Урок 5

Символы. Строки. ЕГЭ и регулярные выражения

Учимся работать со строками. «Заканчиваем школу» — решаем задачи ЕГЭ по информатике. Знакомимся с регулярными выражениями.

Символы и строки

Обработка текстовой информации — одна из самых распространённых задач современного программирования. C# предоставляет для её решения широкий набор средств: символы char, неизменяемые строки string, изменяемые строки StringBuider и регулярные выражения Regex. В данном разделе мы рассмотрим работу с символами, неизменяемыми и изменяемыми строками.

Символы char

Символьный тип char предназначен для хранения символа в кодировке Unicode. Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .Net из пространства имён System. В этом классе определены статические методы, позволяющие задавать вид и категорию символа, а также преобразовывать символ в верхний или нижний регистр, в число.

Рассмотрим основные методы:

Метод	**Описание**
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и -1 в противном случае.
GetUnicodeCategory	Возвращает категорию Unicode-символа. В Unicode символы разделены на категории, например, цифры (DecimalDigitNumber), римские цифры (LetterNumber), разделители строк (LineSeparator), буквы в нижнем регистре (LowercaseLetter) и т.д.
IsControl	Возвращает true, если символ является управляющим.
IsDigit	Возвращает true, если символ является десятичной цифрой.
IsLetter	Возвращает true, если символ является буквой.
IsLetterOrDigit	Возвращает true, если символ является буквой или десятичной цифрой.
IsLower	Возвращает true, если символ задан в нижнем регистре.
IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным).
IsPunctuation	Возвращает true, если символ является знаком препинания.
IsSeparator	Возвращает true, если символ является разделителем.

IsUpper	Возвращает true, если символ задан в нижнем регистре.
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки, возврат каретки).
Parse	Преобразует строку в символ (строка должна состоять из одного символа).
ToLower	Преобразует символ в нижний регистр.
ToUpper	Преобразует символ в верхний регистр.

Пример:

```
using System;

class Program
{
  static void Main()
     {
       try
        {
         char b = 'B', c = '\x64', d = '\uffff';
         Console.WriteLine("{0}, {1}, {2}", b, c, d);
          Console.WriteLine("{0}, {1}, {2}", char.ToLower(b), char.ToUpper(c),
char.GetNumericValue(d));
         char a;
         do // цикл выполняется, пока не ввели символ e
          {
           Console.WriteLine("Введите символ: ");
           a = char.Parse(Console.ReadLine());
             Console.WriteLine("Введен символ {0}, его код  {1}, его категория
{2}", a, (int)a, char.GetUnicodeCategory(a));
           if (char.IsLetter(a)) Console.WriteLine("Буква");
           if (char.IsUpper(a)) Console.WriteLine("Верхний регистр");
           if (char.IsLower(a)) Console.WriteLine("Нижний регистр");
           if (char.IsControl(a)) Console.WriteLine("Управляющий символ");
           if (char.IsNumber(a)) Console.WriteLine("Число");
           if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");
         } while (a != 'e');
       }
     catch
       {
            Console.WriteLine("Возникло исключение");
         }
   }
}
```

Используя символьный тип, можно организовать массив символов и работать с ним на основе базового класса Array:

```
using System;

class Program
{
static void PrintArray(string line, char[] a)
       {
            Console.WriteLine(line);
            foreach(char  x in a) Console.Write(x);
            Console.WriteLine('\n');
       }

static void Main()
        {
            char[] a ={ 'm', 'a', 'X',  'i', 'M', 'u', 'S' , '!', '!', '!' };
               char [] b="кол около колокола".ToCharArray(); // преобразование
строки в массив символов
            PrintArray("Исходный массив a:", a);
            for (int x=0;x<a.Length; x++)
                if (char.IsLower(a[x])) a[x]=char.ToUpper(a[x]);
            PrintArray("Измененный массив a:", a);
            PrintArray("Исходный массив b:", b);
            Array.Reverse(b);
            PrintArray("Измененный массив b:", b);
       }
}
```

Неизменяемые строки

Тип string, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый тип класса System.String библиотеки .Net. Каждый объект string — это неизменяемая последовательность символов Unicode, то есть методы, предназначенные для изменения строк, возвращают изменённые копии, исходные же строки остаются неизменными.

Создать строку можно несколькими способами:

```
1)    string s;

    // инициализация отложена

2)    string s="Колпак на колпаке, под колпаком колпак";

   // инициализация строковым литералом

3)    string s = @"http://
            geekbrains.ru";
  // символ @ сообщает конструктору string, что строку

  // нужно воспринимать буквально, даже если она занимает несколько строк

4)    string s=new string (' ', 20);

   // конструктор создает строку из 20 пробелов

5)    int x = 12344556;
```

```
   // инициализировали целочисленную переменную
string s = x.ToString();
   // преобразовали ее к типу string
6)    char [] a={'a', 'b', 'c', 'd', 'e'};
   // создали массив символов
string v=new string (a);
   // создание строки из массива символов
7)    char [] a={'a', 'b', 'c', 'd', 'e'};
   // создание строки из части массива символов, при этом:  0
string v=new string (a, 0, 2);
  // показывает с какого символа, 2 - сколько символов использовать для инициализации
```

Класс string обладает богатым набором методов для сравнения строк, поиска в строке и других действий со строками. Рассмотрим эти методы.

Название	Вид	Описание
Compare	Статический метод	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с учётом или без учёта регистра.
CompareTo	Нестатический метод	Сравнение текущего экземпляра строки с другой строкой.
Concat	Статический метод	Слияние произвольного числа строк.
Copy	Статический метод	Создание копии строки.
Empty	Статическое поле	Открытое статическое поле, представляющее пустую строку.
Format	Статический метод	Форматирование строки в соответствии с заданным форматом.
IndexOf, IndexOfAny,	Нестатический метод	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора в данную строку.

LastIndexOf, LastIndexOfAny		
Insert	Нестатический метод	Вставка подстроки в заданную позицию.
Join	Статический метод	Слияние массива строк в единую строку. Между элементами массива вставляются разделители.
Length	Свойство	Возвращает длину строки.
PadLeft, PadRigth	Нестатические методы	Выравнивают строки по левому или правому краю путём вставки нужного числа пробелов в начале или в конце строки.
Remove	Нестатический метод	Удаление подстроки из заданной позиции.
Replace	Нестатический метод	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом.
Split	Нестатический метод	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк.
StartWith, EndWith	Нестатический метод	Возвращают true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой.
Substring	Нестатический метод	Выделение подстроки, начиная с заданной позиции.
ToCharArray	Нестатический метод	Преобразует строку в массив символов.
ToLower, ToUpper	Нестатический метод	Преобразование строки к нижнему или верхнему регистру.
Trim, TrimStart, TrimEnd	Нестатический метод	Удаление пробелов в начале и конце строки или только с одного ее конца.

Напоминаем, что вызов статических методов происходит через обращение к имени класса, например, String.Concat(str1, str2), в остальных случаях — через обращение к экземплярам класса, например, str.ToLower(). На примере рассмотрим использование данных свойств и методов.

```
using System;

class Program
{
static void Main()
{
   string str1 ="Первая строка";
   string str2 = string.Copy(str1);
   string str3 = "Вторая строка";
   string str4 = "ВТОРАЯ строка";
   string strUp, strLow;
   int result, idx;
   Console.WriteLine("str1: " + str1);
   Console.WriteLine("Длина строки str1: " +str1.Length);
// создаем прописную и строчную версии строки str1.
   strLow = str1.ToLower();
   strUp = str1.ToUpper();
   Console.WriteLine("Строчная версия строки str1: " +strLow);
   Console.WriteLine("Прописная версия строки str1: " +strUp);
   Console.WriteLine();
// сравниваем строки
   result = str1.CompareTo(str3);
   if (result == 0) Console.WriteLine("str1 и str3 равны.");
   else if (result < 0) Console.WriteLine("str1 меньше, чем str3");
          else Console.WriteLine("str1 больше, чем str3");
          Console.WriteLine();
// сравниваем строки без учета регистра
   result = String.Compare(str3,str4,true);
      if (result == 0) Console.WriteLine("str3 и str4 равны без учета
регистра.");
         else Console.WriteLine("str3 и str4 не равны без учета регистра.");
         Console.WriteLine();
// сравниваем части строк
   result = String.Compare(str1, 4, str2, 4, 2);
   if (result == 0)  Console.WriteLine("часть str1 и str2 равны");
        else  Console.WriteLine("часть str1 и str2 не равны");
              Console.WriteLine();
// поиск строк
   idx = str2.IndexOf("строка");
   Console.WriteLine("Индекс первого вхождения подстроки строки: " + idx);
   idx = str2.LastIndexOf("о");
   Console.WriteLine("Индекс последнего вхождения символа о: " + idx);
// конкатенация
   string str=String.Concat(str1, str2, str3, str4);
   Console.WriteLine(str);
// удаление подстроки
   str=str.Remove(0,str1.Length);
   Console.WriteLine(str);
// замена подстроки "строка" на пустую подстроку
   str=str.Replace("строка","");
   Console.WriteLine(str);
}
}
```

Очень важные методы обработки — разделения строки на элементы Split и слияние массива строк в единую строку Join.

```
using System;

class Program
{
static void Main()
{
  string poems = "белеет парус одинокий в тумане моря голубом";
  char[] div = { ' '}; // создаем массив разделителей
                              // разбиваем строку на части,
  string[] parts = poems.Split(div);
  Console.WriteLine("Результат разбиения строки на части: ");
  for (int i = 0; i < parts.Length; i++)
   Console.WriteLine(parts[i]);
                                  // собираем эти части в одну строку, в качестве
разделителя используем символ |
  string str = String.Join("|", parts);
  Console.WriteLine("Результат сборки: ");
  Console.WriteLine(str);
}
}
```

Compare и CompareTo

Особенно стоит отметить, как сравниваются строки. Так как в C# нельзя сравнивать строки с помощью операций отношений (<,>), для их сравнения нужно использовать методы Compare или CompareTo.

```
using System;

class Program
{
    static void Main(string[] args)
    {
          string s1="Hello!";
          string s2="Hello!";
          // сравнение с использованием статического метода
          Console.WriteLine(String.Compare(s1,s2));
          // сравнение с использованием нестатического метода
          Console.WriteLine(s1.CompareTo(s2));
    }
}
```

StringBuilder

Чтобы создать строку, которую можно изменять, в C# предусмотрен класс StringBuilder, определённый в пространстве имён System.Text. Объекты этого класса всегда объявляются с явным вызовом конструктора класса (через операцию new) .

Примеры создания изменяемых строк:

```
using System.Text;

class Program
{
    static void Main(string[] args)

    {

          // создание пустой строки, размер по умолчанию 16 символов

           StringBuilder a =new StringBuilder();

          // инициализация строки и выделение необходимой памяти

           StringBuilder b = new StringBuilder("abcd");

          // создание пустой строки и выделение памяти под 100 символов

           StringBuilder c = new StringBuilder(100);

          // инициализация строки и выделение памяти под 100 символов

           StringBuilder d = new StringBuilder("abcd", 100);

          // инициализация подстроки "bcd" и выделение памяти под 100 символов

           StringBuilder e = new StringBuilder("abcd", 1, 3,100);

    }

}
```

Основные элементы класса приведены в таблице:

Название	**Вид**	**Описание**
Append	Нестатический метод	Добавление данных в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки string.
AppendFormat	Нестатический метод	Добавление форматированной строки в конец строки.
Capacity	Свойство	Получение и установка максимального количества символов, которые могут храниться в памяти, выделенной для данного экземпляра StringBuilder. Если устанавливаемое значение меньше текущей длины строки или больше максимального, генерируется исключение ArgumentOutOfRangeException.

Insert	Нестатический метод	Вставка подстроки в заданную позицию.
Length	Изменяемое свойство	Возвращает длину строки. Присвоение ему значения 0 сбрасывает содержимое и очищает строку.
MaxCapacity	Неизменяемое свойство	Возвращает наибольшее количество символов, которое может быть размещено в строке.
Remove	Нестатический метод	Удаление подстроки из заданной позиции.
Replace	Нестатический метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом.
ToString	Нестатический метод	Преобразование в строку типа string.
[]	Индексатор	Возвращает из массива или устанавливает в массиве символ с заданным индексом.
Equals	Нестатический метод	Возвращает true, только если объекты имеют одну и ту же длину, состоят из одних и тех же символов.
CopyTo	Нестатический метод	Копирует подмножество символов строки в массив char.

Как видим, методы класса StringBuilder менее развиты, чем методы класса String, но они позволяют более эффективно использовать память за счёт работы с изменяемыми строками.

Рассмотрим примеры использования данных методов.

```
using System;
using System.Text;

class Program
{
static void Main()
      {
              try
            {
            StringBuilder str=new StringBuilder("Площадь");
            PrintString(str);
            str.Append(" треугольника равна");
            PrintString(str);
```

```
            str.AppendFormat(" {0:f2} см ", 123.456);
            PrintString(str);
            str.Insert(8, "данного ");
            PrintString(str);
            str.Remove(7, 21);
            PrintString(str);
            str.Replace("a", "o");
            PrintString(str);
              Console.WriteLine("Введите первую строку для сравнения:");
            StringBuilder str1=new StringBuilder(Console.ReadLine());
              Console.WriteLine("Введите вторую строку для сравнения:");
            StringBuilder str2=new StringBuilder(Console.ReadLine());
            Console.WriteLine("Строки равны: " + str1.Equals(str2));
                        }
                 catch
            {
                  Console.WriteLine("Возникло исключение");
            }
      }
static void PrintString(StringBuilder a)
      {
        Console.WriteLine("Строка: "+a);
        Console.WriteLine("Текущая длина строки " +a.Length);
        Console.WriteLine("Объем буфера "+a.Capacity);
        Console.WriteLine("Максимальный объем буфера "+a.MaxCapacity);
        Console.WriteLine();
      }
}
```

С изменяемой строкой можно работать не только как с объектом, но и как с массивом символов:

```
using System;
using System.Text;

class Program
{
   static void Main()
   {
      StringBuilder a = new StringBuilder("2*3=3*2");
      Console.WriteLine(a);
       int k=0;
       for (int i = 0; i < a.Length; ++i )
      if (char.IsDigit(a[i]))
                k+=int.Parse(a[i].ToString());
             Console.WriteLine(k);
   }
}
```

На практике часто комбинируют работу с изменяемыми и неизменяемыми строками. Если необходимо изменять строку, используют StringBuilder.

Сравним производительность работы StringBuilder с неизменяемыми строками:

```
using System;
```

```
class StringAppend
{
  const int iIterations = 10000;
  public static void Main()
  {
    DateTime dt = DateTime.Now;
    string str = String.Empty;
    for(int i=0;i<iIterations; i++)
       str += "abcdefghijklmnopqrstuvwxyz\r\n";
    Console.WriteLine(DateTime.Now-dt);
  }
}
```

Те же операции с StringBuilder:

```
using System;
using System.Text;
class StringBuilderAppend
{
   const int iIterations = 10000;
   public static void Main()
   {
     DateTime dt = DateTime.Now;
     StringBuilder sb = new StringBuilder();
     for(int i=0;i<iIterations; i++)
     sb.Append("abcdefghijklmnopqrstuvwxyz\r\n");
     string str = sb.ToString();
     Console.WriteLine(DateTime.Now - dt);
   }
}
```

Регулярные выражения

Стандартный класс string позволяет выполнять над строками различные операции, в том числе поиск, замену, вставку и удаление подстрок. Тем не менее есть классы задач по обработке символьной информации, где стандартных возможностей явно не хватает. Чтобы облегчить решение подобных задач, в .Net Framework встроен более мощный аппарат работы со строками, основанный на регулярных выражениях.

Регулярные выражения предназначены для обработки текстовой информации и обеспечивают:

1. Эффективный поиск в тексте по заданному шаблону.
2. Редактирование текста.
3. Замену всех одинаковых слов в строке синонимичными или их удаление.
4. Выделение из строки необходимой части. Например, из любой ссылки (https://geekbrains.ru/streams) выделить только доменную часть (geekbrains.ru).
5. Проверку соответствия строки заданному шаблону. Например, проверить, правильно ли введен email, телефон и т.д.
6. Извлечение из строки всех входящих подстрок, соответствующих шаблону регулярного выражения. Например, получить все даты из строки.
7. Формирование итоговых отчётов по результатам работы с текстом.

Чтобы работать с регулярными выражениями, необходимо подключить в начале программы пространство имен using **System.Text.RegularExpressions**. В C# работу с регулярными выражениями предоставляет класс Regex.

Создание регулярного выражения имеет следующий вид:

```
Regex myReg = new Regex([шаблон]);
```

Здесь [шаблон] — это строка, содержащая символы и спецсимволы.

У Regex также есть и второй конструктор, который принимает дополнительный параметр — опции поиска.

Пример программы с использованием регулярных выражений:

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

class Program

{

  static void Main()
  {
     string data1 = "Петр, Андрей, Николай";
     string data2 = "Петр, Андрей, Александр";
     Regex regex = new Regex("Николай");      // Создание регулярного выражения
     Console.WriteLine(regex.IsMatch(data1)); // True
     Console.WriteLine(regex.IsMatch(data2)); // False
     Console.ReadKey();
  }

}
```

Здесь в качестве шаблона выступает однозначная строка «Николай». Далее был использован метод IsMatch, который проверяет, содержит ли заданная строка (data1, data2) подстроку, соответствующую шаблону.

Для знакомства регулярными выражениями можно воспользоваться сайтом http://www.regexpal.com/.

На странице имеются два текстовых поля: верхнее и нижнее. Верхнее предназначено для ввода регулярных выражений, нижнее — для целевого текста. Целевой текст — это набор строк, к которым вы хотите применить регулярное выражение.

Метасимволы в регулярных выражениях

Рассмотрим наиболее употребительные метасимволы:

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)		
(...)	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, то есть 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*

Использование регулярных выражений в C#

Безусловно, задачу поиска и замены подстроки в строке можно решить на C# с использованием различных методов System.String и System.Text.StringBuilder. Однако в некоторых случаях это потребует написания большого объёма кода C#. Если вы используете регулярные выражения, код сокращается буквально до нескольких строк. По сути вы создаете экземпляр объекта RegEx, передаёте ему строку для обработки, а также само регулярное выражение (строку, включающую инструкции на языке регулярных выражений) — и всё готово.

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

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 (удобно для некоторых азиатских и других языков, которые читаются в таком направлении).
Singleline	Специфицирует однострочный режим, в котором точка (.) символизирует соответствие любому символу.

Регулярное выражение записывается в виде строкового литерала, причём перед строкой необходимо ставить символ @, который говорит о том, что строку нужно будет рассматривать и в том случае, если она будет занимать несколько строчек на экране. Однако символ @ можно не ставить, если в качестве шаблона используется шаблон без метасимволов.

Замечание. Если нужно найти какой-то символ, который является метасимволом, например, точку, можно это сделать, защитив ее обратным слэшем. То есть просто точка означает любой одиночный символ, а \. означает просто точку.

Примеры регулярных выражений:

слово rus	@rus или rus
номер телефона в формате xxx-xx-xx	@\d\d\d-\d\d-\d\d или @\d{3}(-\d\d){2}
номер автомобиля, etc: D123AF42RUS	@[A-Z]\d{3}[A-Z]{2}\d{2,3}RUS

Наконец, можно не просто извлекать совпадения в исходной строке, а заменять их на собственные значения. Для этого используется метод Regex.Replace().

Проверка, что введённый email-адрес правильный:

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

class Program
{
  static void Main(string[] args)
  {
Regex myReg = new
Regex(@"[A-Za-z]+[\.A-Za-z0-9_-]*[A-Za-z0-9]+@[A-Za-z0-9]+\.[A-Za-z]{2,6}");
     Console.WriteLine(myReg.IsMatch("email@email.com"));// True
```

```
      Console.WriteLine(myReg.IsMatch("email@email"));        // False
      Console.WriteLine(myReg.IsMatch("@email.com"));         // False
      Console.ReadKey();
   }
}
```

Практическая часть урока

Задача 1. Вывести все слова сообщения, которые начинаются и заканчиваются на одну и ту же букву.

Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания. Вывести все слова сообщения, которые начинаются и заканчиваются на одну и ту же букву.

```
using System;
using System.Text;

class Program
{
  static void Main()
       {
       Console.WriteLine("Введите строку: ");
       StringBuilder a = new StringBuilder(Console.ReadLine());
       Console.WriteLine("Исходная строка: "+a);
       for (int i=0; i<a.Length;)
             if (char.IsPunctuation(a[i])) a.Remove(i,1);
             else ++i;
       string str=a.ToString();
       string[] s=str.Split(' ');
              Console.WriteLine("Искомые слова: ");
       for (int i=0; i<s.Length; ++i)
       if (s[i][0]==s[i][s[i].Length-1]) Console.WriteLine(s[i]);
       }
}
```

Задача 2. Задача ЕГЭ. Частота символов.

На вход программы подаются произвольные алфавитно-цифровые символы. Ввод этих символов заканчивается точкой.

Требуется написать программу, которая будет печатать последовательность строчных английских букв ('a' 'b'... 'z') из входной последовательности и частот их повторения.

Печать должна происходить в алфавитном порядке. Например, пусть на вход подаются следующие символы:

fhb5kbfshfm.

В этом случае программа должна вывести:

b2
f3
h2
k1
m1
s1

Здесь необходимо использовать частотный массив.

```
using System;
    class Program
    {
        static void Main(string[] args)
        {
            int[] count = new int[26];
            int i,k, cA;
            char c;
            cA = (int)('a');    // Сохраним код буквы 'a'
                                // В C# при создании массива он автоматически заполняется нулями. Здесь это сделано, чтобы лишний раз подчеркнуть, что частотный массив должен быть полностью заполнен 0
            for (i = 0; i < 26; i++) count[i] = 0;
            Console.Write("Введите строку:");
            string s = Console.ReadLine();
             s = s + '.';     // По условию задачи, строка ДОЛЖНА заканчиваться точкой и добавлять ее нет необходимости. Но мы добавим точку, чтобы можно было самим ее не вводить
            i = 0;
            do
            {
                c = s[i];  // Сохраним текущий символ
                k = (int)c;// Получим код символа
                           // Если символ - маленькая буква
                if (c >= 'a' && c <= 'z')
                      count[k - cA]++;
                           // Увеличим частоту вхождения этой буквы в частотном массиве
                i++;
            }
         while (c != '.');// Повторяем цикл, пока не встретим '.'
                          // Выводим частотный массив на экран
                          // В массиве под номером 0 - сколько раз встретилась 'a'
                         // Под номером 1 - сколько раз встретилась 'b' и т.д.
            for (i = 0; i < 26; i++)
                if (count[i] > 0)
                        // Если символ встретился хотя бы один раз
                    Console.WriteLine("{0}{1}",(char)(cA + i), count[i]);
                          // Выводим символ и частоту
        }
    }
```

Задача 3. Задача ЕГЭ. Логины.

На вход программы подаются фамилии и имена учеников. Известно, что общее количество учеников не превосходит 100.

В первой строке вводится количество учеников, принимавших участие в соревнованиях, N. Далее следует N строк, имеющих следующий формат:

{Фамилия} {Имя}

Здесь {Фамилия} — строка, состоящая не более чем из 20 символов; {Имя} — строка, состоящая не более чем из 15 символов. При этом {Фамилия} и {Имя} разделены одним пробелом.

Примеры входных строк:

Иванова Мария

Петров Сергей

Требуется написать программу, которая формирует и печатает уникальный логин для каждого ученика по следующему правилу: если фамилия встречается первый раз, логин — это данная фамилия, если фамилия встречается второй раз, логин — это фамилия, в конец которой приписывается число 2 и т.д. Например, для входной последовательности

Иванова Мария
Петров Сергей
Бойцова Екатерина
Петров Иван
Иванова Наталья

будут сформированы следующие логины:

Иванова
Петров
Бойцова
Петров2
Иванова2

```
using System;
using System.IO;
    struct Info
    {
        public string name;
        public int count;
    }
    class Program
    {
        static void Main()
        {
            Info[] info = new Info[100];
            StreamReader sr = new StreamReader("..\\..\\data.txt");
            int N = int.Parse(sr.ReadLine());
            for (int i=0;i< N;i++)
            {
                string s = sr.ReadLine();
```

```
         string[] ss = s.Split(' ');
                int c = 1;
                for (int j=0;j< i;j++)
                    if (ss[0]==info[j].name) c++;
                info[i].name = ss[0];
                info[i].count = c;
            }
            sr.Close();
            for (int i = 0; i < N; i++)
            {
                Console.Write(info[i].name);
                if (info[i].count > 1) Console.Write(info[i].count);
                Console.WriteLine();
            }
        }
    }
```

Задача 4. Задача ЕГЭ. Олимпиады.

На вход программе подаются сведения о номерах школ учащихся, участвовавших в олимпиаде. В первой строке сообщается количество учащихся N, каждая из следующих N строк имеет формат:

<Фамилия> <Инициалы> <номер школы>

<Фамилия> — строка, состоящая не более чем из 20 символов; <Инициалы> — строка, состоящая из 4-х символов (буква, точка, буква, точка), <номер школы> — не более чем двузначный номер. <Фамилия> и <Инициалы>. <Инициалы> и <номер школы> разделены одним пробелом.

Пример файла data.txt:
6
Иванов П.С. 57
Иванова М.Т. 12
Петров С.А. 54
Бойцова Е.К. 12
Петров И.И. 33
Иванова Н.П. 10

Требуется написать как можно более эффективную программу, которая будет выводить на экран информацию, из какой школы было меньше всего участников (таких школ может быть несколько). При этом необходимо вывести информацию только по школам, пославшим хотя бы одного участника. Следует учитывать, что N>=1000.

```
using System;
using System.IO;
    struct Element
    {
        public string FIO;
        public int N;
    }
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("..\\..\\data.txt");
            int N = int.Parse(sr.ReadLine());
            Element[] a = new Element[N];
```

```
      for(int i=0;i< N;i++)
            {
                string[] s = sr.ReadLine().Split(' ');
                a[i].FIO = s[0]+s[1];
                a[i].N = int.Parse(s[2]);
            }
            sr.Close();
            int[] massiv = new int[100];
            for(int i=0;i< N;i++) massiv[a[i].N]++;
            int min = 100;
            for (int i = 0; i < 100; i++)
                if (massiv[i] != 0 && massiv[i] < min) min = massiv[i];
            for (int i = 0; i < 100; i++)
                if (massiv[i] == min) Console.WriteLine(i);
                Console.ReadKey();
        }
    }
```

Задача 5. Задача ЕГЭ. Температура за 2008.

На вход программы подается 366 строк, которые содержат информацию о среднесуточной температуре всех дней 2008 года. В каждой из строк сначала записана дата в виде dd.mm (на запись номера дня и номера месяца в числовом формате отводится строго два символа, день от месяца отделен точкой), затем через пробел записано значение температуры — число со знаком плюс или минус, с точностью до 1 цифры после десятичной точки.

Информация отсортирована по значению температуры, то есть хронологический порядок нарушен. Требуется написать программу, которая будет выводить на экран информацию о месяце (месяцах), среднемесячная температура у которого (которых) наименее отклоняется от среднегодовой. В первой строке вывести среднегодовую температуру. Найденные значения для каждого из месяцев следует выводить в отдельной строке в виде: номер месяца, значение среднемесячной температуры, отклонение от среднегодовой температуры.

Пример:

Input	**Output**
26.01 -19.1 22.01 -17.8 25.01 -17.2 24.01 -15.4 23.01 -15.3 21.01 -14.3 27.02 -13.6 12.07 24.2 ...	6.41 11 7.89 0.34

```
using System;
using System.IO;
    class Program
    {
        static void Main(string[] args)
```

```
    {
            const int DAYS = 366;                               // Кол-во дней в
2008 г.
             double[] tMonth = new double[12];          // Массив для хранения
температур по месяцам
            int i, month;
             double t, tYear = 0;                                 // Температура
за год
            StreamReader sr = new StreamReader("..\\..\\data.txt");
             for (i = 0; i < DAYS; i++)                          // Считываем файл
с датами и температурами
            {
// Считываем строчку с данными за день и заменяем в строке . на ,
            string s = sr.ReadLine().Replace('.', ',');
// Разбиваем строку по пробелу
                string[] ss = s.Split(new char[] { ' ' });
// Получаем месяц. И уменьшаем месяц на единицу(у нас массив с месяцами с 0, а
номера месяцев с 1)
                month = int.Parse(ss[0].Substring(3)) - 1;
                    t = double.Parse(ss[1]);                        // Получаем
температуру в этот день
                 tMonth[month] = tMonth[month] + t;// Подсчитываем температуру
в месяце
                  tYear = tYear + t;                                // Суммируем
температуры в году
            }
            sr.Close();
             for (i = 0; i < 12; i++)                            // Получаем
среднюю температуру в месяце
                switch (i)
                {
                    case 1:
                        tMonth[i] = tMonth[i] / 29;
                        break;
                    case 4:
                    case 6:
                    case 9:
                    case 11:
                        tMonth[i] = tMonth[i] / 30;
                        break;
                    default:
                        tMonth[i] = tMonth[i] / 31;
                        break;
                }
            tYear = tYear / DAYS;                               // Получаем
среднюю температуру за год
              double min = Math.Abs(tMonth[0] - tYear);// Принимаем начальное
минимальное значение
             for (i = 0; i < 12; i++)                                   //
Находим минимальное значение
                if (Math.Abs(tMonth[i] - tYear) < min)
                    min = Math.Abs(tMonth[i] - tYear);
// Все месяцы среднее отклонение температуры равно min, выводим на экран
            Console.WriteLine("{0:F2}", tYear);
            for (i = 0; i < 12; i++)
                if (Math.Abs(tMonth[i] - tYear) == min) Console.WriteLine("{0}
{1:F2} {2:F2}", i, tMonth[i], tMonth[i] - tYear);
        }
```

```
    }
```

Задача 6. Заявление на отпуск.

Написать программу создания заявления по шаблону. Разработать простой шаблон документа, куда вместо меняющегося текста вставить теги.

```
                                    Директору <name1>
                                    <name2>
                                    от <name3>
                                    <name4>

                  Заявление

Прошу предоставить мне ежегодный очередной оплачиваемый
отпуск с <data1> г. по <data2> г.

<data3>                                          <name5>
```

Разработать программу, которая вместо тегов подставляет пользовательские данные.

```
using System;
using System.IO;
using System.Text.RegularExpressions;
// Напишем программу "создание заявления по шаблону".
//  Разработаем  простой  шаблон  документа,  куда  вместо  меняющегося  текста
вставим теги.
// Смотри файл shablon.txt
// Создадим структуру, связывающую некоторый тег со строкой
    struct Element
    {
        public string tag;
        public string str;
// Создадим конструктор, чтобы проще было заполнить массив
        public Element(string tag, string newString)
        {
            this.tag =tag;
            str = newString;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
// Считаем текст из файла
            string s = File.ReadAllText("shablon.txt");
// Заполним массив элементов, сопоставив тегам соответствующий текст
                  Element[]  e  =  new  Element[8]  {  new  Element("name1","ООО
'Дружба'"),
                                        new Element("name2","Иванову И.И."),
                                             new Element("name3","менеджера по
продажам"),
                                        new Element("name4","Сидорова А.А."),
                                        new Element("name5","Сидоров А.А."),
```

```
                                        new Element("data1","01.06.16"),
                                        new Element("data2","14.06.16"),
                                        new Element("data3","20.04.16"),

        };
// Пробежимся по массиву и поменяем все вхождения тегов на текст
            foreach(var el in e)
            {
                Regex reg = new Regex("<"+el.tag+">");
                s=reg.Replace(s, el.str);
            }
            Console.WriteLine(s);
            Console.ReadLine();
        }
    }
```

Домашнее задание

1. Создать программу, которая будет проверять корректность ввода логина. Корректным логином будет строка от 2 до 10 символов, содержащая только буквы латинского алфавита или цифры, при этом цифра не может быть первой:
   а) без использования регулярных выражений;
   б) **с использованием регулярных выражений.
2. Разработать статический класс **Message**, содержащий следующие статические методы для обработки текста:
   а) Вывести только те слова сообщения, которые содержат не более n букв.
   б) Удалить из сообщения все слова, которые заканчиваются на заданный символ.
   в) Найти самое длинное слово сообщения.
   г) Сформировать строку с помощью StringBuilder из самых длинных слов сообщения.
   д) ***Создать метод, который производит частотный анализ текста. В качестве параметра в него передается массив слов и текст, в качестве результата метод возвращает сколько раз каждое из слов массива входит в этот текст. Здесь требуется использовать класс Dictionary.

3. *Для двух строк написать метод, определяющий, является ли одна строка перестановкой другой.
   Например:
   badc являются перестановкой **abcd**.
4. *Задача ЕГЭ.

На вход программе подаются сведения о сдаче экзаменов учениками 9-х классов некоторой средней школы. В первой строке сообщается количество учеников N, которое не меньше 10, но не превосходит 100, каждая из следующих N строк имеет следующий формат:

<Фамилия> <Имя> <оценки>,

где <Фамилия> — строка, состоящая не более чем из 20 символов, <Имя> — строка, состоящая не более чем из 15 символов, <оценки> — через пробел три целых числа, соответствующие оценкам по пятибалльной системе. <Фамилия> и <Имя>, а также <Имя> и <оценки> разделены одним пробелом. Пример входной строки:

Иванов Петр 4 5 3

Требуется написать как можно более эффективную программу, которая будет выводить на экран фамилии и имена трёх худших по среднему баллу учеников. Если среди остальных есть ученики, набравшие тот же средний балл, что и один из трёх худших, следует вывести и их фамилии и имена.

Достаточно решить 2 задачи. Старайтесь разбивать программы на подпрограммы. Переписывайте в начало программы условие и свою фамилию. Все программы сделать в одном решении. Для решения задач используйте неизменяемые строки (string)

Дополнительные материалы

1. Демоверсии ГИА по информатике.
2. Дистанционная подготовка к олимпиадам по информатике.
3. Демоверсии ЕГЭ по информатике.
4. Регулярные выражения в C#. Класс Regex.
5. Регулярные выражения в C#.
6. regexpal.com.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Гойвертс Я., Левитан С. Регулярные выражения. Сборник рецептов. — СПб: Символ-Плюс, 2009.
2. Павловская Т.А. Программирование на языке высокого уровня. — СПб: Питер, 2009.
3. Петцольд Ч. Программирование на C#. Т.1. — М.: Русская редакция, 2001.
4. Шилдт Г. C# 4.0. Полное руководство. — М.: ООО «И.Д. Вильямс», 2011.
5. MSDN.