Строки в C#

Строки

Строка является объектом типа String, значением которого является текст. Текст хранится в виде последовательной доступной только для чтения набора объектов Char. В конце строки на языке C# отсутствует символ, заканчивающийся на NULL; поэтому строка C# может содержать любое число внедренных символов NULL ("\0"). Свойство Length строки представляет число объектов Char, содержащихся в этой строке, а не число символов Юникода.

В C# ключевое слово string является псевдонимом свойства String. Поэтому String и string эквивалентны

Строки

Объявление и инициализацию строк можно выполнять различными способами:

```
string message1;
string str = "Пример строки";
char[] letters = { 'A', 'B', 'C' };
string alphabet = new string(letters);
```

Строки

Над строками определены следующие операции:

- присваивание (=);
- конкатенация (объединение) или сцепление строк (+);
- две операции проверки эквивалентности: равно (= =) и не равно (!=);
- взятие индекса ([]).

Строки

Переприсваивание

Строки можно целиком переприсваивать:

```
string s1 = "Hello";
string s2 = s1;
```

Строки

Объединение строк

Можно объединять строки с помощью оператора +:

```
string s1 = "orange";
string s2 = "red";
s1 += s2;Console.WriteLine(s1);  // напечатается "orangered"
```

Строковые объекты являются неизменяемыми: после создания их нельзя изменить. Все методы String и операторы C#, которые, как можно было бы представить, изменяют строку, в действительности возвращают результаты в новый строковый объект.

.

Строки

Постоянство строк

Строковые объекты являются неизменяемыми: после создания их нельзя изменить. Все методы String и операторы C#, которые, как можно было бы представить, изменяют строку, в действительности возвращают результаты в новый строковый объект. В примере, когда содержимое строк s1 и s2 объединяется в одну строку, две исходные строки не изменяются. Оператор += создает новую строку с объединенным содержимым. Этот новый объект присваивается переменной s1, а исходный объект, который был присвоен строке s1, освобождается для сборки мусора, поскольку ни одна переменная не содержит ссылку на него.

Строки

Сравнения

Самый простой способ сравнения двух строк — использовать операторы == и !=, осуществляющие сравнение с учетом регистра:

```
string color1 = "red";
string color2 = "green";
string color3 = "red";
if (color1 == color3)   Console.WriteLine("Cтроки равны");
if (color1 != color2)   Console.WriteLine("Cтроки не равны");
```

Не допускается использование >,<,>=,<= для сравнения строк. Для строковых объектов существует метод CompareTo(), возвращающий целочисленное значение, зависящее от того, что одна строка может быть меньше (<), равна (==) или больше другой (>). При сравнении строк используется значение Юникода, при этом значение строчных букв меньше, чем значение заглавных.

Строки

Доступ к отдельным знакам

Квадратные скобки [] служат для доступа к отдельным знакам в объекте string, но при этом возможен доступ только для чтения:

```
string str = "test";
char x = str[2];  // x = 's';
```

Строки

В C# существуют два вида строковых констант:

- обычные константы, которые представляют строку символов, заключённую в кавычки;
- @-константы, заданные обычной константой с предшествующим знаком @.

Строки

Обычные константы

В обычных константах некоторые символы интерпретируются особым образом. Связано это, прежде всего, с тем, что необходимо уметь задавать в строке непечатаемые символы, такие, как, например, символ табуляции. Возникает необходимость задавать символы их кодом – в виде escape-последовательностей. Для всех этих целей используется комбинация символов, начинающаяся символом "\" - обратная косая черта. Это так называемые Escape-последовательности

Строки

Escape-последовательность	Имя символа
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратная косая черта
\0	Нуль-символ
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция

При этом возникают неудобства: например, при задании констант, определяющих путь к файлу, приходится каждый раз удваивать символ обратной косой черты.

string path2 = "C:\\Users\\Mikant\\Documents";

Строки

@-константы

В @-константах все символы трактуются в полном соответствии с их изображением. Поэтому путь к файлу лучше задавать @-константой. Единственная проблема в таких случаях: как задать символ кавычки, чтобы он не воспринимался как конец самой константы. Решением является удвоение символа.

string path1 = @"C:\Users\Mikant\Documents";

Строки

Метод	Назначение
Compare()	Статический метод, который позволяет сравнить две строки
Concat()	Комбинирует отдельные экземпляры строк в одну строку (конкатенация)
Contains()	Метод, который позволяет определить, содержится ли в строке определенная подстрока
CopyTo()	Копирует определенное число символов, начиная с определенной позиции в новый экземпляр массива
IndexOf()	Находит первое вхождение заданной подстроки или символа в строке
Insert()	Метод, который позволяет вставить строку внутрь другой определенной строки
LastIndexOf()	То же, что IndexOf, но находит последнее вхождение
Split()	Метод, возвращающий массив string с присутствующими в данном экземпляре подстроками внутри, которые отделяются друг от друга элементами из указанного массива char или string

Строки

Метод	Назначение
Remove() Replace()	Методы, которые позволяют получить копию строки с соответствующими изменениями (удалением или заменой символов)
Substring()	Извлекает подстроку, начиная с определенной позиции строки
ToUpper () ToLower()	Методы, которые позволяют создавать копию текущей строки в формате, соответственно, верхнего или нижнего регистра
Trim()	Метод, который позволяет удалять все вхождения определенного набора символов с начала и конца текущей строки
Length()	Определяет длину строки

Строки

```
string s6 = «РГппУ";
Console.WriteLine(s6.ToUpper()); // Напечатается РГППУ

string s3 = "Visual C# Express";
string s5 = s3.Replace("C#", "Basic");
Console.WriteLine(s5);// напечатается "Visual Basic Express”

string s3 = "Visual C# Express";
string s4 = s3.Substring(7, 2);
Console.WriteLine(s4);// напечатается "C#"
```

Строки

```
char razdelitel = ' ';
string text = "Шла Саша по шоссе и сосала сушку";
Console.WriteLine("Исходный текст: '{0}'", text);
string[] words = text.Split(razdelitel);
Console.WriteLine("{0} слов в тексте:", words.Length);
```

В качестве разделителя может выступать массив символов.

```
char[] delimiterChars = { ' ', ',', '.', ':', '\t' };
string text = "one\ttwo three:four,five six seven";
Console.WriteLine("Original text: '{0}'", text);
string[] words = text.Split(delimiterChars);
Console.WriteLine("{0} words in text:", words.Length);
```

Строки

Метод Join

Конкатенация массива строк в единую строку. При конкатенации между элементами массива вставляются разделители. Операция, заданная методом Join, является обратной к операции, заданной методом Split. Последний является динамическим методом и, используя разделители, осуществляет разделение строки на элементы

```
Words = txt.Split(',', ' ');
 for(int i=0;i< Words.Length; i++)
Console.WriteLine("Words[{0}]= {1}",i, Words[i]);
txtjoin = string.Join(" ",Words);
```

Метод Format

Метод Format, как и большинство методов, является перегруженным и может вызываться с разным числом параметров.

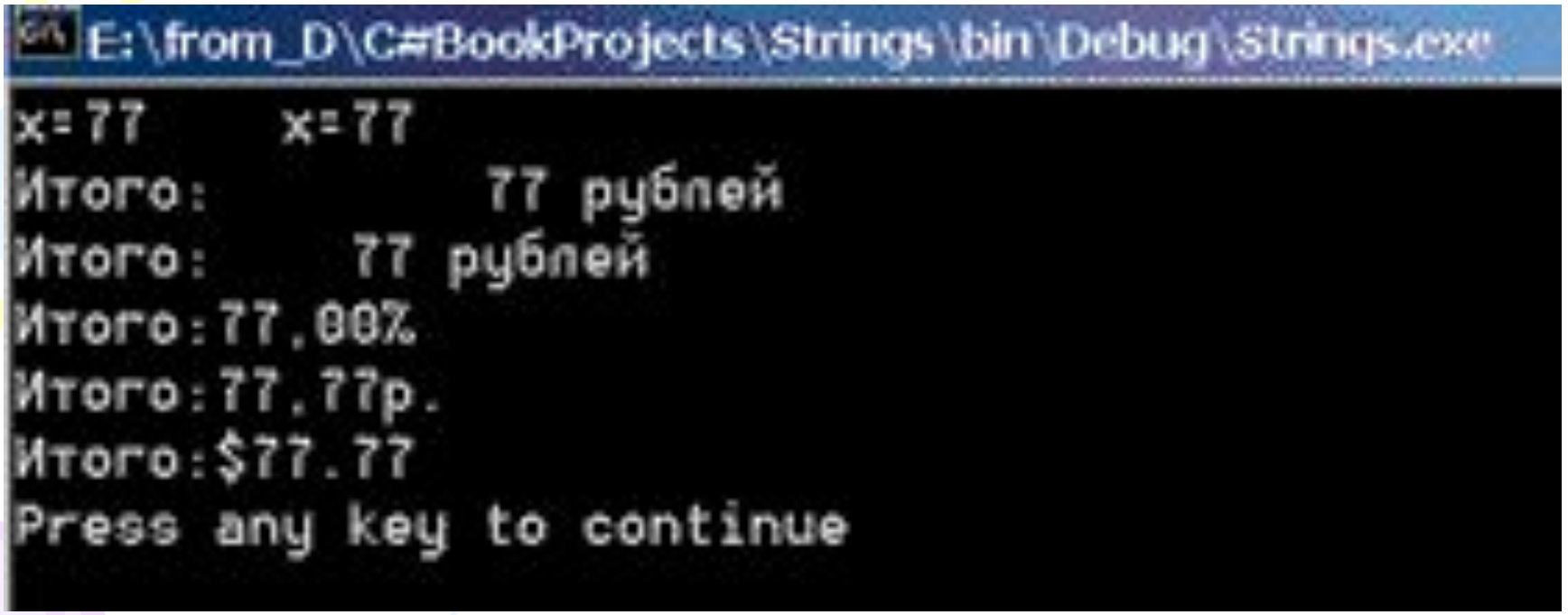
Общий синтаксис, специфицирующий формат, таков:

{N [,M [:<коды_форматирования>]]}

Обязательный параметр N задает индекс объекта, заменяющего формат. Второй параметр M, если он задан, определяет минимальную ширину поля, которое отводится строке, вставляемой вместо формата. Третий необязательный параметр задает коды форматирования, указывающие, как следует форматировать объект.

Метод Format

```
int x=77;
string s= string.Format("x={0}",x);
Console.WriteLine(s + "\tx={0}",x);
s= string.Format("Итого:{0,10} рублей",x);
 Console.WriteLine(s);
s= string.Format("Итого:{0,6:######} рублей",x);
Console.WriteLine(s);
 s= string.Format("Итого:{0:P} ",0.77);
Console.WriteLine(s);
 s= string.Format("Итого:{0,4:C} ",77.77);
 Console.WriteLine(s);
```


E:\from_D\C#BookProjects\Strings\bin\Debug\Strings.exe
x=77 x=77
Итого: 77 рублей
Итого: 77 рублей
Итого:77,00%
Итого:77,77р.
Итого:$77.77
Press any key to continue

Спецификаторы формата для строк

C или c	Вывод значений в денежном (currency) формате.
D или d	Вывод целых значений.
E или e	Вывод значений в экспоненциальном формате, то есть в виде d.ddd…E+ddd или d.ddd…e+ddd.
F или f	Вывод значений с фиксированной точностью.
G или g	Формат общего вида. Вывод значений с фиксированной точностью или в экспоненциальном формате, в зависимости от того, какой формат требует меньшего количества позиций.
N или n	Вывод значений в формате d,ddd,ddd.ddd. После спецификации можно задать целое число, определяющее длину дробной части
P или p	Вывод числа в процентном формате
R или r	Отмена округления числа при преобразовании в строку. Гарантирует, что при обратном преобразовании в значение того же типа получится то же самое
X или x	Вывод значений в шестнадцатеричном формате.

Строки

Преобразование строк в другие типы

С помощью объекта Convert:

```
N = Convert.ToInt32(s1);
M = Convert.ToDouble(s2);
F = Convert.ToBoolean(s3);
B = Convert.ToByte(s4);
C = Convert.ToChar(k);
s5= Convert.ToString(x);
```

Класс StringBuilder

Класс string не разрешает изменять существующие объекты. Строковый класс StringBuilder позволяет компенсировать этот недостаток. Этот класс принадлежит к изменяемым классам и его можно найти в пространстве имен System.Text.

Объекты этого класса объявляются с явным вызовом конструктора класса. Поскольку специальных констант этого типа не существует, то вызов конструктора для инициализации объекта просто необходим.

public StringBuilder (string str, int cap). Параметр str задает строку инициализации, cap - емкость объекта объем памяти, отводимой данному экземпляру класса StringBuilder. Каждая из этих групп не является обязательной и может быть опущена.

StringBuilder s1 =new StringBuilder("ABC")

Класс StringBuilder

Операции над строками

Над строками этого класса определены практически те же операции с той же семантикой, что и над строками класса String:

присваивание (=);

две операции проверки эквивалентности (==) и (!=);

взятие индекса ([]).

Класс StringBuilder

Операция конкатенации (+) не определена над строками класса StringBuilder, ее роль играет метод Append, дописывающий новую строку в хвост уже существующей.

Со строкой этого класса можно работать как с массивом, но, в отличие от класса String, здесь уже все делается как надо: допускается не только чтение отдельного символа, но и его изменение.

Класс StringBuilder

```
StringBuilder s1 =new
StringBuilder("ABC"),
s2 =new StringBuilder("CDE");
StringBuilder s3 = new StringBuilder();
s3= s1.Append(s2);
StringBuilder s = new
StringBuilder("Zenon"); s[0]='L';
```

Для того чтобы имя класса StringBuilder стало доступным, в проект добавлено предложение using System.Text, ссылающееся на соответствующее пространство имен.

Основные методы

У класса StringBuilder методов значительно меньше, чем у класса String. Это и понятно - класс создавался с целью дать возможность изменять значение строки. По этой причине у класса есть основные методы, позволяющие выполнять такие операции над строкой как вставка, удаление и замена подстрок, но нет методов, подобных поиску вхождения, которые можно выполнять над обычными строками. Технология работы обычно такова: конструируется строка класса StringBuilder ; выполняются операции, требующие изменение значения; полученная строка преобразуется в строку класса String ; над этой строкой выполняются операции, не требующие изменения значения строки. Давайте чуть более подробно рассмотрим основные методы класса StringBuilder:

public StringBuilder Append (<объект>). К строке, вызвавшей метод, присоединяется строка, полученная из объекта, который передан методу в качестве параметра

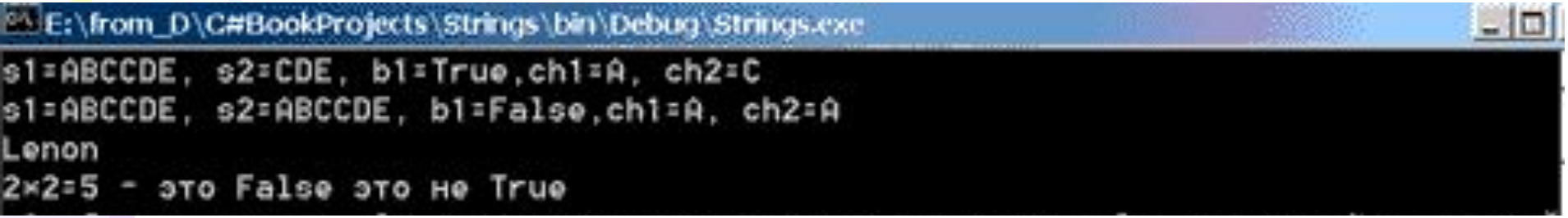
public StringBuilder Insert (int location,<объект>). Метод вставляет строку, полученную из объекта, в позицию, указанную параметром location. Метод Append является частным случаем метода Insert ;

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. В соответствии с этими спецификациями находятся и форматируются объекты. Полученные в результате форматирования строки присоединяются в конец исходной строки.

```
StringBuilder strbuild = new
StringBuilder();
string str = "это     это      не ";
strbuild.Append(str);
strbuild.Append(true);
strbuild.Insert(4,false);
strbuild.Insert(0,"2*2=5 - ");
Console.WriteLine(strbuild);
```



Текстовые файлы в C#

Классы для работы с файлами

Для работы с классами необходимо подключить пространство имен System.IO

using System.IO

Классы:

- StreamWriter - Реализует TextWriter для записи символов в поток в определенной кодировке.
- StreamReader - Реализует TextReader, который считывает символы из потока байтов в определенной кодировке.
- File - Предоставляет статические методы для создания, копирования, удаления, перемещения и открытия файлов

Методы класса File

CreateText - Создается или открывается файл для записи текста в кодировке UTF-8.

AppendText - Создает StreamWriter добавляющий в существующий файл текст в кодировке UTF-8.

OpenText - Открывает для чтения существующий файл, содержащий текст в кодировке UTF-8.

Exists - Определяет, существует ли заданный файл.

ReadAllText(String) - Открывает текстовый файл, считывает все строки файла и затем закрывает файл

ReadAllLines(String) - Открывает текстовый файл, считывает все строки файла и затем закрывает файл.

WriteAllText(String, String) - Создает новый файл, записывает в него указанную строку и затем закрывает файл. Если целевой файл уже существует, он будет переопределен.

WriteAllLines(String, String[]) - Создает новый файл, записывает в него указанный массив строк и затем закрывает файл.

Методы класса StreamWriter

StreamWriter(String) - Инициализирует новый экземпляр класса StreamWriter для указанного файла с помощью кодировки по умолчанию и размера буфера.

Write(String) - Записывает в поток строку.

WriteLine(String) - Записывает в текстовую строку или поток строку, за которой следует признак конца строки.

Close - Закрывает текущий объект StreamWriter и базовый поток.

Методы и свойства класса StreamReader

StreamReader(String) - Инициализирует новый экземпляр класса StreamReader для указанного имени файла.

ReadToEnd - Считывает все символы, начиная с текущей позиции до конца потока.

ReadLine - Выполняет чтение строки символов из текущего потока и возвращает данные в виде строки.

EndOfStream - Получает значение, определяющее, находится ли позиция текущего потока в конце потока.

Примеры

Создание или открытие файла и построчное добавление записей

```
if(File.Exists(@"d:\vbbook.txt"))

    writefl = File.AppendText(@"d:\vbbook.txt");
    else
    writefl = File.CreateText(@"d:\vbbook.txt");
```

writefl описан начале как StreamWriter writefl;

Примеры

Построчная запись в файл

```
writefl.WriteLine(textBox1.Text);
```

В конце записи следует закрыть файл:

```
writefl.Close();
```

Примеры

Открытие файла для чтения

```
if (File.Exists(@"d:\vbbook.txt"))
            readfl = new
  StreamReader(@"d:\vbbook.txt");
     else MessageBox.Show("файла нет");
```

В начале readfl описан как

```
 StreamReader readfl;
```

Примеры

Построчное чтение из файла

```
if (readfl.EndOfStream == false)
                label1.Text =
  readfl.ReadLine();
            else label1.Text = "end";
```

В конце чтения надо закрыть файл:

```
readfl.Close();
```

Примеры

Чтение файла в массив строк

```
strmas =
  File.ReadAllLines(@"d:\vbbook.txt");
```

В начале массив описан как:

```
string[] strmas ;
```

Примеры

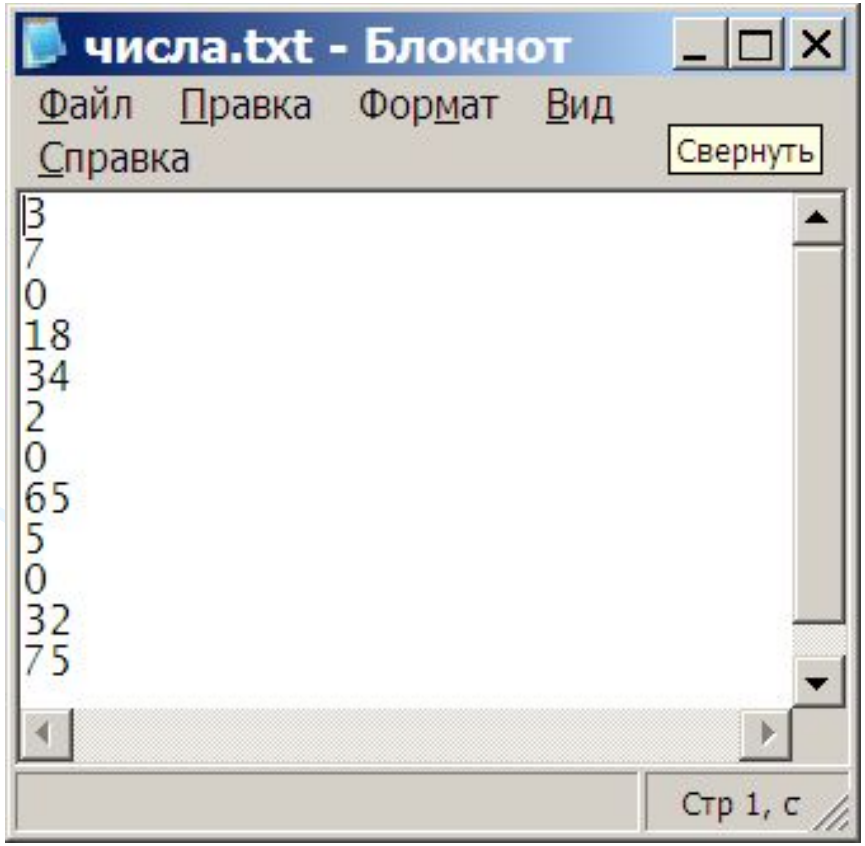
Запись текста из текстового поля в файл

```
File.WriteAllText(@"d:\vbbook.txt",textBox1.Text);
```

Задание 1

Имеется текстовый файл “числа.txt”, содержащий целые числа (не более 20).

Переписать числа из этого файла в файл “без нулей.txt”удалив нули.



Код программы

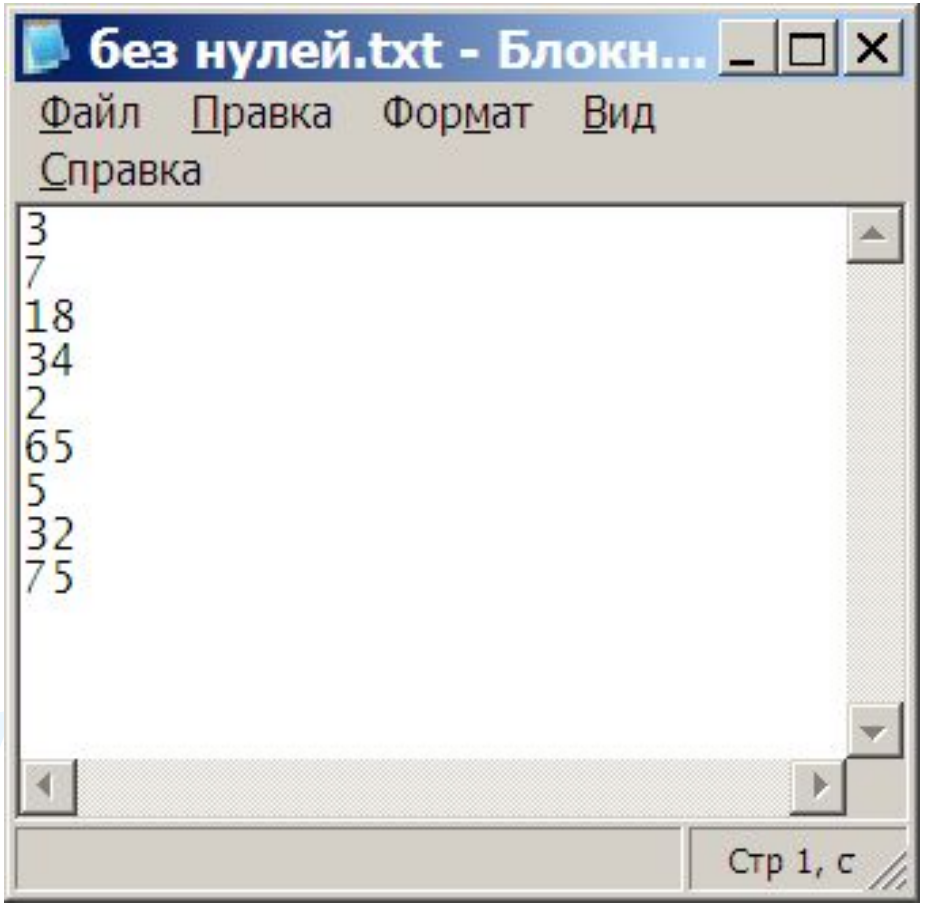
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace ConsoleApplication1
{    class Program
    {
        static void Main(string[] args)
        {   StreamReader readfl;// поток для считывания данных
            StreamWriter writefl; //поток для записи данных
            int[] massiv=new int[20];//массив чисел
            int k = 0; //начальный индекс массива
//проверка, существует ли данный файл
            if (File.Exists(@"d:\числа.txt"))
                    readfl = new StreamReader(@"d:\числа.txt");
                else {Console.WriteLine("файла нет");
                    Console.ReadKey();
                        return;}
```

Код программы

```
//чтение чисел из файла в массив
while (readfl.EndOfStream == false)
            {
                massiv[k] = Convert.ToInt32(readfl.ReadLine());
                k++;
            }
//вывод массива на экран
            for(int i=0;i<k;i++)
                Console.WriteLine(massiv[i]);
                 Console.ReadKey();
                 readfl.Close(); //закрытие файла
//создание нового файла
                 writefl=File.CreateText(@"d:\без нулей.txt");
//запись чисел в новый файл
                 for (int i = 0; i < k; i++)
                     if (massiv[i] != 0) writefl.WriteLine(massiv[i]);
                 writefl.Close(); //закрытие файла
            }
    }
}
```

Примеры

Новый файл “без нулей.txt”будет иметь вид:



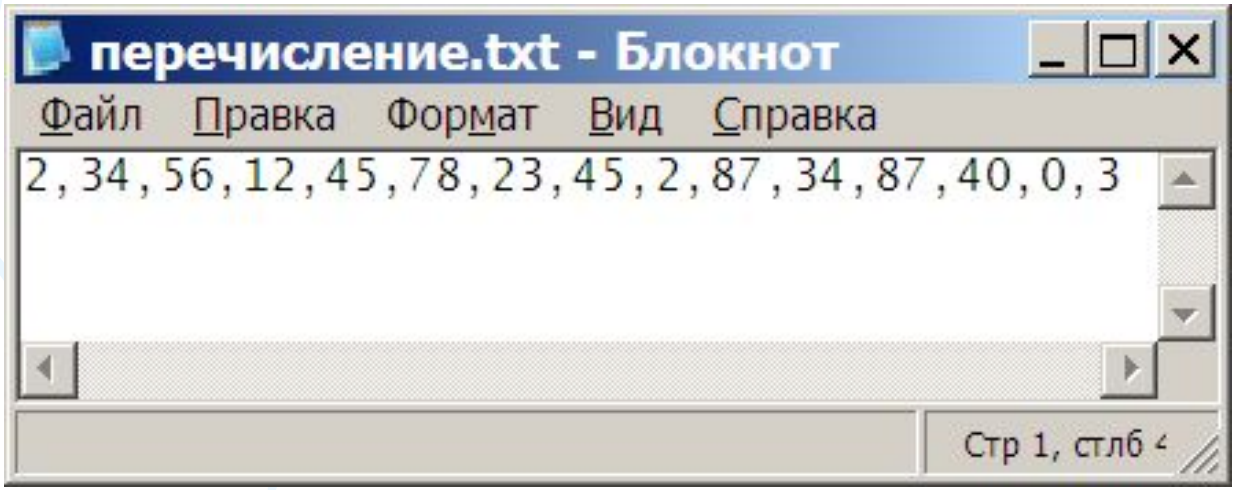
Примеры

В приведенном выше примере заранее оговаривается максимальное количество чисел в файле(не более 20), т.к. требуется зарезервировать место для элементов массива.

Кроме этого, в файле каждое число располагалось в отдельной строке.

Задание 2

Дан файл “перечисление.txt”, содержащий числа, записанные в строке и разделенные друг от друга запятой. Требуется считать числа из файла и вывести их на экран.



Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace ConsoleApplication1
{    class Program
    {
        static void Main(string[] args)
        {   StreamReader readfl;// поток для считывания данных
            string stroka;//строка, куда будут считываться данные
//проверка, существует ли данный файл
            if (File.Exists(@"d:\перечисление.txt"))
                readfl = new StreamReader(@"d:\перечисление.txt");
               else
              {  Console.WriteLine("файла нет");
                 Console.ReadKey();
                  return;       }
```

Код программы

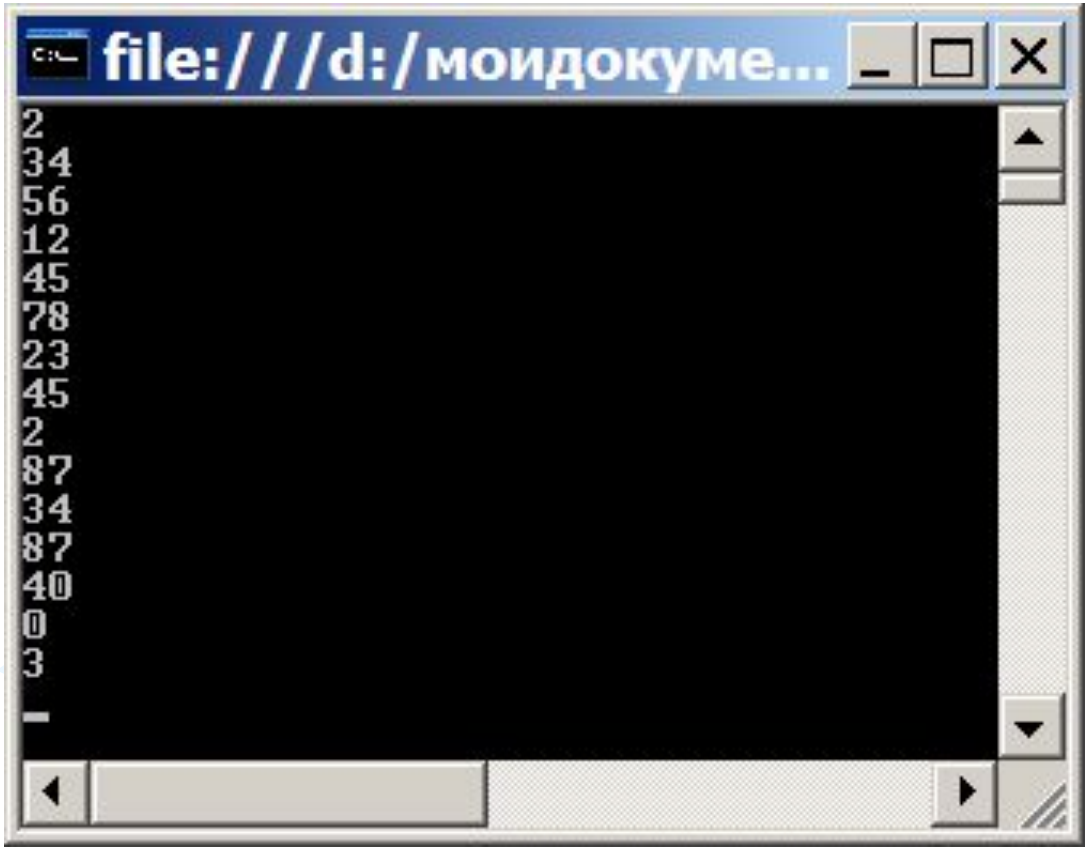
```
stroka = readfl.ReadLine();//считывание данных в строку

//получение массива строк, разделенных запятой
            string[] chisla=stroka.Split(',');

//объявление массива чисел такого же размера, как массив строк
            int[] massiv=new int[chisla.Length];
// записи элементов из массива строк в числовой массив
            for (int i = 0; i < chisla.Length; i++)
                massiv[i] = Convert.ToInt32(chisla[i]);
//вывод числового массива на экран
            for (int i = 0; i < chisla.Length; i++)
                Console.WriteLine(massiv[i]);
            Console.ReadKey();
            readfl.Close();
            }
    }
}
```

Примеры

Результат вывода массива на экран



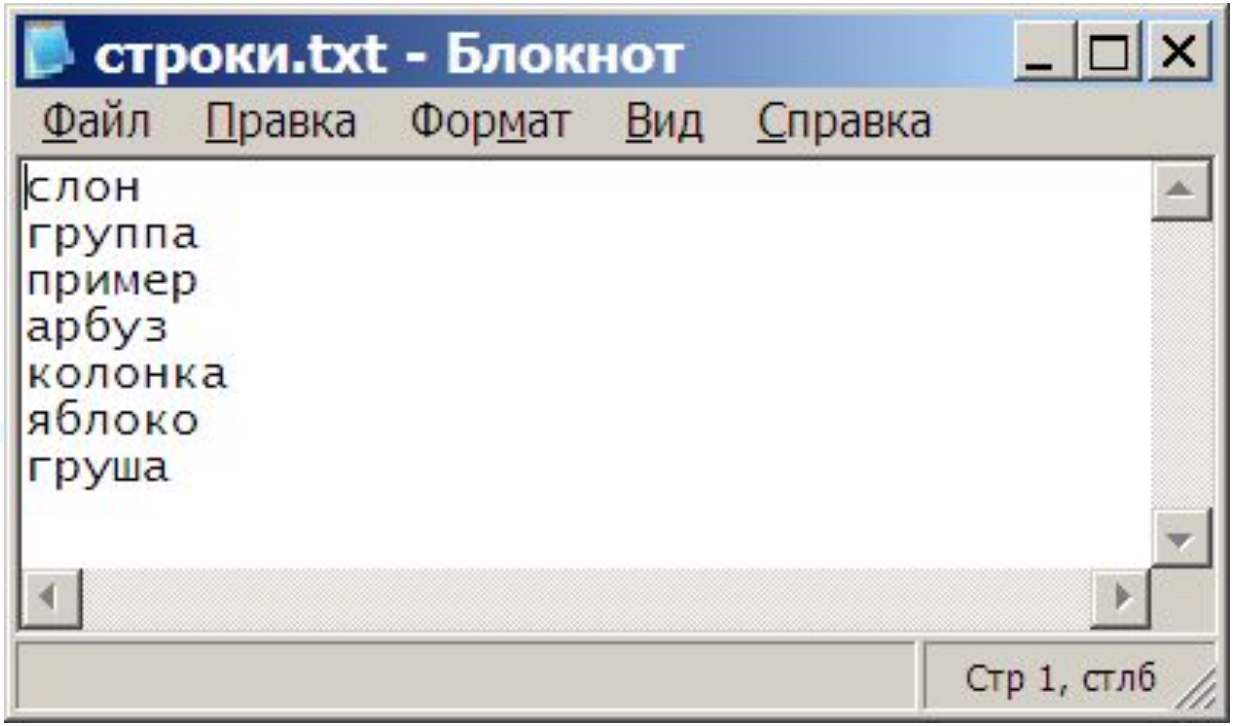
Примеры

Ранее рассматривались примеры, в которых использовались классы StreamWriter и StreamReader.

Иногда проще работать со всем файлом, как целым, используя класс File.

Задание 4

Дан файл”строки.txt”, содержащий набор строк. Получить новый файл “сортировка.txt”, в котором эти строки будут отсортированы по алфавиту.



Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace ConsoleApplication2
{    class Program
   { static void Main(string[] args)
     {string[] strmas ;//объявление массива строк
//проверка, еслит файл существует, то …
      if (File.Exists(@"d:\строки.txt"))
       {  //считывание данных в массив сторк
          strmas = File.ReadAllLines(@"d:\строки.txt");
           //сортировка массива
          Array.Sort(strmas);
          //запись массива в файл
          File.WriteAllLines(@"d:\сортировка.txt", strmas);
       }
     }
   }
}
```

Примеры

Результат выполнения программы