# Дополнительное задание к работе №6 "Регулярные выражения"

**Цель работы:** Получить навыки работы с регулярными выражениями.

## **1.Теоретические сведения**

### **1.1. Классы стандартной библиотеки .NET для работы с регулярными выражениями**

Классы, предназначенные для работы с регулярными выражениями, содержатся в сборке System.dll и описаны в пространстве имен **System.Text.RegularExpressions**. Основным классом, обеспечивающим реализацию всех действий, связанных с использованием регулярных выражений (поиск, замена, разбиение строки), является класс **Regex**. Все прочие классы являются вспомогательными, и используются при описании параметров методов класса Regex или их возвращаемых значений.

В описаниях методов те параметры, которые можно не указывать, заключаются в квадратные скобки [ ].

### **Класс Regex:**

### **Конструктор и свойства**

Regex(pattern: string [; options: RegexOptions]);

где pattern – регулярное выражение, options – дополнительные параметры поиска.

**Перечисление RegexOptions:**

* **Compiled**: при установке этого значения регулярное выражение компилируется в сборку, что обеспечивает более быстрое выполнение.
* **CultureInvariant**: при установке этого значения будут игнорироваться региональные различия.
* **IgnoreCase**: при установке этого значения будет игнорироваться регистр.
* **IgnorePatternWhitespace**: удаляет из строки пробелы и разрешает комментарии, начинающиеся со знака #.
* **Multiline**: указывает, что текст надо рассматривать в многострочном режиме. При таком режиме символы "^" и "$" совпадают, соответственно, с началом и концом любой строки, а не с началом и концом всего текста.
* **RightToLeft**: приписывает читать строку справа налево.
* **Singleline**: устанавливает однострочный режим, а весь текст рассматривается как одна строка.

### **Основные методы**

Все основные методы класса **Regex** реализованы в двух вариантах: статическом (вызов через имя класса) и объектном (вызов через имя объекта), при этом каждый из вариантов перегружен для нескольких наборов параметров.

Имеется шесть основных методов.

|  |  |  |
| --- | --- | --- |
| **Название метода** | **Параметры (первый вариант для статического метода, второй для метода объекта)** | **Результат** |
| **IsMatch()** | * string input, string pattern [,RegexOptions options]), где input определяет строку, в которой выполняется поиск, pattern - регулярное выражение, options – дополнительные параметры. * string pattern [, int start], где pattern регулярное выражение,start – позиция, с которой начинается поиск. | boolean - возвращает True, если требуемое выражение найдено, и False в противном случае; |
| **Match()** | * string input, string pattern [,RegexOptions options]), где input определяет строку, в которой выполняется поиск, pattern - регулярное выражение, options – дополнительные параметры. * string pattern [, int start], где pattern регулярное выражение, start – позиция, с которой начинается поиск. | Match- возвращает первое найденное выражение; |
| **Matches()** | * string input, string pattern [,RegexOptions options]), где input определяет строку, в которой выполняется поиск, pattern - регулярное выражение, options – дополнительные параметры. * string pattern [, int start], где pattern - регулярное выражение, start – позиция с которой начинается поиск. | MatchCollection - возвращает все найденные выражения; |
| **Split()** | * string input, string pattern [,RegexOptions options]), где input определяет строку, в которой выполняется поиск, pattern - pattern, options – дополнительные параметры. * string pattern [, int count[, int start]]), где pattern - регулярное выражение, count – максимальное количество раз, которое выполняется поиск, start – позиция, с которой начинается поиск. | string[] - разбивает строку на фрагменты; разделители фрагментов определяются регулярным выражением; |
| **Replace()** | * string input, string pattern, string replacement [,RegexOptions options]), где input определяет строку, в которой выполняется поиск, pattern - регулярное выражение, replacement – строка, на которую выполняется замена, options – дополнительные параметры. * string pattern, string replacement[, int count[, int start]], где pattern - регулярное выражение, replacement – строка, на которую выполняется замена, count – максимальное количество раз, которое выполняется поиск, start – позиция, с которой начинается поиск. | string - заменяет найденные выражения. |

## **3. Язык регулярных выражений**

### **Специальные символы**

Язык регулярных выражений использует некоторые символы в качестве специальных (управляющих) символов; к ним относятся \, \*, +, ?, |, {, [, (, ), ^, $, ., #. Если символ, указанный в регулярном выражении, не совпадает с одним из его специальных символов, то он считается обычным символом. Для того чтобы специальный символ языка регулярных выражений интерпретировался как обычный символ, его надо экранировать, поместив перед ним символ \ (обратная косая черта). Для автоматического экранирования всех требуемых символов строки s достаточно вызвать для нее метод **Regex.Escape(s),** возвращающий преобразованную строку.

Для указания некоторых символов, не имеющих визуального представления, а также для указания символов по их коду можно использовать следующие управляющие последовательности:

* \t - табуляция;
* \r - возврат каретки;
* \f - новая страница;
* \n - новая строка;
* \xNN - ASCII-символ в 16-ричной системе счисления;
* \uNNNN - Unicode-символ в 16-ричной системе счисления.

### **Множества символов**

Для указания того, что в данной позиции регулярного выражения может находиться один из символов, входящих в некоторое множество, можно использовать следующие конструкции:

* [abcd] - допускается один из символов, указанных в списке;
* [^abcd] - допускается любой из символов, кроме тех, которые указаны в списке;
* [a-d] - допускается один из символов, лежащих в указанном диапазоне;
* [^a-d] - допускается любой из символов, кроме тех, которые лежат в указанном диапазоне;
* \d - десятичная цифра, т. е. [0-9];
* \D - любой символ, кроме десятичной цифры;
* \w - словообразующий символ (буква, цифра или символ подчеркивания);
* \W - любой символ, не являющийся словообразующим;
* \s - пробельный символ, т. е. [ \t\r\f\n];
* \S - любой непробельный символ;
* . - любой символ, кроме \n (в режиме SingleLine - любой символ).

Символы, указываемые в квадратных скобках, не должны экранироваться (за исключением символа ]).

### **Квантификаторы**

Любой квантификатор, указанный после символа, множества или группы символов, означает, что этот символ, элемент из множества или группа могут повторяться несколько раз. Количество повторений определяется типом квантификатора:

* \* - 0 или более повторений;
* + - 1 или более повторений;
* ? - 0 или 1 повторение;
* {N} - ровно N повторений;
* {N,} - не менее N повторений;
* {N,M} - от N до M повторений.

#### **Примеры.**

* 1. Ищется имя файла cv.doc, возможно, снабженное нумерацией (обратите внимание на экранирование точки):

Regex.IsMatch('cv12.doc', 'cv\d\*\.doc') *// True*

* 1. Ищется имя файла cv.doc, которое оканчивается произвольным текстом:

Regex.IsMatch('cvnew.doc', 'cv.\*\.doc') *// True*

Все указанные выше квантификаторы являются "жадными", т. е. захватывают максимально возможное количество символов, удовлетворяющих регулярному выражению. Добавление суффикса ? превращает квантификатор в "ленивый" (захватывающий минимально возможное количество символов):

* 1. Regex.Match('zzABCzz', '.\*') *// ABC*Regex.Match('zzABCzz', '.\*?') *// AB*

### **Директивы нулевой длины**

Директивы нулевой длины указывают дополнительные условия, связанные с расположением требуемого вхождения. Их название объясняется тем, что они никогда не включаются в найденные вхождения:

* ^ - начало строки (в режиме Multiline - начало любой строчки многострочного текста);
* $ - конец строки (в режиме Multiline - конец любой строчки многострочного текста);
* \A - начало строки (в любом режиме);
* \z - конец строки (в любом режиме);
* \Z - конец строки или строчки многострочного текста;
* \b - позиция на границе слова (границей слова считается позиция, в которой словообразующий символ \w соседствует либо с началом/концом строки, либо с символом, который не является словообразующим);
* \B - позиция не на границе (т. е. внутри) слова;
* (?=expr) - продолжать поиск, если для выражения expr есть соответствие справа (положительный просмотр вперед);
* (?!expr) - продолжать поиск, если для выражения expr нет соответствия справа (отрицательный просмотр вперед);
* (?<=expr) - продолжать поиск, если для выражения expr есть соответствие слева (положительный просмотр назад);

#### **Примеры.**

1. Regex.Match('zzABCzz', '.\*(?=)') *// zzABC*Regex.Match('zzABCzz', '(?<=).\*(?=)') *// ABC*
2. Распознавание концов строчек многострочного текста, которые в Windows помечаются двумя символами #13#10.

foreach m: Match **in** Regex.Matches('a.txt'#13#10'b.doc'#13#10'c.txt'#13#10'd.txt', '.\*\.txt(?=\r?$)', RegexOptions.Multiline) **do** Write(m, ' '); *// a.txt c.txt d.txt*

1. Подсчет числа пустых строк в исходном тексте text (квантификатор ? в этом, как и в предыдущем, примере нужен для того, чтобы правильно обработать последнюю строку, после которой могут отсутствовать символы #13#10):

Regex.Matches(text, '^\r?$', RegexOptions.Multiline).Count

### **Группирование и ссылки на группы**

С помощью круглых скобок можно формировать группы регулярного выражения, на которые в дальнейшем можно ссылаться по номеру или по имени:

* (expr) - включить соответствие для выражения expr в нумерованную группу (группы нумеруются от 1 согласно порядку следования их открывающих скобок; группа 0 соответствует всему найденному вхождению);
* (?expr) или (?'name'expr) - включить соответствие для выражения expr в именованную группу с именем name;
* (?:expr) - группирующее выражение, не связываемое с нумерованной или именованной группой;
* \N - ссылка на ранее найденную группу с номером N, означающая, что в данной позиции регулярного выражения должен содержаться текст этой группы;
* \k - ссылка на группу с именем nаme.

#### **Пример.**

Чтобы выделить в регулярном выражении для телефонного номера \d{3}-\d{3}-\d{4} начальную группу из трех цифр (код региона) и завершающую группу из 7 цифр (собственно телефонный номер), достаточно заключить их в круглые скобки и воспользоваться свойством Groups:

**var** m := Regex.Match('123-456-7890', '(\d{3})-(\d{3}-\d{4})');Writeln(m.Groups[0]); *// 123-456-7890*Writeln(m.Groups[1]); *// 123*Writeln(m.Groups[2]); *// 456-7890*

### **Комментарии**

Комментарии, указанные в регулярном выражении, игнорируются при его обработке:

* (?#comment) - комментарий;
* #comment - комментарий до конца строки (только в режиме IgnorePatternWhitespaces).

### Некоторые подстановки в выражениях замены

В выражениях замены используются управляющие последовательности (подстановки), отличные от директив языка регулярных выражений. Ниже приводятся наиболее часто используемые подстановки:

* $$ - символ $;
* $0 - все найденное вхождение;
* $\_ - вся исходная строка;
* $N - найденная группа с номером N (или пустая строка, если группа не найдена);
* ${name} - найденная группа с именем name (или пустая строка, если группа не найдена).

#### **Пример**

Заключаем все числа в угловые скобки.

Regex.Replace('10+2=12', '\d+', '<$0>') *// <10>+<2>=<12>*

### **Опции поиска**

Опции поиска можно указывать не только в виде параметра методов класса Regex и его конструктора, но и непосредственно в регулярном выражении. Для этого предназначена особая директива (?opt), в которой в качестве opt указывается буква соответствующей опции (см. описание класса RegexOptions). Любая опция, кроме (?r), может указываться в любом месте регулярного выражения и впоследствии может быть отменена директивой (?-opt). Опция (?r) должна быть указана в начале регулярного выражения и не может быть отменена. Опции можно объединять; например, директива (?i-ms) включает опцию i и одновременно отключает опции m и s.

#### **Примеры.**

1. Regex.Match('a', 'A', RegexOptions.IgnoreCase) *// a*Regex.Match('a', '(?i)A') *// a*Regex.Match('BaAaAab', '(?i)A+') *// aAaAa*Regex.Match('BaAAaab', '(?i)a(?-i)a') *// Aa*

## **Задание**

Определить соответствует ли строка правилам, заданным в варианте.

## **Варианты**

|  |  |  |
| --- | --- | --- |
| **№** | **Задание** | **Пояснение** |
|  | Написать регулярное выражение определяющее является ли данная строчка строкой "abcdefghijklmnopqrstuv18340" или нет. | Пример правильных выражений:  abcdefghijklmnopqrstuv18340 Пример неправильных выражений:  abcdefghijklmnoasdfasdpqrstuv18340 |
|  | Написать регулярное выражение определяющее является ли данная строчка GUID с или без скобок. Где GUID это строчка, состоящая из 8, 4, 4, 4, 12 шестнадцатеричных цифр разделенных тире | Пример правильных выражений:  {e02fa0e4-01ad-090A-c130-0d05a0008ba0} e02fd0e4-00fd-090A-ca30-0d00a0038ba0 Пример неправильных выражений:  02fa0e4-01ad-090A-c130-0d05a0008ba0} e02fd0e400fd090Aca300d00a0038ba0 |
|  | Написать регулярное выражение определяющее является ли данная строчка валидным URL адресом. В данной задаче правильным URL считаются адреса http и https, явное указание протокола также может отсутствовать. Учитываются только адреса, состоящие из символов, т.е. IP адреса в качестве URL не присутствуют при проверке. Допускаются поддомены, указание порта доступа через двоеточие, GET запросы с передачей параметров, доступ к подпапкам на домене, допускается наличие якоря через решетку. Однобуквенные домены считаются запрещенными. Запрещены спецсимволы, например «-» в начале и конце имени домена. Запрещен символ «\_» и пробел в имени домена. | Пример правильных выражений:  http://www.zcontest.ru http://zcontest.ru http://zcontest.com https://zcontest.ru https://sub.zcontest-ru.com:8080 http://zcontest.ru/dir%201/dir\_2/program.ext?var1=x&var2=my%20value zcon.com/index.html#bookmark   Пример неправильных выражений:  Just Text. http://a.com http://www.domain-.com |
|  | Написать регулярное выражение определяющее является ли данная строчка шестнадцатиричным идентификатором цвета в HTML. Где #FFFFFF для белого, #000000 для черного, #FF0000 для красного и.т.д. | Пример правильных выражений:  #FFFFFF #FF3421 #00ff00   Пример неправильных выражений:  232323 f#fddee #fd2 |
|  | Написать регулярное выражение определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1600 года до 9999 года | Пример правильных выражений:  29/02/2000 30/04/2003 01/01/2003   Пример неправильных выражений:  29/02/2001 30-04-2003 1/1/1899 |
|  | Написать регулярное выражение определяющее является ли данная строчка валидным E-mail адресом. | Пример правильных выражений:  mail@mail.ru valid@megapochta.com aa@aa.info   Пример неправильных выражений:  bug@@@com.ru @val.ru Just Text2 val@val val@val.a.a.a.a 12323123@111[]][] |
|  | Составить регулярное выражение, проверяющее, является ли заданная строчка IP адресом, записанным в десятичном виде | Пример правильных выражений:  127.0.0.1 255.255.255.0 192.168.0.1   Пример неправильных выражений:  1300.6.7.8 abc.def.gha.bcd 254.hzf.bar.10 |
|  | Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть английская буква, цифра и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру. | Пример правильных выражений:  C00l\_Pass SupperPas1   Пример неправильных выражений:  Cool\_pass C00l |
|  | Выделить тегом **<strong>** повторяющиеся слова. | Примеры: Тhis is a test**=>** this is a test  Тhis is is a test **=>** this is <strong>is</strong> a test |
|  | Составить регулярное выражение, проверяющее, является ли заданная строчка числом с запятой или пробелом, в качестве разделителя разрядов. | Пример правильных выражений:  8,205,500.4672 Пример неправильных выражений: 1,5826,000 |
|  | Написать регулярное выражение определяющее является ли данная строчка строкой "abcdefghijklmnopqrstuv18340" или нет. | Пример правильных выражений:  abcdefghijklmnopqrstuv18340 Пример неправильных выражений:  abcdefghijklmnoasdfasdpqrstuv18340 |
|  | Написать регулярное выражение определяющее является ли данная строчка GUID с или без скобок. Где GUID это строчка, состоящая из 8, 4, 4, 4, 12 шестнадцатеричных цифр разделенных тире | Пример правильных выражений:  {e02fa0e4-01ad-090A-c130-0d05a0008ba0} e02fd0e4-00fd-090A-ca30-0d00a0038ba0 Пример неправильных выражений:  02fa0e4-01ad-090A-c130-0d05a0008ba0} e02fd0e400fd090Aca300d00a0038ba0 |
|  | Написать регулярное выражение определяющее является ли данная строчка валидным URL адресом. В данной задаче правильным URL считаются адреса http и https, явное указание протокола также может отсутствовать. Учитываются только адреса, состоящие из символов, т.е. IP адреса в качестве URL не присутствуют при проверке. Допускаются поддомены, указание порта доступа через двоеточие, GET запросы с передачей параметров, доступ к подпапкам на домене, допускается наличие якоря через решетку. Однобуквенные домены считаются запрещенными. Запрещены спецсимволы, например «-» в начале и конце имени домена. Запрещен символ «\_» и пробел в имени домена. | Пример правильных выражений:  http://www.zcontest.ru http://zcontest.ru http://zcontest.com https://zcontest.ru https://sub.zcontest-ru.com:8080 http://zcontest.ru/dir%201/dir\_2/program.ext?var1=x&var2=my%20value zcon.com/index.html#bookmark   Пример неправильных выражений:  Just Text. http://a.com http://www.domain-.com |
|  | Написать регулярное выражение определяющее является ли данная строчка шестнадцатиричным идентификатором цвета в HTML. Где #FFFFFF для белого, #000000 для черного, #FF0000 для красного и.т.д. | Пример правильных выражений:  #FFFFFF #FF3421 #00ff00   Пример неправильных выражений:  232323 f#fddee #fd2 |
|  | Написать регулярное выражение определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1600 года до 9999 года | Пример правильных выражений:  29/02/2000 30/04/2003 01/01/2003   Пример неправильных выражений:  29/02/2001 30-04-2003 1/1/1899 |
|  | Написать регулярное выражение определяющее является ли данная строчка валидным E-mail адресом. | Пример правильных выражений:  mail@mail.ru valid@megapochta.com aa@aa.info   Пример неправильных выражений:  bug@@@com.ru @val.ru Just Text2 val@val val@val.a.a.a.a 12323123@111[]][] |
|  | Составить регулярное выражение, проверяющее, является ли заданная строчка IP адресом, записанным в десятичном виде | Пример правильных выражений:  127.0.0.1 255.255.255.0 192.168.0.1   Пример неправильных выражений:  1300.6.7.8 abc.def.gha.bcd 254.hzf.bar.10 |
|  | Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть английская буква, цифра и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру. | Пример правильных выражений:  C00l\_Pass SupperPas1   Пример неправильных выражений:  Cool\_pass C00l |
|  | Выделить тегом **<strong>** повторяющиеся слова. | Примеры: Тhis is a test**=>** this is a test  Тhis is is a test **=>** this is <strong>is</strong> a test |
|  | Составить регулярное выражение, проверяющее, является ли заданная строчка числом с запятой или пробелом, в качестве разделителя разрядов. | Пример правильных выражений:  8,205,500.4672 Пример неправильных выражений: 1,5826,000 |