Символы и строки

символы char,
неизменяемые строки string,
изменяемые строки StringBilder,
регулярные выражения Regex

Символы char

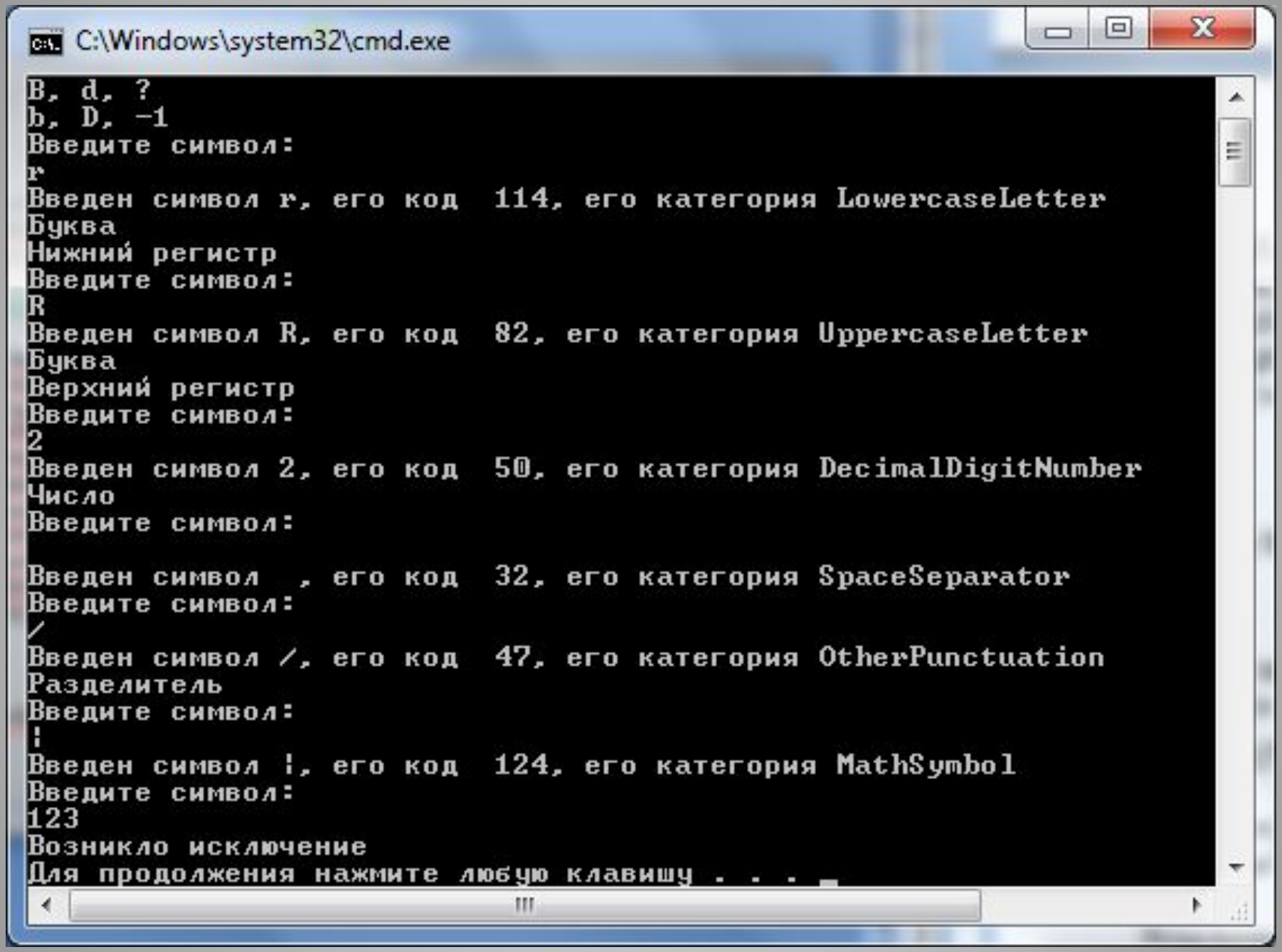
- Символьный тип char предназначен для хранения символа в кодировке Unicode.

- Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .Net из пространства имен System.

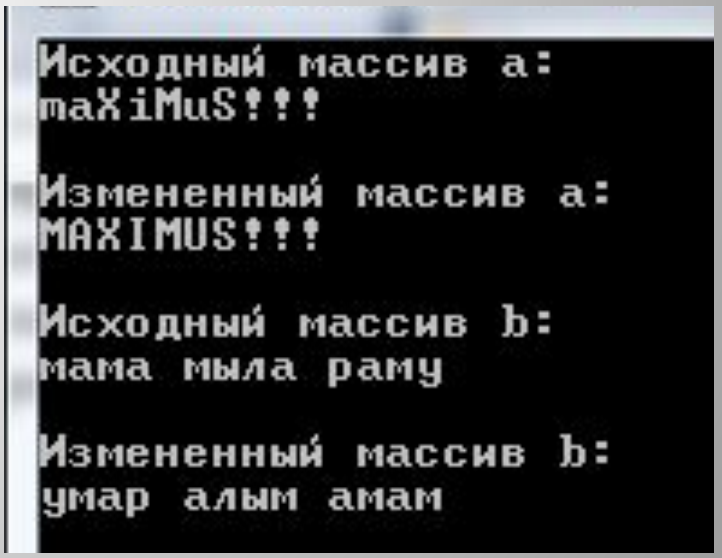
Основные методы

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

```
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 выполнятеся до тех пор, 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");
      }
  }
```


C:\Windows\system32\cmd.exe
B, d, ?
b, D, -1
Введите символ:
r
Введен символ r, его код 114, его категория LowercaseLetter
Буква
Нижний регистр
Введите символ:
R
Введен символ R, его код 82, его категория UppercaseLetter
Буква
Верхний регистр
Введите символ:
2
Введен символ 2, его код 50, его категория DecimalDigitNumber
Число
Введите символ:
Введен символ , его код 32, его категория SpaceSeparator
Введите символ:
/
Введен символ /, его код 47, его категория OtherPunctuation
Разделитель
Введите символ:
!
Введен символ !, его код 124, его категория MathSymbol
Введите символ:
123
Возникло исключение
Для продолжения нажмите любую клавишу . . .

```
//Организация массива символов и работа с ним на основе базового класса Array:
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);
    }

static void PrintArray(string line, Array a)
    {
        Console.WriteLine(line);
        foreach( object x in a) Console.Write(x);
        Console.WriteLine('\n');
    }
```



Неизменяемые строки string

- Тип string, предназначенный для работы со стоками символов в кодировке Unicode.
- Ему соответствует базовый тип класса System.String библиотеки .Net.
- Каждый объект string - это неизменяемая последовательность символов Unicode, т.е. методы, предназначенные для изменения строк, возвращают измененные копии, исходные же строки остаются неизменными.

Создание строк

string s;	**инициализация отложена**
string s="кол около колокола";	инициализация строковым литералом
string s=@"Привет! Сегодня хорошая погода!!! "	символ @ сообщает конструктору string, что строку нужно воспринимать буквально, даже если она занимает несколько строк
string s=new string (' ', 20);	конструктор создает строку из 20 пробелов
int x = 12344556; string s = x.ToString();	инициализировали целочисленную переменную преобразовали ее к типу string
char [] a={'a', 'b', 'c', 'd', 'e'}; string v=new string (a);	создали массив символов создание строки из массива символов
char [] a={'a', 'b', 'c', 'd', 'e'}; string v=new string (a, 0, 2)	создание строки из части массива символов, при этом: 0 показывает с какого символа, 2 – сколько символов использовать для инициализации

Методы работы со строками

Название	*Вид*	*Описание*
Compare	Статический метод	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с учетом или без учета регистра.
CompareTo	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, LastIndexOf, LastIndexOfAny	Экземплярные методы	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора в данную строку.
Insert	Экземплярный 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	Экземплярные методы	Выравнивают строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки.
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новыми подстрокой
Split	Экземплярный метод	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк.
StartWith, EndWith	Экземплярные методы	Возвращают true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой.
Substring	Экземплярный метод	Выделение подстроки, начиная с заданной позиции
ToCharArray	Экземплярный метод	Преобразует строку в массив символов
ToLower, ToUpper	Экземплярный методы	Преобразование строки к нижнему или верхнему регистру
Trim, TrimStart, TrimEnd	Экземплярные методы	Удаление пробелов в начале и конце строки или только с одного ее конца.

- вызов статических методов происходит через обращение к имени класса:

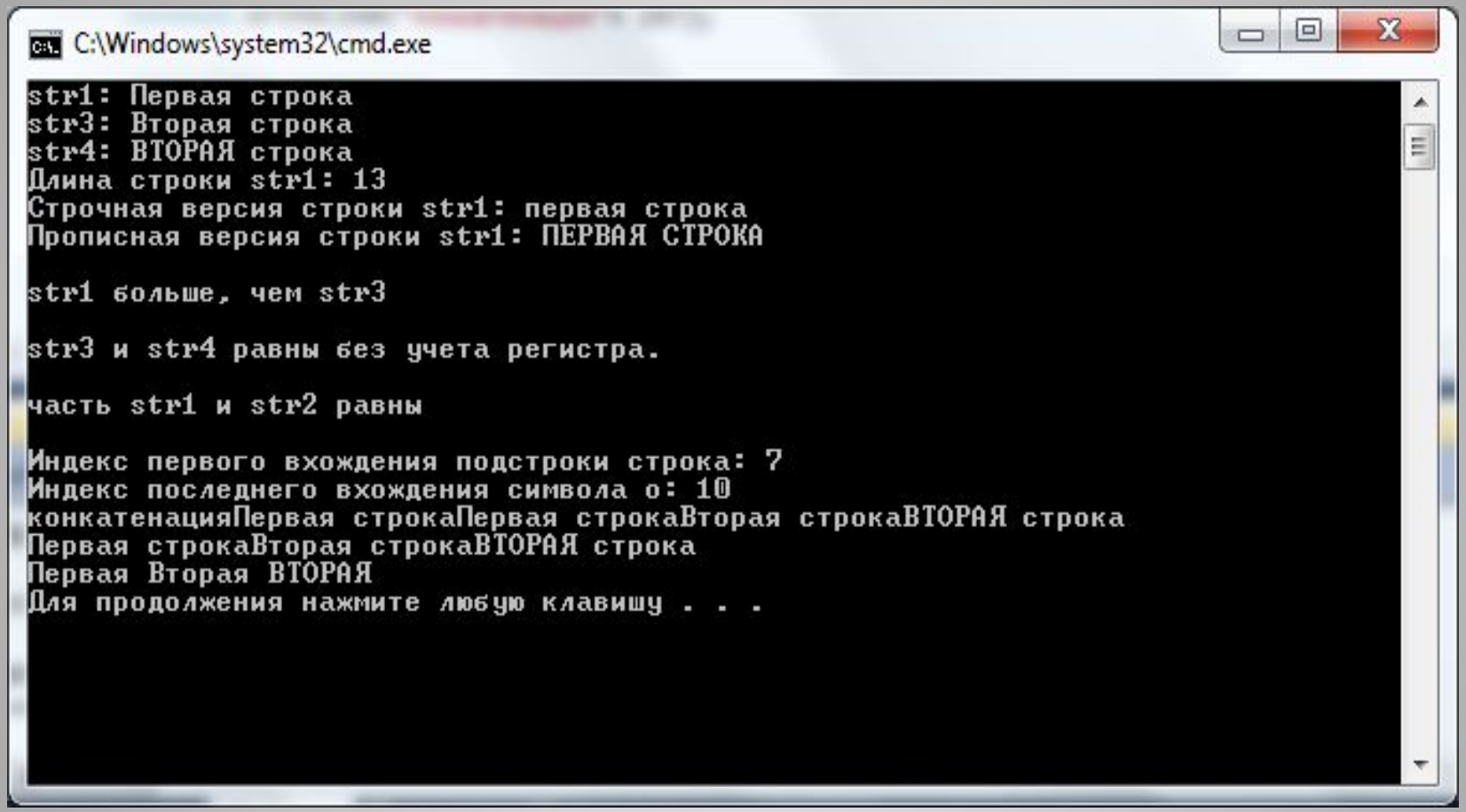
String.Concat(str1, str2)

- в остальных случаях через обращение к экземплярам класса:

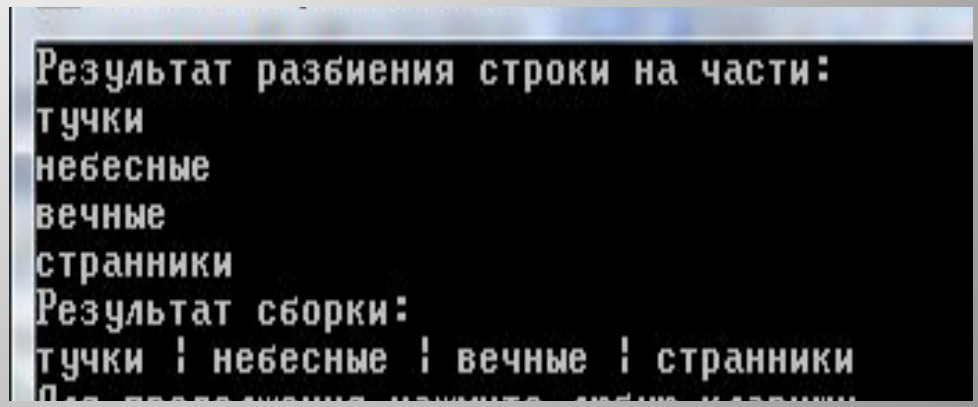
str.ToLower()

```
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("str3: " + str3);
         Console.WriteLine("str4: " + str4);
         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 строка: " + idx);
idx = str2.LastIndexOf("о");
Console.WriteLine("Индекс последнего вхождения символа о: " + idx);
//конкатенация
string str = String.Concat(str1, str2, str3, str4);
Console.WriteLine("конкатенация"+ 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);
}   }            }
```


C:\Windows\system32\cmd.exe
str1: Первая строка
str3: Вторая строка
str4: ВТОРАЯ строка
Длина строки str1: 13
Строчная версия строки str1: первая строка
Прописная версия строки str1: ПЕРВАЯ СТРОКА
str1 больше, чем str3
str3 и str4 равны без учета регистра.
часть str1 и str2 равны
Индекс первого вхождения подстроки строка: 7
Индекс последнего вхождения символа о: 10
конкатенацияПервая строкаПервая строкаВторая строкаВТОРАЯ строка
Первая строкаВторая строкаВТОРАЯ строка
Первая Вторая ВТОРАЯ
Для продолжения нажмите любую клавишу . . .

```
class Program
   {
  //  методы разделения строки на элементы Split и слияние массива строк в единую
      строку Join

static void Main()
  {
string poems = "тучки небесные вечные странники";
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]);
// Теперь собираем эти части в одну строку, в качестве разделителя используем символ |
string whole = String.Join(" | ", parts);
Console.WriteLine("Результат сборки: ");
Console.WriteLine(whole);
  }   }
```

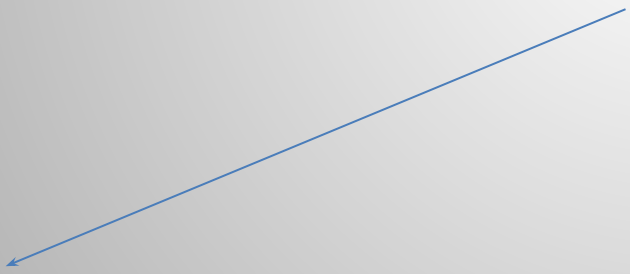


При работе с объектами класса string нужно учитывать их свойство неизменяемости, т.е. методы изменяют не сами строки, а их копии.

```
string a="";
for (int i = 1; i <= 100; i++) a +="!";
Console.WriteLine(a);
```

В этом случае в памяти компьютера будет сформировано 100 различных строк вида:

```
!
!!
!!!
…
!!!...!!
```



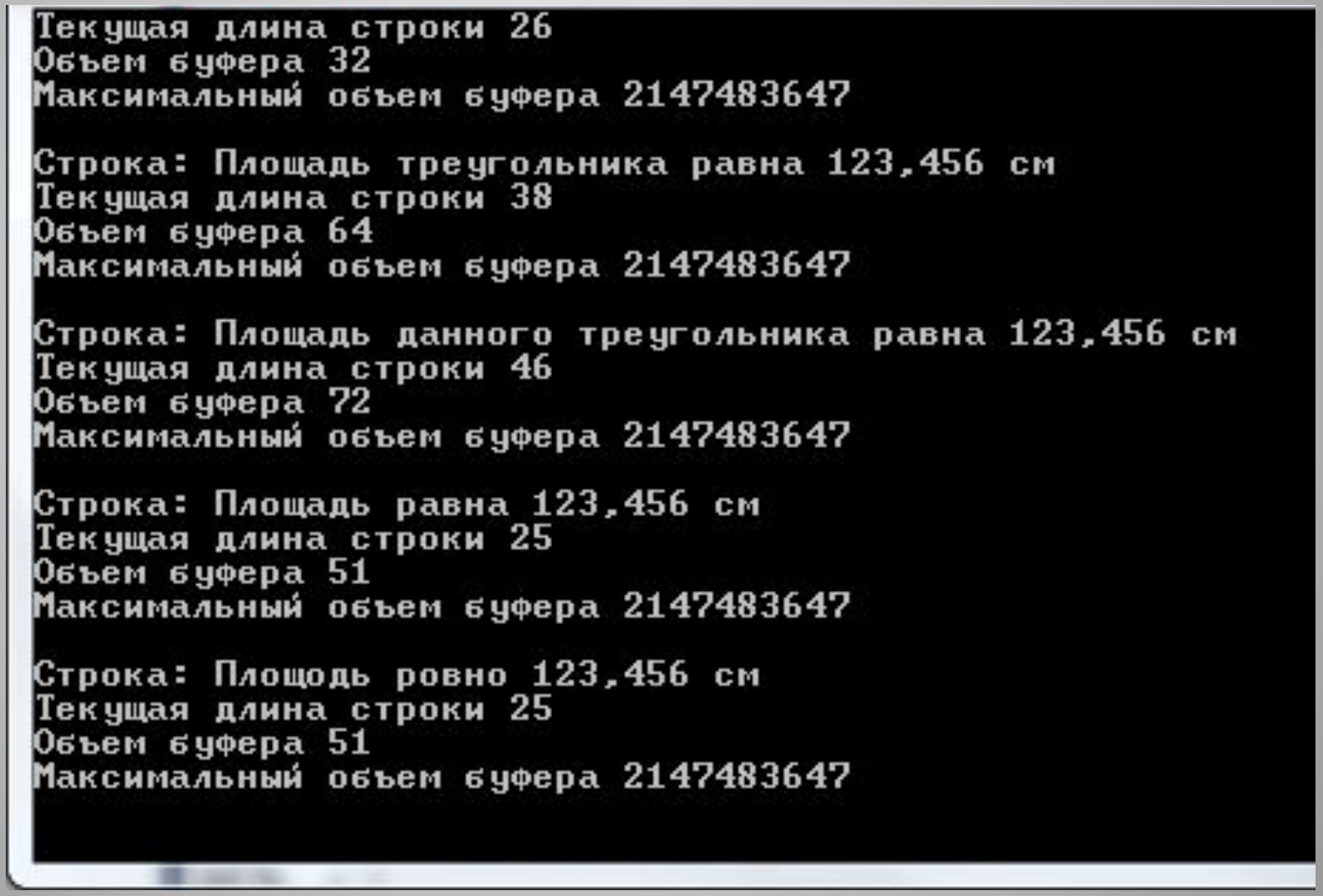
Только последняя строка будет храниться в переменной a.
Ссылки на все остальные строчки будут потеряны, но эти строки будут храниться в памяти компьютера и засорять память. Бороться с таким засорением придется сборщику мусора, что будет сказываться на производительности программы, поэтому если нужно изменять строку, то лучше пользоваться классом **StringBuilder.**

Изменяемые строки

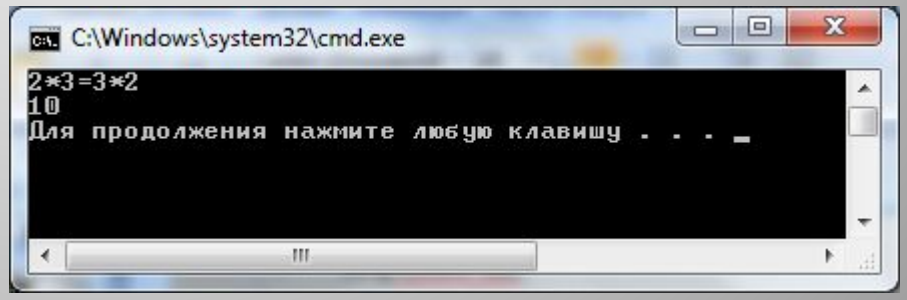
- Чтобы создать строку, которую можно изменять, в C# предусмотрен класс ***StringBuilder***, определенный в пространстве имен ***System.Text***. Объекты этого класса всегда объявляются с явным вызовом конструктора класса (через операцию new) .

StringBuilder a =new StringBuilder();	**инициализация строки и выделение необходимой памяти**
StringBuilder d = new StringBuilder("abcd", 100);	инициализация строки и выделение памяти под 100 символов
StringBuilder b = new StringBuilder("abcd");	создание пустой строки, размер по умолчанию 16 символов
StringBuilder c = new StringBuilder(100);	создание пустой строки и выделение памяти под 100 символов
StringBuilder d = new StringBuilder("abcd", 1, 3,100);	инициализация подстрокой "bcd", и выделение памяти под 100 символов

```
static void Main()    {
            try          {
            StringBuilder str=new StringBuilder("Площадь"); PrintString(str);
            str.Append(" треугольника равна");          PrintString(str);
            str.AppendFormat(" {0} см ", 123.456);          PrintString(str);
            str.Insert(8, "данного ");              PrintString(str);
            str.Remove(7, 21);                      PrintString(str);
            str.Replace("a", "o");                          PrintString(str);
            StringBuilder str1=new StringBuilder(Console.ReadLine());
            StringBuilder str2=new StringBuilder(Console.ReadLine());
            Console.WriteLine(str1.Equals(str2));
            }
            catch               {
Console.WriteLine("Вознико 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();
      }
```


Текущая длина строки 26
Объем буфера 32
Максимальный объем буфера 2147483647
Строка: Площадь треугольника равна 123,456 см
Текущая длина строки 38
Объем буфера 64
Максимальный объем буфера 2147483647
Строка: Площадь данного треугольника равна 123,456 см
Текущая длина строки 46
Объем буфера 72
Максимальный объем буфера 2147483647
Строка: Площадь равна 123,456 см
Текущая длина строки 25
Объем буфера 51
Максимальный объем буфера 2147483647
Строка: Площодь ровно 123,456 см
Текущая длина строки 25
Объем буфера 51
Максимальный объем буфера 2147483647

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication2
{
   class Program
   {

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

      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);
      }

   }
}
```



Регулярные выражения

Регулярное выражение – это шаблон, по которому выполняется поиск соответствующего фрагмента текста.

1. Эффективный поиск в тексте по заданному шаблону;
2. Редактирование текста;
3. Формирование итоговых отчетов по результатам работы с текстом.

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.

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.	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. Можно исп. 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.	Выражение c\wt соответствует фрагментам: cbt, cct, c2t и т.д., но не соответствует фрагментам c%t, c{t и т.д.
\W	Любой не алфавитно-цифровой символ.	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.	Выражение \s\w\w\w\s соответствует любому слову из трех букв, окруженному пробельными символами.
\S	Любой не пробельный символ.	Выражение \s\S\S\S\s соответствует любым трем непробельным символам, окруженным пробельными.
\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.

Кроме метасимволов, обозначающие классы символов, могут применяться уточняющие метасимволы:

Уточняющие символы	***Описание***
^	Фрагмент, совпадающий с регулярными выражениями, следует искать только в начале строки
$	Фрагмент, совпадающий с регулярными выражениями, следует искать только в конце строки
\A	Фрагмент, совпадающий с регулярными выражениями, следует искать только в начале многострочной строки
\Z	Фрагмент, совпадающий с регулярными выражениями, следует искать только в конце многострочной строки
\b	Фрагмент, совпадающий с регулярными выражениями, начинается или заканчивается на границе слова, т.е. между символами, соответствующими метасимволам \w и \W
\B	Фрагмент, совпадающий с регулярными выражениями, не должен встречаться на границе слов

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, перед строкой необходимо ставить символ **@.**

- Символ **@** можно не ставить, если в качестве шаблона используется шаблон без метасимволов.

- 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 просто точку.

Примеры регулярных выражений:

слово rus – @"rus" или"rus«

№ телефона в формате xxx-xx-xx –
@"\d\d\d-\d\d-\d\d" или @"\d{3}(-\d\d){2} "

номер автомобиля - @"[A-Z]\d{3}[A-Z]{2}\d{2,3}RUS"

Поиск в тексте по шаблону

- Пространство имен библиотеки базовых классов **System.Text.RegularExpressions** содержит все объекты платформы .NET Framework, имеющие отношение к регулярным выражениям.

- Класс, поддерживающий регулярные выражения, - класс **Regex**, он представляет неизменяемые откомпилированные регулярные выражения.

- Для описания регулярного выражения в классе определено несколько перегруженных конструкторов:

1) **Regex()** – создает пустое выражение;

2) **Regex(String)** – создает заданное выражение;

3) **Regex(String, RegexOptions)** – создает заданное выражение и задает параметры для его обработки с помощью элементов перечисления **RegexOptions** (например, различать или нет прописные и строчные буквы).

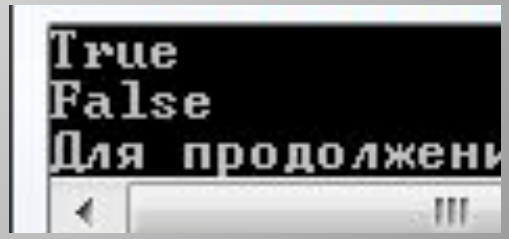
- Поиск фрагментов строки, соответствующих заданному выражению, выполняется с помощью методов класса Regex:

- **IsMach,**
- **Mach,**
- **Matches.**

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

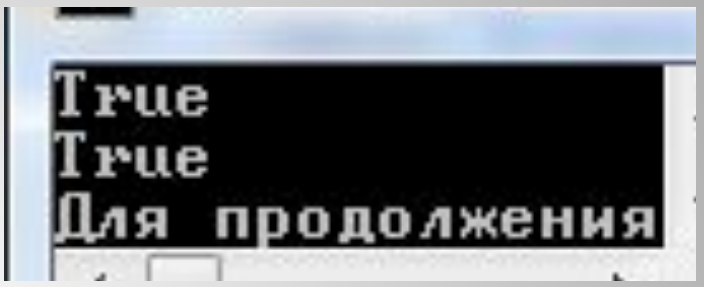
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program
   {
      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));       }

   }          }
```

RegexOptions.IgnoreCase – означает, что регулярное выражение применяется без учеба регистра символов



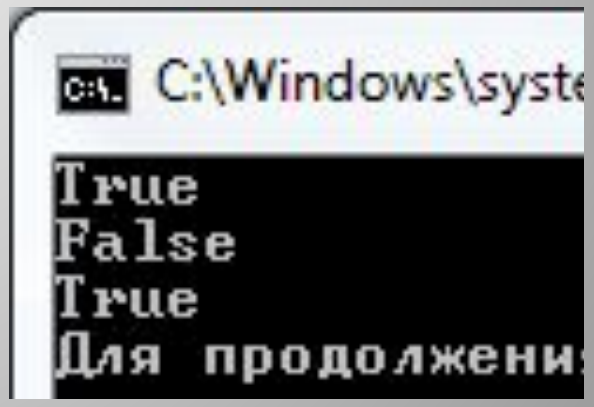
/* Можно использовать конструкцию выбора из нескольких элементов.
Варианты выбора перечисляются через вертикальную черту*/

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program
   {
   static void Main(string[] args)
      {
         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));
      }   }   }
```

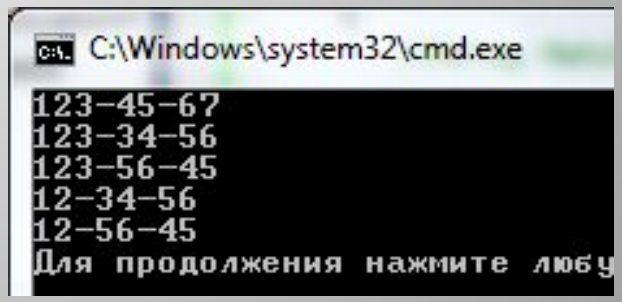


//есть ли в заданных строках номера телефона в формате xx-xx-xx или xxx-xx-xx:

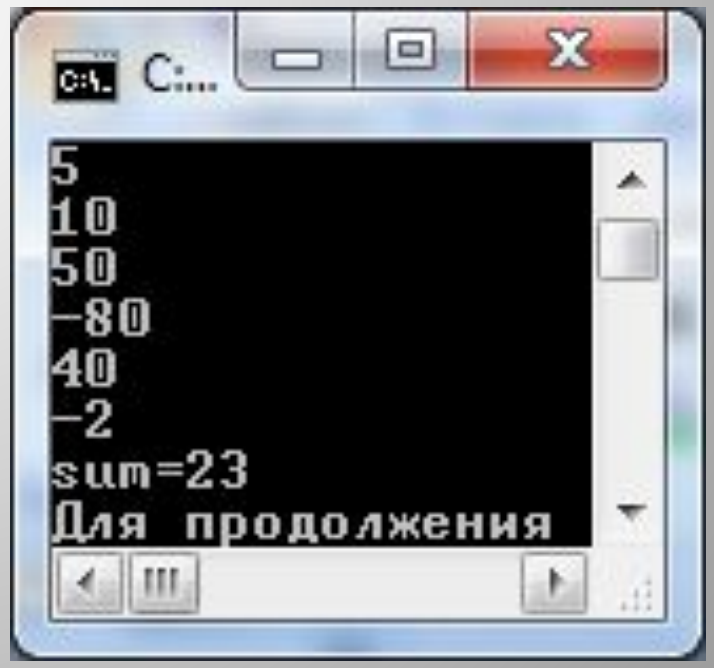
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program
   {
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));
       }   } }
```



```
/*Метод Match класса Regex не просто определяет, содержится ли текст,
    соответствующий шаблону, а возвращает объект класса Match –
    последовательность фрагментов текста, совпавших с шаблоном*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program    {
static void Main()       {
        Regex r = new Regex(@"\d{2,3}(-\d\d){2}");
        string text = @"Контакты в Минске tel:123-45-67, 123-34-56; fax:123-56-45
                        Контакты в Бресте tel:12-34-56; fax:12-56-45";
        Match tel = r.Match(text);
        while (tel.Success)
        {
           Console.WriteLine(tel);
           tel = tel.NextMatch();          }
     }   }   }
```

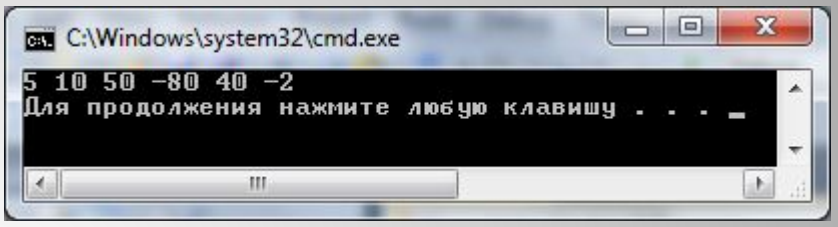


```
//подсчитать сумму целых чисел, встречающихся в тексте:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3{
   class Program    {
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.WriteLine(teg);
             sum += int.Parse(teg.ToString());
             teg = teg.NextMatch();
          }
          Console.WriteLine("sum=" + sum);
    }   }    }
```



/*Метод Matches класса Regex возвращает объект класса MatchCollection – коллекцию всех фрагментов заданной строки, совпавших с шаблоном. При этом метод Matches многократно запускает метод Match, каждый раз начиная поиск с того места, на котором закончился предыдущий поиск*/

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program
   {
static void Main(string[] args)
{
        string text = @"5*10=50 -80/40=-2";
        Regex theReg = new Regex(@"[-+]?\d+");
        MatchCollection theMatches = theReg.Matches(text);
        foreach (Match theMatch in theMatches)
{       Console.Write("{0} ", theMatch.ToString());     }
        Console.WriteLine();        }
}   }
```

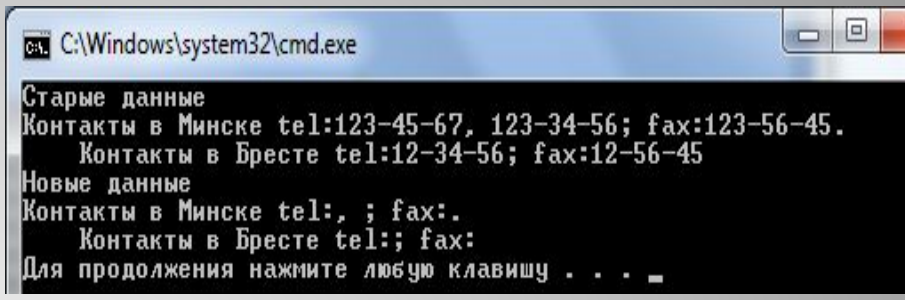


Редактирование текста

```
// метод Replace класса Regex позволяет выполнять замену одного фрагмента
    текста другим или удаление фрагментов текста:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3    {
   class Program    {

      static void Main(string[] args)      {
         string text = @"Контакты в Минске tel:123-45-67, 123-34-56; fax:123-56-45.
                Контакты в Бресте tel:12-34-56; fax:11-56-45";
         Console.WriteLine("Старые данные\n" + text);
         string newText = Regex.Replace(text, "123-", "890-");
         Console.WriteLine("Новые данные\n" + newText);
      }
   }
}
```

```
Старые данные
Контакты в Минске tel:123-45-67, 123-34-56; fax:123-56-45.
                Контакты в Бресте tel:12-34-56; fax:11-56-45
Новые данные
Контакты в Минске tel:890-45-67, 890-34-56; fax:890-56-45.
                Контакты в Бресте tel:12-34-56; fax:11-56-45
Для продолжения нажмите любую клавишу . . .
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program    {
     // Удаление всех номеров телефонов из текста:

static void Main()
{
string text = @"Контакты в Минске tel:123-45-67, 123-34-56; fax:123-56-45.
   Контакты в Бресте 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);
}
}
      }
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication3
{
   class Program
   {
     //Разбиение исходного текста на фрагменты:

static void Main()
{
string text = @"Контакты в Минске tel:123-45-67, 123-34-56; fax:123-56-45.
                Контакты в Бресте tel:12-34-56; fax:12-56-45";
string []newText=Regex.Split(text,"[ ,.:;]+");
foreach( string a in newText)
Console.WriteLine(a);
}
}
      }
```