |
| \d | Цифровой символ, аналог [0-9] |
| \D | Любой символ, отличный от ASCII-цифр. Эквивалентно [^0-9] |
| ^ | Соответствует началу строкового выражения или началу строки при многострочном поиске.  Пример:  ^Hello – соответствует строке "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.  Пример:  Hello, World$ - соответствует строкам, заканчивающимся на “World” |
| \b | Соответствует границе слова, т.е. соответствует позиции между символом \w и символом \W или между символом \w и началом или концом строки.  Пример:  \b(my)\b - в строке "Hello my world" выберет слово "my" (Скобки – оператор группировки, подробнее см. в таблице 3). |
| \B | Соответствует позиции, не являющейся границей слов.  Пример:  \B(ld)\b - соответствие найдется в слове "World", но не в слове "ld" |

*Таблица 2 – метасимволы, отвечающие за кратность повторения символа*

|  |  |
| --- | --- |
| **Символ** | **Значение** |
| {n,m} | Соответствует предшествующему шаблону, повторенному не менее n и не более m раз  Пример:  Приве{1,3}т – соответствует строкам «Привет», «Привеет», «Привееет»  \d{2,5} – соответствует числам, содержащим от 2 до 5 цифр  [a-z]{3,5} – соответствует латинским словам, состоящим из 3-5 строчных букв |
| {n,} | Соответствует предшествующему шаблону, повторенному n или более раз  Пример:  Приве{1,}т – соответствует строкам «Привет», «Привеет», …, «Привееееееет», …  \d{2,} – соответствует числам, содержащим не менее 2 цифр  [a-z]{3,} – соответствует латинским словам, состоящим не менее, чем из 3 строчных букв |
| {n} | Соответствует в точности n экземплярам предшествующего шаблона  Пример:  \d{2} – число, состоящее из 2х цифр |
| ? | Соответствует нулю или одному экземпляру предшествующего шаблона; предшествующий шаблон является необязательным. Аналог {0,1}.  Пример:  (г\. )?[А-Я][а-я]{2,} – шаблон для задания строк вида «г. Новосибирск», «г. Москва», где сокращение «г.» может отсутствовать, т.е. допустимыми будут также строки «Новосибирск», «Москва», |
| + | Соответствует одному или более экземплярам предшествующего шаблона. Аналог {1,}.  Пример:  \d+ - любое число, состоящее более чем из одной цифры |
| \* | Соответствует нулю или более экземплярам предшествующего шаблона  Эквивалентно {0,} |

*Таблица 3 – метасимволы, отвечающие за группировку символов, частей шаблона*

|  |  |
| --- | --- |
| **Символ** | **Значение** |
| | | Соответствует либо подвыражению слева, либо подвыражению справа (аналог логической операции ИЛИ). |
| ( ) | Группирует элементы в скобках в единое целое, которое может использоваться с символами из таблицы 2, а также символом ‘|’. Запоминает символы, соответствующие этой группе для использования в последующих ссылках. Доступ при таком способе группировке возможен только по индексу (номеру) группы. |
| (?<name>) | Именованная группа – то же самое, что и обычная группа, но с доступом по имени. |
| (?: ) | Только группировка. Группирует элементы в скобках после ‘?:’ в единое целое, но не запоминает символы, соответствующие этой группе. |

Отдельно стоит обратить внимание на то, что если нужно вставить в шаблон какой-то из символов, попавший в одну из этих таблиц, то его нужно экранировать с помощью символа слэша ‘\’. Например, одним из таких символов является точка, которая в терминах регулярных выражений обозначает любой символ, кроме символов разделителей, и если нужно найти в строке именно точку, то ее следует описывать как \.

**Пример использования регулярных выражений**

Для примера рассмотрим задачу поиска в строке всех десятичных чисел, для этого нужно использовать символ \d (десятичная цифра в диапазоне 0-9). После этого необходимо задать количество вхождений, т.к. если оставить \d без модификаторов, то будет производиться поиск чисел, состоящих из одной цифры, и, например, для строки “123” регулярное выражение покажет 3 совпадения – для каждой из цифр 1,2,3. Чтобы обработать все число целиком, нужно указать кратность, охватывающую его целиком, а т.к. числа могут быть любыми по длине, используем символ ‘+’ (от 1-го до бесконечности, подробнее в табл. 2). В этом случае регулярного выражения примет вид: \d+

Чтобы использовать это регулярное выражение в C#, нужно создать объект класса Regex, передав ему шаблон:

Regex regex = new Regex(@“\d+”); //символ @ используется для автоматического экранирования слэшей.

Затем, используя объект этого класса мы можем проверить удовлетворяет ли какая-то строка переданному шаблону:

string s = “Hello, World!

regex. IsMatch (s); //false

string s2 = “Hello, R2D2”;

regex.IsMatch(s2); //true

Если мы хотим включить дополнительные символы для поиска, их нужно добавить в соответствующей позиции шаблона. Например, для поиска вещественных чисел нужно к имеющемуся шаблону добавить разделитель (точка или запятая) и дробную часть числа. В этом случае выражение примет вид:

//Регулярное выражение для поиска десятичных вещественных чисел

Regex regex2 = new Regex(@“\d+[\.,]\d+”);

Чтобы сделать шаблон универсальным – как для целых, так и для вещественных чисел, нужно дробную часть сделать опциональной, для этого можно использовать оператор группировки из таблицы 3 совместно с указанием кратности {0,1} или оператором ‘?’:

//Регулярное выражение для поиска десятичных чисел:

Regex regex3 = new Regex(@“\d+(?:[\.,]\d+)?”);

Чтобы использовать регулярное выражение и проверить строку на соответствие, необходимо вызвать методы Match или IsMatch, передав в качестве параметра проверяемую строку. Различие этих методов состоит в возвращаемом значении – первый метод возвращает объект типа Match, содержащий информацию о найденном фрагменте текста (значение, позицию в строке), а также ссылку на следующее совпадение, IsMatch возвращает значение типа bool, показывающее, было ли хотя бы одно совпадение с шаблоном в проверяемой строке. Например:

Regex r3 = new Regex(@“\d+(?:[\.,]\d+)?”);

string str = “Эта строка содержит число 234,54!”;

//следующие строки выведут «ОК»

if( r3.IsMatch( str ) ){

Console.WriteLine(“ОК!”);

} else{

Console.WriteLine(“Not ОК ☹”);

}

//а этот код может получить текст, соответствующий шаблону:

Match m = r3.Match( str );

if( m.Success ){ //проверяем успешность поиска

string number = m.Value; //в переменную number попадет 234,54

Console.WriteLine( number );

}

В случае, если строка может содержать несколько совпадений с шаблоном и нужно получить их все, нужно использовать свойство Next объекта Match для перехода к следующему совпадению. Ниже показан пример:

*//* в исходной строке нужно найти все числа, соответствующие стоимостям продуктов

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 = @"\d+(?:[\.,]\d+)?”);

Regex regex = new Regex(pattern);

*// Получаем совпадения в экземпляре класса Match*

Match match = regex.Match(input);

*// отображаем все совпадения*:

while (match.Success) {

*// значение, подходящее под шаблон содержится в свойстве Value*

Console.WriteLine(match.Value);

*// Переходим к следующему совпадению:*

match = match.NextMatch();

}

Для использования части результата, полученного с помощью регулярного выражения, можно использовать группировку. Составим регулярное выражения для поиска в строке RGB цветов. Как правило, RGB цвет записывается в виде шестнадцатеричного числа, начинающегося с символа ‘#’, за которым находятся 6 hex чисел, где каждая пара соответствует отдельному компоненту цвета, например, в записи #20a43f, красному компоненту (R) будет соответствовать 20, зеленому (G) a4 и синему (B) 3f. Каждому компоненту будет соответствовать фрагмент регулярного выражения “[a-f0-9]{2}”. Буквы a-f использованы для описания шестнадцатеричного числа, “{2}” указывает, что компонент каждого цвета состоит из двух цифр. Чтобы мы могли затем обратиться к этому компоненту, его необходимо взять в круглые скобки (сгруппировать). С учетом этого, задача имеет следующее решение:

Regex colorRegex = new Regex(@“#([a-f0-9]{2})([a-f0-9]{2})([a-f0-9]{2})”);

string s = Console.ReadLine(); //вводим строку с консоли

Match m = colorRegex.Match( s );

if( m.Success ){

//нумерация групп начинается с 1

Console.WriteLine($”R = {m.Groups[1].Value}” );

Console.WriteLine($”G = {m.Groups[2].Value}” );

Console.WriteLine($”B = {m.Groups[3].Value}” );

}

В некоторых случаях может оказаться неудобным работать с элементами группы по их номерам, в этом случае им можно задать имена. Например, для примера выше, дадим группам названия, соответствующие цвету, который в них будет содержаться (red, green, blue):

Regex colorRegex =

new Regex(@"#(?<red>[a-f0-9]{2})(?<green>[a-f0-9]{2})(?<blue>[a-f0-9]{2})");

string s = Console.ReadLine();

Match m = colorRegex.Match(s);

if (m.Success) {

Console.WriteLine($"R = { m.Groups["red"].Value}");

Console.WriteLine($"G = { m.Groups["green"].Value}" );

Console.WriteLine($"B = { m.Groups["blue"].Value}" );

}

По сравнению с первым примером изменились шаблон регулярного выражения, и способ обращения к элементам группы.

**Задание**

Для каждого из пунктов заданий данные для проверки брать из текстового файла, в котором на каждой строке будет содержаться новая порция данных.

string [] input = File.ReadAllLines(@“имя\_файла”); //чтение массива строк

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

//проверка i-го элемента массива на соответствие регулярному выражению

}

Тестовые данные в файле должны содержать как строки, удовлетворяющие шаблону, так и не подходящие под него.

* Написать регулярное выражение, которые проверят строки на соответствие: «a», «aaaaaa», «a aa a».
* Написать регулярное выражение, заставляющее вводить не менее 5 алфавитно-цифровых символов.
* Написать регулярное выражение, которое проверят email простого вида (например, [test@test.test](mailto:test@test.test)).
* Выполните одно из заданий, приведенных ниже. Вариант задания согласуется с преподавателем. Для получения значения подстроки использовать группировку.
* Написать регулярное выражение, выполняющее проверку адреса, записанного в формате: «ул. строка д. число/число», например, «ул. Высоцкого д. 20/3». Если введенная строка удовлетворяет шаблону, вывести отдельно название улицы и номер дома, иначе вывести строку «Не найдено».

***Примечание****: слова «ул.», «д.», а также дробь в номере дома могут отсутствовать.*

* Написать регулярное выражение, которые считывает из строки географические координаты городов. Координаты представлены в следующем формате: Новосибирск: широта 60.5678, долгота 54.1234. Названия некоторых городов могут быть написаны через дефис. Если введенная строка удовлетворяет шаблону, вывести отдельно название города, широту и долготу, иначе вывести строку «Не найдено».

***Примечание****: слова «*широта*», «долгота» могут отсутствовать. Координата может быть представлена целым числом.*

* Написать регулярное выражение, которые считывает из строки данные пользователя. Пользователь вводит строку следующего формата: Иванов Иван Иванович, 20 лет, г. Новосибирск. Если введенная строка удовлетворяет шаблону, вывести отдельно название города, фамилию пользователя и его возраст, иначе вывести строку «Не найдено».

***Примечание****: слово «*г.*» может отсутствовать, также необязательным является указание отчества, первой цифрой возраста не может быть 0.*

* Используя текстовый редактор, поддерживающий регулярные выражения, например Notepad++ или Visual Studio, выполните следующие действия над файлом «Лабораторная работа 5 - testData.xml» с использованием регулярных выражений (на каждое действие – отдельный шаблон):
* найдите все значения, попадающие под ваш вариант задания из пункта 4;
* удалите нумерацию строк (автозамена в этих редакторах также поддерживает использование регулярных выражений);
* исправьте форматирование - все узлы первого уровня должны иметь отступ 2 символа;
* некоторые из элементов имеют закрывающийся тег, не совпадающий по имени с открывающимся (пример правильного открытия-закрытия: <имя\_узла> значение </имя\_узла>, неправильного: <имя\_узла> значение </другое\_имя\_узла>).

Исправьте это.

**Дополнительное задание**

* Составьте регулярное выражения выбирающее выражения, содержащее стоимость продукта из текста. Посчитать и выведите стоимость 1 кг. продукта. Например, дана следующая строка:

*«Добро пожаловать в наш магазин, вот наши цены: 1 кг. яблоки - 90 руб., 2 кг. апельсины - 130 руб. Также в ассортименте орехи в следующей фасовке: 0.5 кг. миндаль - 500 руб.»*

Правильным выводом программы будет следующий:

яблоки - 90 руб/кг

апельсины – 65 руб/кг

миндаль – 1000 руб/кг

* Дан файл, содержащий некоторый текст. Требуется разработать программу, которая с помощью регулярных выражений выберет из него все интернет адреса в формате [http://www.sample.domen.ru](http://www.sample.domen.ru/). Все выбранные адреса должны быть записаны в новый файл, содержащий найденную ссылку, а также позицию (смещение от начала файла), где эта ссылка находится в исходном файле.

Адреса, хранящиеся в файле, должны удовлетворять следующим ограничениям:

* количество уровней доменных имен может варьироваться, минимальное количество равно двум (например myDomen.ru), максимальное - пяти (например domen5.domen4.domen3.domen2.domen1);
* доменное имя может включать буквы, цифры, тире, знак нижнего подчеркивания, но при этом не может начинаться с тире;
* в составе адреса может отсутствовать указание протокола, а также аббревиатура www;
* если же протокол указан, то это может быть одна из следующих записей: http, https, ftp.

**Источники литературы**

* <https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/regular-expressions>
* <https://professorweb.ru/my/csharp/charp_theory/level4/4_10.php>
* <https://metanit.com/sharp/tutorial/7.4.php>
* <http://website-lab.ru/article/regexp/shpargalka_po_regulyarnyim_vyirajeniyam/>