УО «МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ А.А. КУЛЕШОВА»

СОЦИАЛЬНО-ГУМАНИТАРНЫЙ КОЛЛЕДЖ

Дисциплина
«Конструирование программ и языки программирования»

Создание и применение регулярных выражений (2 часа)

Методические рекомендации к лабораторной работе № 8

Могилев 2018

Понятие «Регулярные выражения». Методические указания по лабораторной работе №8 по дисциплине «Конструирование программ и языки программирования». Для учащихся очной формы обучения специальности 2–40 01 01 «Программное обеспечение информационных технологий.

Оглавление

1 Цель работы

Приобретение навыков практического применения, закрепление знаний при создании простейших программ с использованием регулярных выражений.

2 Ход работы

1. Изучение теоретического материала.
2. Выполнение практических индивидуальных заданий по вариантам (вариант уточняйте у преподавателя).
3. Оформление отчета.
   3.1. Отчет оформляется индивидуально каждым студентом. Отчет должен содержать задание, алгоритм и листинг программы.
   3.2. Отчет по лабораторной работе выполняется на листах формата А4. В состав отчета входят:
      1) титульный лист;
      2) цель работы;
      3) текст индивидуального задания;
      4) выполнение индивидуального задания.
4. Контрольные вопросы.

3 Краткие теоретические сведения

Регулярные выражения предназначены для обработки текстовой информации и обеспечивают:

- эффективный поиск в тексте по заданному шаблону;
- редактирование, замену и удаление подстрок;
- формирование итоговых отчетов по результатам работы с текстом.

Регулярное выражение — это шаблон (образец), по которому выполняется поиск соответствующего ему фрагмента текста. Язык описания регулярных выражений состоит из символов двух видов: обычных и метасимволов. Обычный символ представляет в выражении сам себя, а метасимвол — некоторый класс символов, например любую цифру или букву.

Класс символов	Описание	Пример
.	Любой символ, кроме \n	Выражение c.t соответствует фрагментам cat, cut, c1t, c{t и т. д.
[]	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование диапазонов символов	Выражение c[aul]t соответствует фрагментам cat, cut и c1t, а выражение c[a-z]t — фрагментам cat, cbt, cct, cdt, ..., czt
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование диапазонов символов	Выражение c[^aul]t соответствует фрагментам cbt, c2t, cXt и т. д., а выражение c[^a-zA-Z]t — фрагментам cит, c1t, cЧt, c3t и т. д.
\w	Любой алфавитно-цифровой символ, то есть символ из множества прописных и строчных букв и десятичных цифр	Выражение c\wt соответствует фрагментам cat, cut, c1t, c0t и т. д., но не соответствует фрагментам c{t, c;t и т. д.
\W	Любой *не* алфавитно-цифровой символ, то есть символ, не входящий в множество прописных и строчных букв и десятичных цифр	Выражение c\Wt соответствует фрагментам c{t, c;t, c t и т. д., но не соответствует фрагментам cat, cut, c1t, c0t и т. д.
\s	Любой пробельный символ, например символ пробела, табуляции (\t, \v), перевода строки (\n, \r), новой страницы (\f)	Выражение \s\w\w\w\s соответствует любому слову из трех букв, окруженному пробельными символами
\S	Любой *не* пробельный символ, то есть символ, не входящий в множество пробельных	Выражение \s\S\S\s соответствует любым двум непробельным символам, окруженным пробельными
\d	Любая десятичная цифра	Выражение c\dt соответствует фрагментам c1t, c2t, ..., c9t
\D	Любой символ, не являющийся десятичной цифрой	Выражение c\Dt не соответствует фрагментам c1t, c2t, ..., c9t

Метасимвол	Описание	Пример
*	Ноль или более повторений предыдущего элемента	Выражение ca*t соответствует фрагментам ct, cat, caat, caaaaaaaaaaat и т. д.
+	Одно или более повторений предыдущего элемента	Выражение ca+t соответствует фрагментам cat, caat, caaaaaaaaaaat и т. д.
?	Ни одного или одно повторение предыдущего элемента	Выражение ca?t соответствует фрагментам ct и cat
{n}	Ровно n повторений предыдущего элемента	Выражение ca{3}t соответствует фрагменту caaat, а выражение (cat){2} — фрагменту catcat[1]
{n,}	По крайней мере n повторений предыдущего элемента	Выражение ca{3,}t соответствует фрагментам caaat, caaaat, caaaaaaaaaaat и т. д.
{n,m}	От n до m повторений предыдущего элемента	Выражение ca{2,4}t соответствует фрагментам caat, caaat и caaaat

Классы библиотеки .NET для работы с регулярными выражениями объединены в пространство имен System.Text.RegularExpressions. Класс Regex представляет собственно регулярное выражение. Класс является неизменяемым, то есть после создания экземпляра его корректировка не допускается. Для описания регулярного выражения в классе определено несколько перегруженных конструкторов:

- Regex() — создает пустое выражение;
- Regex(String) — создает заданное выражение;
- Regex(String, RegexOptions) — создает заданное выражение и задает параметры для его обработки с помощью элементов перечисления RegexOptions (например, различать или не различать прописные и строчные буквы).

Пример конструктора, задающего выражение для поиска в тексте повторяющихся слов, расположенных подряд и разделенных произвольным количеством пробелов, независимо от регистра:

```
Regex rx = new Regex( @"\b(?<word>\w+)\s+(\k<word>)\b", RegexOptions.IgnoreCase );
```

Поиск фрагментов строки, соответствующих заданному выражению, выполняется с помощью методов IsMatch, Match и Matches.

Метод **IsMatch** возвращает true, если фрагмент, соответствующий выражению в заданной строке найден, и false в противном случае.

Метод **Match** класса Regex, в отличие от метода IsMatch, не просто определяет, произошло ли совпадение, а возвращает объект класса Match — очередной фрагмент, совпавший с образцом.

Метод **Matches** класса Regex возвращает объект класса MatchCollection — коллекцию всех фрагментов заданной строки, совпавших с образцом.

Метод **Split** класса Regex разбивает заданную строку на фрагменты в соответствии с разделителями, заданными с помощью регулярного выражения, и возвращает эти фрагменты в массиве строк.

Метод **Replace** класса Regex позволяет выполнять замену фрагментов текста.

Пример 1. Замена подстрок

Предположим, список рассылки содержит записи, в которых, помимо имени и фамилии, может указываться обращение ("Mr.", "Mrs.", "Miss" или "Ms."). Если при создании меток для конвертов по такому списку указывать обращение не требуется, можно использовать для удаления

обращений регулярное выражение, как показано в следующем примере.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
   public static void Main()
   {
      string pattern = "(Mr\\.? |Mrs\\.? |Miss |Ms\\.? )";
      string[] names = { "Mr. Henry Hunt", "Ms. Sara Samuels",
                         "Abraham Adams", "Ms. Nicole Norris" };
      foreach (string name in names)
         Console.WriteLine(Regex.Replace(name, pattern, String.Empty));
   }
}

// The example displays the following output:
//    Henry Hunt
//    Sara Samuels
//    Abraham Adams
//    Nicole Norris
```

Шаблон регулярного выражения (Mr\.? |Mrs\.? |Miss |Ms\.?) совпадает с любым вхождением "Mr ", "Mr. " , "Mrs ", "Mrs. " , "Miss ", "Ms или "Ms. " . Вызов метода Regex.Replaceприведет к замене найденных при сопоставлении подстрок на String.Empty; другими словами, найденная подстрока удаляется из исходной строки.

Пример 2. Определение повторяющихся слов

Случайное повторение слов — частая ошибка писателей. Для выявления повторяющихся слов можно использовать регулярное выражение, как показано в следующем примере.

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      string pattern = @"\b(\w+?)\s\1\b";
      string input = "This this is a nice day. What about this? This taste
s good. I saw a a dog.";
      foreach (Match match in Regex.Matches(input, pattern, RegexOptions.I
gnoreCase))
         Console.WriteLine("{0} (duplicates '{1}) at position {2}",
                           match.Value, match.Groups[1].Value, match.Index
);
   }
}

// The example displays the following output:
```

```
//       This this (duplicates 'This)' at position 0
//       a a (duplicates 'a)' at position 66
```

Конструкции группирования отображают части выражений регулярных выражений и захватывают части строки входной строки.

Следующая конструкция группирования захватывает соответствующую часть выражения:

```
(часть_выражения)
```

где часть выражения — это любой допустимый шаблон регулярного выражения. Записывает, что использование скобок нумеруется автоматически слева направо, основываясь на порядке открывающих скобок в регулярном выражении, начиная с 1. Захват с номером 0 – это текст, соответствующий всему шаблону регулярного выражения.

Можно получить доступ к группам записи следующими способами:

- С помощью **конструкции обратной ссылки** в регулярном выражении. Для ссылки на соответствующую часть выражения в том же регулярном выражении используется синтаксис \номер, где номер – это порядковый номер захваченной части выражения.
- С помощью **конструкции именованной обратной ссылки** в регулярном выражении. Для ссылки на соответствующую часть выражения в том же регулярном выражении используется синтаксис \k<номер>, где номер – это порядковый номер захваченной части выражения. Захваченная часть выражения получает имя по умолчанию, идентичное ее порядковому номеру. Дополнительные сведения см. в разделе Именованные соответствующие части выражений.

В следующем примере приводится регулярное выражение, которое определяет повторяющиеся слова в тексте.

```
using System;
using System.Text.RegularExpressions;

public class Example
{
   public static void Main()
   {
      string pattern = @"(\w+)\s(\1)";
      string input = "He said that that was the the correct answer.";
      foreach (Match match in Regex.Matches(input, pattern, RegexOptions.I
gnoreCase))
         Console.WriteLine("Duplicate '{0}' found at positions {1} and {2}
.",
                           match.Groups[1].Value, match.Groups[1].Index, m
atch.Groups[2].Index);
   }
}
// The example displays the following output:
//       Duplicate 'that' found at positions 8 and 13.
//       Duplicate 'the' found at positions 22 and 26.
```

Шаблон регулярного выражения определяется следующим образом:

```
(\w+)\s(\1)\W
```

В следующей таблице показано, как интерпретируется шаблон регулярного выражения.

Шаблон	Описание
(\w+)	Совпадение с одним или несколькими символами слова. Это первая группа записи.
\s	Соответствует пробелу.
(\1)	Сопоставить строку в первой захваченной группе. Это вторая группа записи. В примере эта группа назначается захваченной группе, так что начальная позиция повторяющегося слова может быть получена из свойства Match.Index.
\W	Сопоставить с небуквенным символом, включая пробелы и знаки препинания. Это не позволяет шаблону регулярного выражения выделять слова, которые начинаются со слова из первой группы записи.

4 Задания

1. Выполнить задание по вариантам.

Вариант 1

Создайте программу, которая будет проверять корректность ввода логина. Корректным логином будет строка от 2-х до 10-ти символов, содержащая только буквы и цифры, и при этом цифра не может быть первой. Пример: fi662eg.

Вариант 2

Создайте программу, которая будет проверять корректность ввода фамилии и имени, разделенных пробелом. Первые буквы должны быть заглавными. Пример: Пупкин Петя.

Вариант 3

Создайте программу, которая будет осуществлять ввод номера телефона по шаблону. Шаблон: 55-555-55-55. Пример: 12-655-79-47

Вариант 4

Создайте программу, которая будет находить текст внутри шаблонных скобок. Шаблон: <[text]>.
Пример: <[text<[test]>. должно найти: test. (без самих скобок!)

Вариант 5

Создайте программу, которая будет находить слова, написанные через тире в предложении, заканчивающимся только на восклицательный знак. Пример: single-minded people! blue-green. color! должно найти: single-minded.

Вариант 6

Создайте программу, которая будет осуществлять ввод суммы денег. Деньги должны вводиться в виде числа, заканчивающегося на знак доллара($), на сокращение российских рублей (RUB) или на сокращение белорусских рублей(BYN). Знак минуса перед числом разрешается.
Пример: -123$

Вариант 7

Создайте программу, которая будет осуществлять ввод числа в шестнадцатеричной системе. Число должно начинаться со специальной приставки (0x) либо заканчиваться на суффикс (h) и иметь длину в 8 разрядов. буквы могут быть разного регистра.
Пример 1: 0x4f09E122
Пример 2: 3123fFA3h

Вариант 8

Создайте программу, которая будет осуществлять ввод простого адреса сайта. Имя может начинаться с типа протокола (http://), должен содержать идентификатор и доменное имя(.com либо .ru). Идентификатор состоит из латинских букв нижнего регистра и цифр.
Пример 1: http://vk.com
Пример 2: yandex.ru

Вариант 9

Создайте программу, которая будет осуществлять ввод комбинации кода по шаблону с повторением. Код – наборы из четырех цифр, разделенных знаками дефиса. Наборы цифр могут повторяться. Пример 1: 1234-7463-1554 Пример 2: 1234-7463-1554-3456 Пример 3: 5746

Вариант 10

Создайте программу, которая будет осуществлять ввод сочетания клавиш. Оно должно состоять из двух или трех элементов. Первый элемент должен быть одним из Ctrl, Alt, Shift. следующие два должны быть одной буквой (любого регистра) или цифрой. Элементы разделены знаком плюса (+).
Пример 1: Shift+f+4
Пример 2: Ctrl+H

Вариант 11

Создайте программу, которая будет осуществлять ввод числа-полиндром. вводится число, состоящее из семи цифр, при повороте которого получается это же число.
Пример 1: 1234321
Пример 2: 8389838

Вариант 12

Создайте программу, которая будет осуществлять ввод простого адреса почтового ящика. почта должна состоять из части с буквами, цифрами и нижнего подчеркивания, из знака эт (@), из части с буквами, точки, части с буквами.
Пример 1: Petya_Pupkin@mail.xyz
Пример 2: asdf453@list.com

Вариант 13

Создайте программу, которая будет осуществлять ввод предложения. Предложение должно начинаться на цифру или большую букву. должно заканчиваться на один из символов (.?!). в середине предложения не должны встречаться символы (.?!).
Пример 1: 12 кораблей плыли по морю!
Пример 2: Петя поел в столовой?

Вариант 14

Создайте программу, которая будет реализовывать игру в слова. Вводом является комбинация из двух слов больше одной буквы разделенных пробелом, второе слово должно начинаться на последнюю букву первого слова
Пример 1: table ellipse
Пример 2: ball limit

Вариант 15

Создайте программу, которая будет осуществлять ввод простой JSON-строки, состоящей из одной пары значений. JSON-строки состоят из пар ключ-значение, заключенных в двойные ковычки("). ключ и значение могут содержать любые символы кроме символа двойных ковычек("). ключ и значение должны разделяться знаком двоеточия(:). строка должна иметь на конце и в начале фигурные скобки ({}).
Пример 1: {"test":"text"}
Пример 2: {"asdf":"123"}

5 Контрольные вопросы

1. Для чего используются регулярные выражения?
2. Для чего используются конструкции группирования и обратных ссылок?
3. Для чего используется класс Regex?