РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Стандартные классы String и StringBuilder предназначены для работы со строками и позволяют выполнять над ними различные операции, такие как поиск, замена, вставка и удаление подстрок. Тем не менее, есть классы задач по обработке символьной информации, где стандартных возможностей классов String и StringBuilder явно не хватает. Для того, чтобы облегчить решение подобных задач, в пространстве имен System.Text.RegularExpressions были определены классы Regex, Match и MatchCollection, представляющие собой мощный аппарат работы со строками, основанный на регулярных выражениях.

Регулярное выражение – это шаблон, по которому выполняется поиск соответствующего фрагмента текста. Использование регулярных выражений обеспечивает: проверку строки на соответствие шаблону, поиск в тексте по заданному шаблону, а также разбиение текста на фрагменты.

Метасимволы в регулярных выражениях

Язык описания регулярных выражений состоит из символов двух видов: обычных символов и метасимволов. Обычный символ представляет в выражении сам себя, а метасимвол – некоторый класс символов.

Рассмотрим наиболее употребительные метасимволы:

Класс символов	Описание	Пример
.	Любой символ, кроме \n.	Выражение c.t соответствует фрагментам: cat, cut, c#t, c{t и т.д.
[]	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование диапазонов символов. Для задания диапазона используется символ -.	Выражение c[aui]t соответствует фрагментам: cat, cut, cit. Выражение c[a-c]t соответствует фрагментам: cat, cbt, cct.
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование диапазонов символов.	Выражение c[^aui]t соответствует фрагментам: cbt, cct, c2t и т.д. Выражение c[^a-c]t соответствует фрагментам: cdt, cet, c%t и т.д.
\w	Любой алфавитно-цифровой символ, а также символ подчеркивания.	Выражение c\wt соответствует фрагментам: cbt, cct, c2t, c_t и т.д., но не соответствует фрагментам c%t, c{t и т.д.
\W	Любой символ не удовлетворяющий \w.	Выражение c\Wt соответствует фрагментам: c%t, c{t, c.t и т.д., но не соответствует фрагментам cbt, cct, c2t и т.д.
\s	Любой пробельный символ (пробел, табуляция и переход на	Выражение \s\w\w\w\s соответствует любому слову из трех букв, окруженному

Класс символов	Описание	Пример
	новую строку).	пробельными символами. Следует отметить, что слова стоящие на границе текста не удовлетворяют данному регулярному выражению, т.к. начало/конец строки не являются пробельными символами.
\S	Любой не пробельный символ.	Выражение \s\S\S\S\s соответствует любым трем непробельным символам, окруженным пробельными.
\b	Любой пробельный символ (\s), или начало/конец строки.	Выражение \b\w\w\w\b соответствует любому слову из трех букв, окруженному пробельными символами, или стоящему в начале, или конце строки.
\B	Любой символ, кроме удовлетворяющих \b.	Выражение \B\d\d\d\B соответствует любым трем цифрам, входящим в состав слова так, что ни справа ни слева от них нет конца слова.
\d	Любая десятичная цифра.	Выражение c\dt соответствует фрагментам: c1t, c2t, c3t и т.д.
\D	Любой символ, не являющийся десятичной цифрой.	Выражение c\Dt не соответствует фрагментам: c1t, c2t, c3t и т.д.
|	Задает альтернативу, другими словами, символ | соответствует операции *или*.	Выражение c[a-c]|[0-2] соответствует фрагментам: ca, cb, cc, c0, c1, c2.
^	Если стоит в начале выражения, то фрагмент, совпадающий с регулярным выражением, следует искать только в начале текста. Иначе трактуется просто как символ.	Выражение ^cat определяет последовательность символов cat, расположенную в начале строки.
$	Если стоит в конце выражения, то фрагмент, совпадающий с регулярным выражением, следует искать только в конце текста. Иначе трактуется просто как символ.	Выражение cat$ определяет последовательность символов cat, расположенную в конце строки.
\A	Аналог ^ для многострочной строки.	
\Z	Аналог $ для многострочной строки.	

Если нужно найти какой-то символ, который является метасимволом, например, точку, можно это сделать «защитив» ее обратным слэшем. Т.е., просто точка означает любой одиночный символ, а \. означает символ точку. Например, выражение \d\.\d соответствует фрагментам: 2.6, 4.0, 9.1 и т.д., а выражение C:\\Land соответствует фрагменту: C:\Land

В регулярных выражениях часто используются повторители – метасимволы, которые располагаются непосредственно после обычного символа, или группы символов, и задают количество его повторений в выражении.

Повторители	**Описание**	**Пример**
*	Ноль, или более повторений предыдущего элемента.	Выражение ca*t соответствует фрагментам: ct, cat, caat, caaat и т.д.
+	Одно, или более повторений предыдущего элемента.	Выражение ca+t соответствует фрагментам: cat, caat, caaat и т.д.
?	Не более одного повторения предыдущего элемента.	Выражение ca?t соответствует фрагментам: ct, cat.
{n}	Ровно n повторений предыдущего элемента.	Выражение ca{3}t соответствует фрагменту: caaat. Выражение (cat){2} соответствует фрагменту: catcat.
{n,}	По крайней мере n повторений предыдущего элемента.	Выражение ca{3,}t соответствует фрагментам: caaat, caaaat, caaaaaaat и т.д. Выражение (cat){2,} соответствует фрагментам: catcat, catcatcat и т.д.
{n, m}	От n до m повторений предыдущего элемента.	Выражение ca{2,4}t соответствует фрагментам: caat, caaat, caaaat.

Регулярное выражение записывается в виде строкового литерала, причем перед строкой желательно ставить символ @, который говорит о том, что строку нужно будет рассматривать и в том случае, если она будет занимать несколько строчек на экране. Примеры регулярных выражений:

1. слово rus – @"rus"
2. номер телефона в формате xxx-xx-xx – @"\d\d\d-\d\d-\d\d" или @"\d{3}(-\d\d){2}"
3. целое число со знаком, или без знака – @"[+-]?\d+"
4. номер автомобиля – @"[A-Z]\d{3}[A-Z]{2}\d{2,3}RUS"
5. время в формате чч.мм, или чч:мм – @"([01]\d)|(2[0-4])[\.:][0-5]\d"

Задания

Запишите регулярное выражение, соответствующее:

1) *дате в формате дд.мм.гг, или дд.мм.гггг*
2) *вещественному числу (со знаком и без, с дробной частью и без, с целой частью и без)*

Классы Regex, Match и MatchCollection

Важнейшим классом, поддерживающим работу с регулярными выражениями, является класс Regex. Для описания регулярного выражения в классе определено несколько перегруженных конструкторов:

1. Regex(String) – создает регулярное выражение на основе строкового литерала;
2. Regex(String, RegexOptions) – создает регулярное выражение на основе строкового литерала и задает параметры для его обработки с помощью элементов перечисления RegexOptions (например, различать, или нет прописные и строчные буквы).

В классе Regex реализованы методы IsMach, Mach, Matches, Replace, Split. Рассмотрим данные методы более подробно.

Метод ***IsMach*** возвращает true, если в заданной строке найден фрагмент, соответствующий регулярному выражению; в противном случае метод возвращает false. Например, попытаемся определить, встречается ли в заданном тексте слово *собака*:

```
static void Main()
{
   Regex r = new Regex("собака",RegexOptions.IgnoreCase);
   string text1 = "Кот в доме, собака в конуре.";
   string text2 = "Котик в доме, собачка в конуре.";
   Console.WriteLine(r.IsMatch(text1));
   Console.WriteLine(r.IsMatch(text2));
}
```

Результат работы программы:

```
True
False
```

Замечание

Параметр RegexOptions.IgnoreCase означает, что регулярное выражение применяется без учета регистра символов

Можно использовать конструкцию выбора из нескольких элементов. Варианты выбора перечисляются через вертикальную черту. Например, попытаемся определить, встречается ли в заданном тексте слова *собака* или *кот*:

```
static void Main()
{
   Regex r = new Regex("собака|кот",RegexOptions.IgnoreCase);
   string text1 = "Кот в доме, собака в конуре.";
   string text2 = "Котик в доме, собачка в конуре.";
   Console.WriteLine(r.IsMatch(text1));
   Console.WriteLine(r.IsMatch(text2));
}
```

Результат работы программы:

```
True
True
```

Задание

Объясните, почему для строки text2 метод IsMatch вернул значение True, и что нужно исправить, чтобы производился поиск только целых слов.

Попытаемся определить, есть ли в заданных строках номера телефона в формате xx-xx-xx, или xxx-xx-xx:

```
static void Main()
{
   Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
   string text1 = "tel:123-45-67";
   string text2 = "tel:no";
   string text3 = "tel:12-34-56";
   Console.WriteLine(r.IsMatch(text1));
   Console.WriteLine(r.IsMatch(text2));
   Console.WriteLine(r.IsMatch(text3));
}
```

Результат работы программы:

```
True
False
True
```

Задание

Измените программу так, чтобы можно было определить, содержится в тексте дата в формате дд.мм.гг.

Метод ***Match*** класса Regex возвращает объект класса Match, который ссылается на первый найденный фрагмент текста, соответствующий заданному шаблону.

Следует отметить, что для класса Match можно провести аналогию с АТД «список». Так, все фрагменты текста, соответствующие заданному шаблону, в прямом порядке следования образуют «список». При этом «список» доступен только для чтения.

В классе Match реализовано несколько методов, позволяющих обеспечить работу со «списком». Рассмотрим некоторые из них:

Свойство	Действие
Success	Возвращает true, если список содержит ссылку на найденный фрагмент текста; в противном случае возвращает false.
Length	Возвращает длину найденного фрагмента текста.
Index	Возвращает индекс найденного фрагмента текста в исходной строке.
NextMatch	Переходит к следующему фрагменту текста в «списке».

Рассмотрим следующий пример:

```
static void Main()
{
   Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
```

```
      string text = @"Контакты в Москве tel:123-45-67,
                    123-34-56; fax:123-56-45.
                    Контакты в Саратове tel:12-34-56; fax:12-56-45";
      Match tel = r.Match(text);
      while (tel.Success)
      {
         Console.WriteLine(tel.Value);
         Console.WriteLine("Length={0} Index={1}\n", tel.Length,
                                                     tel.Index);
         tel = tel.NextMatch();
      }
   }
```

Результат работы программы:

```
123-45-67
Length=9 Index=22

123-34-56
Length=9 Index=33

123-56-45
Length=9 Index=48

12-34-56
Length=8 Index=83

12-56-45
Length=8 Index=97
```

Следующий пример: нам надо не просто найти все целые числа в заданной строке, но и подсчитать их сумму:

```
static void Main()
{
   Regex r = new Regex(@"[-+]?\d+");
   string text = @"5*10=50 -80/40=-2";
   Match teg = r.Match(text);
   int sum = 0;
   while (teg.Success)
   {
      Console.Write("{0} ", tel Value);
      sum += int.Parse(teg.ToString());
      teg = teg.NextMatch();
   }
   Console.WriteLine("\nsum={0}", sum);
}
```

Результат работы программы:

```
5 10 50 -80 40 -2
sum=23
```

Задание

Измените программу так, чтобы на экран дополнительно выводилось количество найденных чисел.

Метод ***Matches*** класса Regex возвращает объект класса MatchCollection – коллекцию всех фрагментов заданной строки, совпавших с шаблоном. Полученная коллекция доступна только для чтения. Для просмотра данной коллекции можно использовать цикл foreach, например, следующим образом:

```
static void Main(string[] args)
{
   string text = @"5*10=50-80/40=-2";
   Regex theReg = new Regex(@"[-+]?\d+");
   MatchCollection didgits = theReg.Matches(text);
   Console.WriteLine("Количество чисел в тексте {0}",
                     didgits.Count);
   foreach (Match item in didgits)
   {
      Console.Write("{0} ", item);
   }
   Console.WriteLine();
}
```

Результат работы программы:

```
Количество чисел в тексте 6
5 10 50 -80 40 -2
```

Обратите внимание на то, что переменная item является экземпляром класса Match, поэтому для нее допустимы все методы, определенные в классе Match.

Задание

Измените программу так, чтобы на экран выводились только положительные числа.

Статический метод ***Replace*** класса Regex позволяет выполнять замену одного фрагмента текста другим. В том числе, с его помощью можно удалять фрагменты текста, заменяя их на пустой текст.

Рассмотрим пример, который позволит нам изменить номера телефонов в заданном тексте.

```
static void Main()
{
   string text = @"Контакты в Москве tel: 123-45-67, 123-34-56; fax:
                   123-56-45. Контакты в Саратове tel: 12-34-56;
                   fax: 11-56-45";
   Console.WriteLine("Старые данные: ");
   Console.WriteLine(text);
   string newText = Regex.Replace(text, "123-", "890-");
   Console.WriteLine("\nНовые данные: ");
   Console.WriteLine(newText);
}
```

Результат работы программы:

```
Старые данные:
Контакты в Москве tel: 123-45-67, 123-34-56; fax: 123-56-45.
Контакты в Саратове tel: 12-34-56; fax: 11-56-45
Новые данные:
Контакты в Москве tel: 890-45-67, 890-34-56; fax: 890-56-45.
Контакты в Саратове tel: 12-34-56; fax: 11-56-45
```

Замечание

Данный пример приведен исключительно в демонстрационных целях. В реальности данную задачу лучше решать с помощью метода Replace класса String. Более подробно вопрос о том, когда эффективно использовать регулярных выражений, а когда нет, мы рассмотрим позже.

Использование метода Replace позволяет решить и более сложную задачу – заменить шестизначные номера на семизначные добавлением 0 после первых двух цифр (например, сделать из 12-34-56 номер 120-34-56).

```
static void Main()
{
   string text = @"Контакты в Москве tel: 123-45-67, 123-34-56; fax:
                   123-56-45. Контакты в Саратове tel: 12-34-56;
                   fax: 11-56-45";
   Console.WriteLine("Старые данные: ");
   Console.WriteLine(text);
   string reg = @"\b(\d\d-){2,}\d\d";
   Regex change = new Regex(reg);
   MatchCollection items = change.Matches(text);
   foreach (Match item in items)
   {
      string line = item.ToString();
      Console.WriteLine(item+"   "+ line);
      string newline = line.Insert(2,"0");
      text = Regex.Replace(text,line , newLine);
   }
   Console.WriteLine("\nНовые данные: ");
   Console.WriteLine(text);
}
```

Результат работы программы:

```
Старые данные:
Контакты в Москве tel: 123-45-67, 123-34-56; fax: 123-56-45.
Контакты в Саратове tel: 12-34-56; fax: 11-56-45
Новые данные:
Контакты в Москве tel: 123-45-67, 123-34-56; fax: 123-56-45.
Контакты в Саратове tel: 120-34-56; fax: 110-56-45
```

Задание

Измените программу так, чтобы для номеров задавался код города. Например, номер 120-34-56 заменился бы на (045)120-34-56.

Метод Replace можно использовать для удаления фрагментов текста из строки. Рассмотрим пример, в котором из текста удаляются все номера телефонов:

```
static void Main()
{
   string text = @"Контакты в Москве tel: 123-45-67, 123-34-56; fax:
                   123-56-45. Контакты в Саратове tel: 12-34-56;
                   fax: 12-56-45";
   Console.WriteLine("Старые данные\n"+text);
   string newText = Regex.Replace(text, @"\d{2,3}(-\d\d){2}", "");
```

```
      Console.WriteLine("Новые данные\n" + newText);
   }
```

Результат работы программы:

```
Старые данные:
Контакты в Москве tel: 123-45-67, 123-34-56; fax: 123-56-45.
Контакты в Саратове tel: 12-34-56; fax: 11-56-45
Новые данные:
Контакты в Москве tel: , ; fax: .
Контакты в Саратове tel: ; fax:
```

Задание

Измените программу так, чтобы из текста также удалялись слова tel и fax (если после данных слов стоят двоеточия, то их тоже следует удалить).

Статический метод ***Split*** класса Regex позволяет разбивать текст на фрагменты на основе заданного перечисления разделителей. В качестве результата данный метод возвращает массив строк. Например:

```
static void Main()
{
   string text = @"Контакты в Москве tel: 123-45-67, 123-34-56; fax:
                   123-56-45.";
   string[] newText = Regex.Split(text,"[ ,.:;]+");
   foreach( string a in newText)
      Console.WriteLine(a);
}
```

Результат работы программы:

```
Контакты
в
Москве
tel
123-45-67
23-34-56
fax
123-56-45
```

Задание

Метод Split, реализованный для класса Regex, в отличие от одноименного метода, реализованного для класса String, при разбиении текста на массив слов не создает пустых слов. Объясните, почему.

Скорость работы регулярных выражений

При выполнении операций с использованием регулярных выражений важно помнить о том, что регулярные выражения – это крайне мощное, но и крайне медленное средство. Простейший пример: метод Replace класса Regex работает примерно в 10 раз медленнее аналогичного метода класса string.

Вывод, который следует из этого сделать, следующий: не используйте регулярные выражения там, где можно обойтись работой со строками. Например, задание по замене фрагмента телефонного номера 123 на 890 надо выполнять так:

```
string newText = text.Replace("123-", "890-");
```

а не так:

```
string newText = Regex.Replace(text, "123-", "890-");
```

Практикум №18

В текстовом файле содержится осмысленное сообщение. Слова сообщения разделяются пробелами и знаками препинания.

1. Выведите все слова заданной длины.
2. Выведите на экран все слова сообщения, записанные с заглавной буквы.
3. Удалите из сообщения все однобуквенные слова.
4. Удалите из сообщения только те русские слова, которые начинаются на гласную букву.
5. Замените все английские слова многоточиями.
6. Найдите максимальное целое число, встречающееся в сообщении.
7. Найдите сумму всех имеющихся в тексте чисел (целых и вещественных) причем вещественное число может быть записано в экспоненциальной форме.
8. В сообщении могут встречаться номера телефонов, записанные в формате xx-xx-xx, xxx-xxx, или xxx-xx-xx. Вывести все номера телефонов, которые содержатся в сообщении.
9. В сообщении может содержаться дата в формате дд.мм.гггг. В заданном формате дд – целое число из диапазона от 1 до 31, мм – целое число из диапазона от 1 до 12, а гггг – целое число из диапазона от 1900 до 2010 (если какая-то часть формата нарушена, то данная подстрока в качестве даты не рассматривается). Выведите на экран все даты, которые относятся к текущему году.
10. В сообщении могут содержаться IP-адреса компьютеров в формате d.d.d.d, где d – целое число из диапазона от 0 до 255. Вывести все IP-адреса содержащиеся в тексте.
11. Выведите на экран все адреса web-сайтов, содержащиеся в сообщении.
12. В сообщении может содержаться время в формате чч:мм:сс. В заданном формате чч – целое число из диапазона от 00 до 23, мм и сс – целые числа из диапазона от 00 до 59 (если какая-то часть формата нарушена, то данная подстрока в качестве даты не рассматривается). Вывести на экран все сообщения о времени.