# Строки в С#

## Строковые и буквальные строковые литералы

Для представления текстовой информации в С# используются объекты класса string. Класс string представляет собой один из предопределенных типов языка С#. В .Net Framework этому типу соответствует класс System.String. Один из видов объектов класса string мы уже многократно применяли — это строковые константы, или строковые литералы.

**Строковая константа, или строковый литерал, имеет две формы:**

* обычный (регулярный) строковый литерал (regu­lar-string-literal);
* буквальный строковый литерал (verbatim-string-literal).

**Регулярный строковый литерал** - это последователь­ность символов и эскейп-последователъностей, заключенная в кавычки (не в апострофы). Как и в других языках на базе С, в С# строковые литералы могут содержать различные управляющие последовательности символов (characters), которые позволяют уточнять то, как символьные данные должны выводиться в выходном потоке.

Начинается каждая такая управляющая последовательность с символа обратной косой черты, за которым следует интерпретируемый знак.

|  |  |
| --- | --- |
| Управляющая последовательность | Описание |
| \’ | Вставляет в строковый литерал символ одинарной кавычки |
| \” | Вставляет в строковый литерал символ двойной кавычки |
| \\ | Вставляет в строковый литерал символ обратной косой черты. Может  быть полезной при определении путей к файлам и сетевым ресурсам |
| \a | Заставляет систему выдавать звуковой сигнал, который в консольных приложениях может служить своего рода звуковой подсказкой пользователю |
| \n | Вставляет символ новой строки (на платформах Windows) |
| \t | Вставляет в строковый литерал символ горизонтальной табуляции |

Обрабатывая регулярный строковый литерал, компи­лятор из его символов формирует строковый объект и при этом заменяет эскейп-последовательности соответствующими кодами (символов или управляющих кодов). Например, литералу

"\u004F\x4E\u0045\ttwo"

будет соответствовать строка, при выводе которой на экране текст появится в таком виде:

ONE two

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \u004F | \x4E | u0045 | \t | two |
| юникод символа 'О', | шестнадцатеричный код символа 'N', | юникод символа 'Е', | эскейп-последовательность, представляющая код табуляции. |  |

**Буквальный (дословный) строковый литерал** начинается с префикса @, за которым в кавычках размещается последова­тельность символов. Символы такого литерала воспринимаются буквально, т.е. в такой строке не обрабатываются эскейп-после-довательности, а каждый символ воспринимается как таковой. В результате выполнения оператора:

Console.WriteLine(@"\u004F\x4E\u0045\ttwo");

на экране появится \u004F\x4E\u0045\ttwo

Если в буквальном литерале необходимо поместить кавычку, то она изображается двумя рядом стоящими кавычками.

Буквальный литерал может быть размещен в коде программы на нескольких строках, и это раз­мещение сохраняется при его выводе.

Console.WriteLine(@"1. Создать массив.

2. Печать массива.

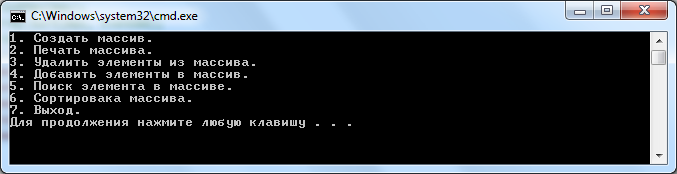
3. Удалить элементы из массива.

4. Добавить элементы в массив.

5. Поиск элемента в массиве.

6. Сортировака массива.

7. Выход.");

****

Использовать ‘\n’ для перехода на новую строку в буквальном литерале нельзя..

## Ссылки типа string

Каждый строковый литерал — это объект класса (типа) string.

string stroka;

Класс string является ссылочным типом. Кроме литералов, можно опре­делить объекты класса string с использованием конструкторов. (Конструктор - специальный метод класса, предназначенный для инициализации объекта класса в процессе его создания.) Конструкторы класса string позволяют инициализировать объ­екты-строки несколькими способами.

string str1="Это строка 1";

char []charArr={'M','a','c','c','и','в'};

string str2=new string(charArr);

string str3=new string('S',5);

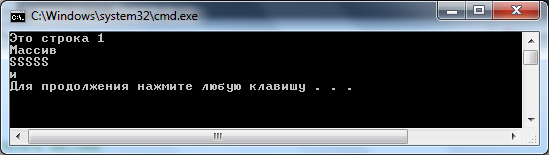
string str4 = new string(charArr, 4, 1);

Console.WriteLine(str1);

Console.WriteLine(str2);

Console.WriteLine(str3);

Console.WriteLine(str4);



Строковые объекты, как создаваемые с применени­ем конструкторов, так и формируемые для представле­ния строковых литералов, компилятор размещает в динамической памяти. Ссылки на строки размещаются в стеке. Размер строки при определении строкового объекта явно не указывает­ся, он определяется автоматически при инициализации. **Ни размер строки, ни ее содержимое не могут изменяться после создания строки!!**

Если инициализация при объявлении строковой ссылки от­сутствует, то ей присваивается значение null, и ее нельзя исполь­зовать в выражениях до присваивания ей конкретного значения. Пример ошибки:

string s;

s+=”abcd”;//ошибка компиляции

Кроме явного задания строковых литералов и применения конструкторов для создания строковых объектов, используют метод ToString(). Этот метод определен для всех встроенных типов.

**Примеры:**

242.ToString()// результат строка”242”

После выполнения операторов:

bool b= 5>4;

string sbool =b.ToString();/\* значением sbool будет строка "True"\*/

## Операции над строками

Строки языка С# предназначены для хранения последова­тельностей символов, для каждого из которых отводится 2 байта, и они хранятся в кодировке Unicode (как данные типа char). В некотором смысле строка подобна одномерному массиву с эле­ментами типа char. Элементы (символы строки) последователь­но нумеруются, начиная с 0. Последний символ имеет номер на 1 меньше длины строки.

1. **операция индексирова­ния**:

строка[индекс],индекс – целое число,>0.

Результат выражения с операцией индексирования - символ (значение типа **char),** размещенный в той позиции строки, номер которой соответствует индексному выражению. Если значение индекса меньше нуля, а также больше или равно длине строки, возникает исключительная ситуация (генерируется исключение).

1. **Операция присваивания (=)** для строк выполняется не так как для массивов. Когда ссылке с типом массива присваива­ется значения ссылки на другой уже существующий массив, изменяет только значение ссылки. Массив, как объект, стано­вится доступен для нескольких ссылок. Т.е. адрес первого элемента массива хранится в нескольких переменных ссылочного типа.

Операция присваивания для строк приводит к созданию нового экземпляра той строки, на которую ссылается выражение справа от знака операции =. Ранее существовавшая строка никак не ассоциируется с новой ссылкой

Пример:

char [] charArr={'м','а','с','с','и','в','1'};

char[] newArr=charArr;//тот же адрес и та же строка

newArr[6]='2';

Console.WriteLine(charArr);

Console.WriteLine(newArr);

string firstString="Cтрока1";

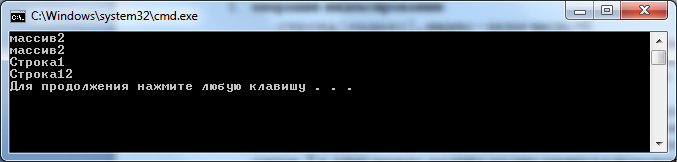
string secondString=firstString;//новый адрес и новая строка

//secondString[6] = '2';//Error

secondString = secondString + '2';

Console.WriteLine(firstString);

Console.WriteLine(secondString);



1. **Операции сравнения**на равенство === и неравенство !=, применя­емые к строкам, сравнивают последовательности символов в строках. (Для массивов сравниваются значения ссылок.)

char[] charArr = { 'м', 'а', 'с', 'с', 'и', 'в', '1' };

char[] newArr = charArr;//тот же адрес и та же строка

newArr[6] = '2';

Console.WriteLine(charArr);

Console.WriteLine(newArr);

bool ok = charArr == newArr;

Console.WriteLine(ok);

string firstString="Cтрока1";

string secondString=firstString;//новый адрес и новая строка

bool ok = firstString == secondString;

Console.WriteLine(ok);

//secondString[6] = '2';//Error

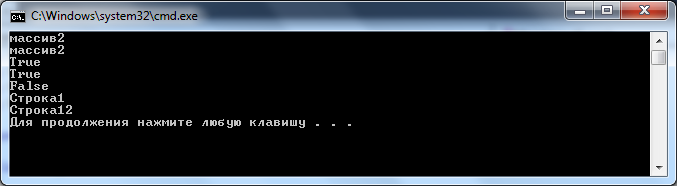
secondString = secondString + '2';

ok = firstString == secondString;

Console.WriteLine(ok);

Console.WriteLine(firstString);

Console.WriteLine(secondString);



**4. Сцепление (конкатенацию)** строк выполняет операция **+**.

**Примеры:**

Console.WriteLine(12 + 34);

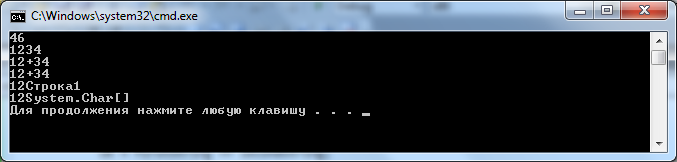
Console.WriteLine("12" + "34");

Console.WriteLine("12" + "+" + "34");

Console.WriteLine(12+"+"+34);

Console.WriteLine("12" + firstString);

Console.WriteLine("12" + newArr);

****

### Методы и свойства класса String

* **int Length() –** свойство,позволяющее получить длину (количество символов) конкретной строки (объекта класса **string).**
* **int CompareTo()** - метод, который сравнивает две строки и возвращает целочисленное значение. Для двух строк **S1,** S2 результат положительный, если S1>S2, отрицательный, если **S1<S2,** и нулевой, если SI *==* S2. Сравнение строк выполняется лексикографически.
* **static string Concat()** - метод (их несколько) выполняет кон­катенацию строк-параметров. Аргументов-строк может быть два, три или произвольное количество.
* **static string Copy()** — статический метод возвращает копию существующей строки.
* **static string Format()** - статический метод, формирующий строку на основе набора параметров.
* **int IndexOf()** — нестатический метод поиска в вызывающей строке подстроки, заданной параметром. Возвращает индекс или -1, если поиск неудачен. Поиск - с начала строки.
* **string Insert()** - нестатический метод для вставки строки-параметра в копию вызывающей строки с позиции, заданной дополнительным параметром.
* **static string Join()** - статический метод, объединяющий в одну строку строки массива-параметра. Первый параметр типа **string** задает разделитель, которым будут отделены друг от друга в результирующей строке элементы массива.
* **int LastIndexOf()** - нестатический метод поиска в вызы­вающей строке подстроки, заданной параметром. Возвращает индекс или -1, если поиск неудачен. Поиск с конца строки.
* **string Remove() -** удаляет символы из копии строки.
* **string Replace()** - заменяет символы в копии строки.
* **string [] Split()** - формирует массив строк из фрагментов вызывающей строки. Параметр типа **char** задает разделители, которыми в строке разделены фрагменты.
* **char [] ToCharArray()** — копирует символы вызывающем строки в массив типа **сhаг[].**
* **string Trim()** - удаляет вхождение заданных символом (например, пробела) в начале и в конце строки.
* **string Substring()** — выделяет из строки подстроку. Параметры задают начало и длину выделяемой части строки.

**Пример 1.**

Console.WriteLine("введите 1-ю строку ");

string str1 = Console.ReadLine();

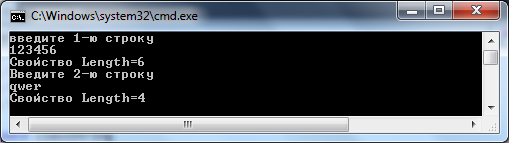
Console.WriteLine("Свойство Length="+str1.Length);

Console.WriteLine("Введите 2-ю строку");

string str2 = Console.ReadLine();

Console.WriteLine("Свойство Length=" + str2.Length);

Console.ReadLine();

****

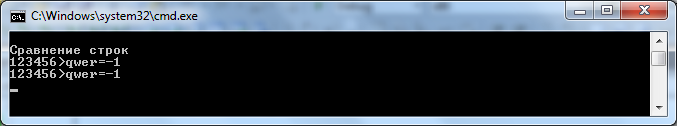
**Пример 2.**

Console.WriteLine("Сравнение строк");

Console.WriteLine("{0}>{1}={2}",str1,str2,str1.CompareTo(str2));

Console.WriteLine("{0}>{1}={2}", str1, str2, String.Compare(str1, str2));

Console.ReadLine();

****

****

****

**Пример 3.**

Console.WriteLine("Сцепление строк");

Console.WriteLine("{0}>{1}={2}", str1, str2, String.Concat(str1, str2));

Console.ReadLine();

****

**Пример 4**

Console.WriteLine("Копирование строк");

string str = String.Copy(str1);

Console.WriteLine("str=" + str);

Console.ReadLine();

****

**Пример 5**

Console.WriteLine("Поиск в строке");

Console.WriteLine("Ищем {0} в {1}={2} ",str2, str1,str1.IndexOf(str2));

Console.WriteLine("Ищем {0} в {1}={2} ",str, str1,str1.IndexOf(str));

Console.ReadLine();

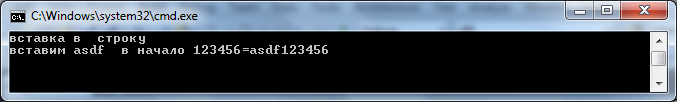


**Пример 6**

Console.WriteLine("вставка в строку");

Console.WriteLine("вставим {0} в начало {1}={2} ", str1,str2, str2.Insert(0,str1));

Console.ReadLine();

****

**Пример 7**

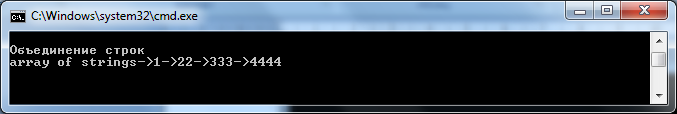
Console.WriteLine("Объединение строк");

string []Mas1={"array of strings","1","22","333","4444",};

str=String.Join("->",Mas1);

Console.WriteLine(str);

Console.ReadLine();

****

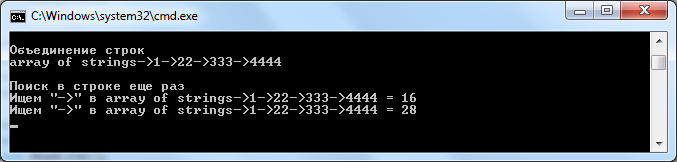
**Пример 8**

Console.WriteLine("Поиск в строке еще раз");

Console.WriteLine("Ищем \"->\" в {0} = {1} ",str, str.IndexOf("->"));

Console.WriteLine("Ищем \"->\" в {0} = {1} " , str, str.LastIndexOf("->"));

Console.ReadLine();

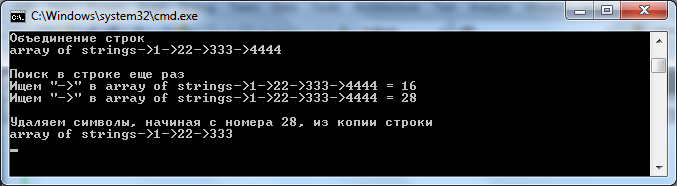
****

**Пример 9**

Console.WriteLine("Удаляем символы из копии строки");

Console.WriteLine(str.Remove(28));

Console.ReadLine();

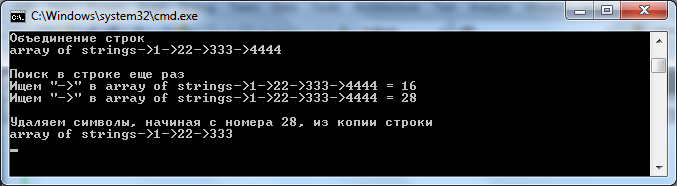


**Пример 10**

Console.WriteLine("Заменяем символ \'-\' на \'<\' из копии строки");

Console.WriteLine(str.Replace('-','<'));

Console.ReadLine();

****

**Пример 11**

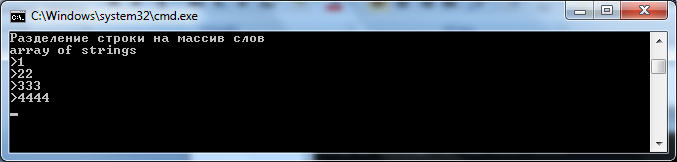
Console.WriteLine("Разделение строки на массив слов");

Mas1 = str.Split('-');

foreach (string s in Mas1)

Console.WriteLine(s);

Console.ReadLine();

****

**Пример 12**

Console.WriteLine("Разделение строки на буквы");

char[] charMas;

charMas = str.ToCharArray();

foreach (char ch in charMas)

Console.WriteLine(ch);

Console.ReadLine();

**Пример 13**

Console.WriteLine("Удаление символов в начале и конце строки");

str = str.Insert(0, " ");

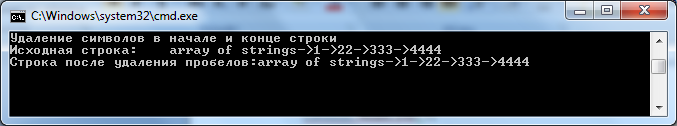
str = str.Insert(str.Length, " ");

Console.WriteLine("Исходная строка:"+str);

str=str.Trim();

Console.WriteLine("Строка после удаления пробелов:" + str);

Console.ReadLine();

****

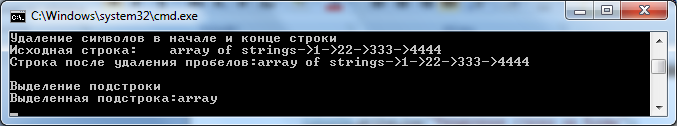
**Пример 14**

Console.WriteLine("Выделение подстроки");

string substr = str.Substring(0, 5);

Console.WriteLine("Выделенная подстрока:" + substr);

Console.ReadLine();

****

### Форматирование строк

При выводе, например, с помощью Console.Write(), значений базовых типов (например, int или double) они автоматиче­ски преобразуются в символьные строки. Если программиста не устраивает автоматически выбранный формат их внешнего представления, он может его изменить. Для этого можно вос­пользоваться статическим методом Format класса string или ис­пользовать так называемую строку форматирования в качестве первого параметра методов, поддерживающих форматирование, например, Console.Write() и Console.WriteLine (). В обоих случаях правила подготовки исходных данных для получения желаемого результата (новой строки) одинаковы.

static string Format (string form, params object[]ar);

* string form – строка форматирования, включает поля подстановок {N[,W]:S[R]]},

где N – номер аргумента,

W – ширина поля,

S – спецификатор формата,

R – спецификатор точности.

* params object[]ar – параметры, подставляемые вместо номера аргумента.

**Пример.**

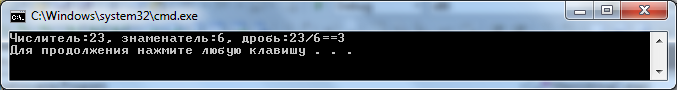
int num=23, den=6;

string result,//ссылка на строку с результатом

form="Числитель:{0}, знаменатель:{1}, дробь:{0}/{1}=={2}";

result=string.Format(form,num,den,num/den);

Console.WriteLine(result);//Числитель 23, знаменатель 6, дробь: 23/6==3



W - ширина поля в поле подстановки определяет количество позиций, выделяемых для изображения подставляемого значения. Если ширина поля не указана, то она определяется автоматически - минималь­но достаточной для изображения значения. Если ширина поля указана и превышает длину помещаемого в поле значения, то при положительной длине поля W значение выравнивается по правой границе. Если перед шириной поля W стоит минус, то выравнивание выполняется по левой границе поля.

Спецификатор формата S задает вид изображае­мого значения.

С,с – валютный, R – количество десятичных разрядов.

D,d – целочисленный, R – минимальное количество цифр.

E,e – экспоненциальный, R – число разрядов после точки.

F,f – с фиксированной точкой, R – число разрядов после точки.

G,g – короткий из E или F.

Х,х – шестнадцатеричный, R – минимальное число цифр.

Пример

static void Main(string[] args)

{

double dou = 1234.567;

string form = "Спецификация E4: {0:E4},\nспецификация F4: {0:F4}";

string result = string. Format( form, dou);

Console. WriteLine(result);

}



Обратите внимание на округление при выводе по формату Е4 и на дополнительный нуль в изображении числа с фикси­рованной точкой при выводе по формату F4. Это определяется спецификатором точности, равным 4.

### Массивы строк

В массив помещают­ся не строки, а только ссылки на них, но при использовании массивов ссылок на строки не требуются никакие специальные операции для организации обращения к собственно строкам через ссылки на них. Поэтому в литературе, посвященной языку С#, зачастую говорят просто о массивах строк.

static void Main(string[] args)

{

string [] stAr = new string[4];

for (int i = 0; i<stAr.Length; i++)

stAr[i] = new string('\*', i + 1);

foreach (string rs in stAr) Console.WriteLine(rs);

}

**Пример**

static void Main(string[] args)

{

Console.WriteLine("Введите строку");

string str = Console.ReadLine();

str=str.Trim();

for (int i = 0; i < str.Length - 1; i++)

if (str[i] == ' ' && str[i + 1] == ' ')

{

str = str.Remove(i, 1);

i--;

}

string[] words = str.Split(' ');

str = "";

foreach (string s in words)

{

// if (s[0] <= '0' || s[0] >= '9') str = str + s + " ";

if(!char.IsDigit(s[0])) str=str+s+" ";

}

if (String.Compare(str, "") == 0) Console.WriteLine("Все слова удалены, результат - пустая строка");

else Console.WriteLine(str);

}

}

### Неизменяемость объектов класса String

К символам объекта класса **string,** можно обращаться только для получения их значений. Например, для получения значения одного символа строки используется выражение с операцией индек­сирования []. Чтобы изменить строку можно воспользоваться следующим алгоритмом:

1. Переписать символы строки в массив с элементами типа char.
2. Выполнить преобразования в массиве с элементами типа char.
3. Создать новую строку, используя конструктор с параметром string(char[]).

**Пример**

string st= "0123456789";//строка

char[] mas;//массив

mas = st.ToCharArray();//преобразование строки в массив

Array.Reverse(mas);//переворот массива

st = new string(mas);//формирование строки из перевернутого массива

Console.WriteLine(st);//вывод перевернутой строки

### Тип System.Text.StringBuilder

Из-за того, что тип string может оказаться неэффективным при **необдуманном** использовании, в библиотеках базовых классов .NET поставляется еще пространство имен System.Text. Внутри этого (достаточно небольшого) пространства имен предлагается класс StringBuilder (Построитель строк).

Класс StringBuilder компенсирует недостаток класса String. Класс принадлежит к изменяемым классам и находится в пространстве имен System.Text. Объекты класса объявляются с явным вызовом конструктора класса.

* Конструктор без параметров создает пустую строку.
* public StringBuilder (string str, int cap). Параметр str задает строку инициализации, cap – емкость объекта;
* public StringBuilder (int curcap, int maxcap). Параметры curcap и maxcap задают начальную и максимальную емкость объекта;
* public StringBuilder (string str, int start, int len, int cap). Параметры str, start, len задают строку инициализации, cap – емкость объекта.

**Над строками класса определены операции:**

* присваивание (=);
* две операции проверки эквивалентности (= =) и (!=);
* взятие индекса ([]).

Операция конкатенации (+) не определена, ее роль играет метод Append.

Со строкой этого класса можно работать как с массивом, допускается не только чтение отдельного символа, но и его изменение.

**Основные методы класса:**

* public StringBuilder Append (<объект>). К строке, вызвавшей метод, присоединяется строка, полученная в качестве параметра. Метод перегружен и может принимать на входе объекты всех простых типов. В качестве результата возвращается ссылка на объект, вызвавший метод.
* public StringBuilder Insert (int location,<объект>). Метод вставляет строку в позицию, указанную параметром location.
* public StringBuilder Remove (int start, int len). Метод удаляет подстроку длины len, начинающуюся с позиции start.
* public StringBuilder Replace (string str1,string str2). Все вхождения подстроки str1 заменяются на строку str2. public StringBuilder AppendFormat (<строка форматов>, <объекты>). Метод является комбинацией метода Format класса String и метода Append. Строка форматов, переданная методу, содержит только спецификации форматов. Полученные в результате форматирования строки присоединяются в конец исходной строки.

За исключением метода Remove, все рассмотренные методы являются перегруженными.

**Пример1 .**

StringBuilder s1 =new StringBuilder("ABC"),

s2 =new StringBuilder("CDE"),

s3 = new StringBuilder();

Console.WriteLine("Иницмализация строк: s1={0}, s2={1}, s3={2}", s1, s2, s3);

s3= s1.Append(s2);

Console.WriteLine("Добавление строк \ns1={0}, s2={1}, s3={2}", s1, s2, s3);

bool ok= (s1==s3);

Console.WriteLine("Сравнение строк \n{0}=={1}, ok={2}", s1, s3, ok);

char ch1 = s1[0], ch2=s2[0];

Console.WriteLine("Прямой доступ к символам: ch1={0}, ch2={1}", ch1, ch2);

StringBuilder s = new StringBuilder("Zenon");

s[0]='L';

Console.WriteLine(s);

**Пример 2.**

string buf;

Console.WriteLine(">");

buf = Console.ReadLine();

StringBuilder bs = new StringBuilder(buf);

//StringBuilder bs = new StringBuilder("123 abc 12 d 123");

bs.Append(" ");

int l = 0; int r = 0;

while (l < bs.Length - 1)

{

r = l;

while (bs[r] != ' ') r++; //пропускаем символы

if (char.IsDigit(bs[l]))

{

bs.Remove(l, r - l);

}

else

{

l = r + 1;

while (l == ' ' && l < bs.Length - 1) l++;//пропускаем пробелы

}

}

int i = 0, count = 0;

while(i<bs.Length)

{

if(bs[i]==' ')count++;

i++;

}

if(count==bs.Length) Console.WriteLine("Cтрока пустая");

else Console.WriteLine(bs);