СИМВОЛЫ И СТРОКИ

Символы char

Тип char предназначен для хранения символа в кодировке Unicode.

Замечание

Кодировка Unicode является двухбайтной, т.е., каждый символ представлен двумя байтами, а не одним, как это сделано кодировке ASCII, используемой в ОС Windows. Из-за этого могут возникать некоторые проблемы, если вы решите, например, работать посимвольно с файлами, созданными в стандартном текстовом редакторе Блокнот. Способ решения этой проблемы будет рассмотрен в разделе, посвященном работе с файлами.

Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .Net из пространства имен System. В этом классе определены статические методы, позволяющие задавать вид и категорию символа, а также преобразовывать символ в верхний, или нижний регистр, а также 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 (‘\n’, ‘\t’, или ‘\r’).
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 при этом должна состоять из одного символа.

Метод	Описание
ToLower	Преобразует символ в нижний регистр
ToUpper	Преобразует символ в верхний регистр

В следующем примере рассмотрим применение данных методов:

```
class Program
{
   static void Main()
   {
      Console.WriteLine("{0,5} {1,8} {2,15}",
                        "код", "символ", "назначение");
      // реальной верхней границей для i является значение 65535
      for (ushort i = 0; i < 255; i++)
      {
         char a = (char)i;
         Console.Write("\n{0,5} {1,8}", i, a);
         if (char.IsLetter(a))
            Console.Write("{0,20}", "Буква");
         if (char.IsUpper(a))
            Console.Write("{0,20}", "Верхний регистр");
         if (char.IsLower(a))
            Console.Write("{0,20}", "Нижний регистр");
         if (char.IsControl(a))
            Console.Write("{0,20}", "Управляющий символ");
         if (char.IsNumber(a))
            Console.Write("{0,20}", "Число");
         if (char.IsPunctuation(a))
            Console.Write("{0,20}", "Знак препинания");
         if (char. IsDigit (a))
            Console.Write("{0,20}", "Цифра");
         if (char. IsSeparator (a))
            Console.Write("{0,20}", "Разделитель");
         if (char. IsWhiteSpace (a))
            Console.Write("{0,20}", "Пробельный символ");
      }
   }
}
```

Используя символьный тип, можно организовать массив символов и работать с ним, используя класс Array:

```
class Program
{
   static void Print(char[] a)
   {
      foreach (char elem in a)
      {
         Console.Write(elem);
      }
      Console.WriteLine();
   }
```

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

Результат работы программы:

```
Исходный массив a:
maXiMuS!!!
Измененный массив a:
MAXIMUS!!!
Исходный массив b:
кол около колокола
Измененный массив b:
алоколок олоко лок
```

Задание

Измените программу так, чтобы:

1) *в массиве a подсчитывалось количество знаков пунктуации;*
2) *в массиве b символы размещались в алфавитном порядке.*

Строковый тип string

Тип string, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый класс System.String библиотеки .Net. Тип string относится к ссылочным типам, хотя работа с ним во многом напоминает работу с размерными типами.

Существенной особенностью данного класса является то, что каждый его объект – это неизменяемая (immutable) последовательность символов Unicode. Любое действие со строкой ведет к тому, что создается копия строки, в которой и выполняются все изменения. Исходная же строка не меняется. Такой подход к работе со строками может показаться странным, но он обусловлен необходимостью сделать работу со строками максимально быстрой и безопасной. Например, при наличии нескольких одинаковых строк CLR может хранить их по одному и

тому же адресу (данный механизм называется string interning), экономя таким образом память.

Создать объект типа string можно несколькими способами:

1) string s; *// инициализация отложена*

2) string s="кол около колокола"; *//инициализация строковым литералом*

3) string s=@"Привет! *//символ @ сообщает конструктору string, что строку*
Сегодня хорошая погода!!! " *// нужно воспринимать буквально, даже если она занимает*
//несколько строк

4) int x = 12344556; *//инициализировали целочисленную переменную*
string s = x.ToString(); *//преобразовали ее к типу string*

5) string s=new string (' ', 20); *//конструктор создает строку из 20 пробелов*

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

Замечание

В примерах 1-4 используется неявный вызов конструктора. В примерах 5-7 конструктор вызывается явным образом через использование операции new.

С объектом типа string можно работать посимвольно, т.е., поэлементно:

```
class Program
{
   static void Main()
   {
      string a = "кол около колокола";
      Console.WriteLine("Дана строка: {0}", a);
      char b = 'о';
      int k = 0;
      for (int x = 0; x < a.Length; x++)
      {
         if (a[x] == b)
         {
            k++;
         }
      }
      Console.WriteLine("Символ {0} содержится в ней {1} раз",b,k );
   }
}
```

Результат работы программы:

```
Дана строка: кол около колокола
Символ о содержится в ней 7 раз
```

Однако при попытке заменить в данной строке все вхождения буквы о на букву а, ожидаемого результата мы не получим:

```
for (int x = 0; x <  a.Length; x++)
{
   if (a[x] == b)
   {
      a[x] = c;   //1
   }
}
```

Относительно строки 1 компилятор выдаст сообщение об ошибке:

Property or indexer string.this[int] cannot be assigned to – it is read only

Тем самым компилятор запрещает напрямую изменять значение строки.

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 учетом, или без учета регистра.
CompareTo	Экземплярный 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, LastIndexOf	Экземплярные методы	Определение индекса первого или, соответственно, последнего вхождения подстроки в данной строке.
IndexOfAny, LastIndexOfAny	Экземплярные методы	Определение индекса первого или, соответственно, последнего вхождения любого символа из подстроки в данной строке.
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, соответственно, правому краю путем вставки нужного числа пробелов в начале, или в конце строки.

Название	Вид	Описание
Remove	Экземплярный метод	Удаление подстроки из заданной позиции.
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки, или символа новыми подстрокой, или символом.
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	Экземплярные методы	Преобразование строки к нижнему или, соответственно, к верхнему регистру.
Trim, TrimStart, TrimEnd	Экземплярные методы	Удаление пробелов в начале и конце строки, или только с начала, или только с конца соответственно.

Замечание

Напоминаем, что вызов статических методов происходит через обращение к имени класса, например, String.Concat(str1, str2), а обращение к экземплярым методам через объекты (экземпляры класса), например, str.ToLower().

Обратите внимание на то, что все методы возвращают ссылку на новую строку, созданную в результате преобразования копии исходной строки. Для того, чтобы сохранить данное преобразование, нужно установить на него новую ссылку. Рассмотрим пример.

```
string a ="кол около колокола";
Console.WriteLine("Строка a: {0}", a);
a.Remove(0,4);
Console.WriteLine("Строка a: {0}", a);
```

Результат работы программы:

```
Строка a: кол около колокола
Строка a: кол около колокола
```

В данном примере компилятор никаких сообщений не выдаст, но мы и не увидим никаких преобразований со строкой.

Чтобы изменения вступили в силу, необходимо явно присвоить результат выполнения метода Remove какой-нибудь строке.

```
string a ="кол около колокола";
Console.WriteLine("Строка a: {0}", a);
string b=a.Remove(0,4);
Console.WriteLine("Строка a: {0}", a);
Console.WriteLine("Строка b: {0}", b);
```

Результат работы программы:

```
Строка a: кол около колокола
Строка a: кол около колокола
Строка b: около колокола
```

Результат выполнения метода Remove, можно записать и в саму переменную a.

```
string a ="кол около колокола";
Console.WriteLine("Строка a: {0}", a);
a=a.Remove(0,4);
Console.WriteLine("Строка a: {0}", a);
```

Результат работы программы:

```
Строка a: кол около колокола
Строка a: около колокола
```

В этом случае будет потеряна ссылка на исходное строковое значение "кол около колокола", хотя оно и будет занимать память. Освободить занятую память сможет только сборщик мусора.

Рассмотрим следующий фрагмент программы:

```
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:

```
!
!!
!!!
…
!!!...!!
```

И только на последнюю из них будет ссылаться переменная a. Ссылки на все остальные строчки будут потеряны, но, как и в предыдущем примере, эти строки будут храниться в памяти компьютера и засорять ее. Боротся с таким засорением придется сборщику мусора, что будет сказываться на производительности программы.

Рассмотренные примеры определяют область применения типа string – это поиск, сравнение, извлечение информации из строки. Если же нужно изменять строку, то лучше пользоваться классом StringBuilder, который мы рассмотрим позже.

Вернемся к методам класса string и рассмотрим их на следующем примере:

```
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: {0}", str1);
        Console.WriteLine("Длина строки str1: {0}",str1.Length);
        Console.WriteLine("str2: {0}", str2);
        Console.WriteLine("Длина строки str2: {0}",str2.Length);
        Console.WriteLine("str3: {0}", str3);
        Console.WriteLine("Длина строки str3: {0}",str3.Length);
        Console.WriteLine("str4: {0}", str4);
        Console.WriteLine("Длина строки str4: {0}",str4.Length);
        Console.WriteLine();

        // создаем прописную и строчную версии строки str1
        strLow = str1.ToLower();
        strUp = str1.ToUpper();
        Console.WriteLine("Строчная версия строки str1: {0}",strLow);
        Console.WriteLine("Прописная версия строки str1: {0}",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\" в str2: {0}",idx);
            idx = str2.LastIndexOf("о");
            Console.WriteLine("Индекс последнего вхождения символа
                                    \'о\' в str2: {0}", idx);
            Console.WriteLine();

            //конкатенация
            string str = String.Concat(str1, str2, str3, str4);
            Console.WriteLine("Слияние строк: {0}",str);
            Console.WriteLine();

            //замена подстроки "строка" на пустую подстроку
            string newstr = str.Replace("строка","");
            Console.WriteLine("Замена: {0}",newstr);
        }
    }
```

Результат работы программы:

```
str1: Первая строка
Длина строки str1: 13
str2: Первая строка
Длина строки str2: 13
str3: Вторая строка
Длина строки str3: 13
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» в str2: 7
Индекс последнего вхождения символа 'o' в str2: 10
Слияние строк: Первая строкаПервая строкаВторая строкаВТОРАЯ строка
Замена: Первая Первая Вторая ВТОРАЯ
```

Очень важными методами обработки строк являются методы разделения строки на элементы: Split, и слияние массива строк в единую строку: Join.

```
class Program
{
   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]);
      }
      // собираем эти части в одну строку, в качестве
      // разделителя используем символ |
      string whole = String.Join(" | ", parts);
      Console.WriteLine("Результат сборки: ");
      Console.WriteLine(whole);
   }
}
```

Результат работы программы:

```
Результат разбиения строки на части:
тучки
небесные
вечные
странники
Результат сборки:
тучки | небесные | вечные | странники
```

Задание

Измените программу так, чтобы слова в предложении, полученном в результате сборки, были записаны в обратном порядке.

В общем случае строка может содержать и другие разделители:

```
class Program
{
   static void Main()
   {
      string poems = " тучки небесные, вечные странники...";
      char[] div = { ' ', ',', '.'}; //создаем массив разделителей
      // Разбиваем строку на части,
      string[] parts = poems.Split(div);                    // 1
      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 whole = String.Join(" | ", parts);
      Console.WriteLine("Результат сборки: ");
```

```
            Console.WriteLine(whole);
        }
    }
```

Результат работы программы:

```
Результат разбиения строки на части:
тучки
небесные
вечные
странники
Результат сборки:
тучки | небесные |   | вечные | странники |   |   |
```

В результате работы программы мы получили пустые слова. Это связано с особенностью работы метода Split, в котором полагается, что между двумя разделителями должно быть слово. Поэтому, если два разделителя стоят рядом, то между ними стоит пустое слово.

Если необходимо, чтобы при разделении строки на слова пустые строки игнорировались, необходимо использовать перегруженную версию метода Split. Например, если заменить в предыдущем примере строку 1 на следующую команду

```
string[] parts =
   poems.Split(div, StringSplitOptions.RemoveEmptyEntries);
```

в которой параметр StringSplitOptions.RemoveEmptyEntries позволяет игнорировать незначащие разделители, то результат работы программы будет выглядеть следующим образом:

```
тучки | небесные | вечные | странники
```

Используя метод Split можно вводить двумерный массив не поэлементно, а построчно:

```
class Program
{
   static void Main()
   {
      int[][] MyArray;
      Console.Write("Введите количество строк: ");
      int n = int.Parse(Console.ReadLine());
      MyArray = new int[n][];
      for (int i = 0; i < MyArray.Length; i++)
      {
         Console.Write("Введите элементы {0} строки: ", i);
         string line = Console.ReadLine();
         string[] mas =
            line.Split(' ', StringSplitOptions.RemoveEmptyEntries);
         MyArray[i] = new int[mas.Length];
         for (int j = 0; j < MyArray[i].Length; j++)
         {
            MyArray[i][j] = int.Parse(mas[j]);
         }
      }
      Console.WriteLine("Введен массив:");
      Print( MyArray);
   }
```

```
        static void Print(int[][] mas)
        {
           for (int i = 0; i < mas.Length; i++)
           {
              foreach (int x in mas[i])
              {
                 Console.Write("{0} ", x);
              }
              Console.WriteLine();
           }
        }
     }
```

Результат работы программы:

```
Введите количество строк: 3
Введите элементы 0 строки: 2 4 5
Введите элементы 1 строки: 4 6 7 8
Введите элементы 2 строки: -3 4 5
Введен массив:
 2  4  5
 4  6  7  8
-3  4  5
```

Строковый тип StringBuilder

Строковый тип StringBuilder определен в пространстве имен System.Text и предназначен для создания строк, значение которых можно изменять. Объекты данного класса всегда создаются с помощью явного вызова конструктора класса, т.е., через операцию new. Создать объект класса StringBuilder возможно одним из следующих способов:

1) *//создание пустой строки, размер которой по умолчанию 16 символов*

   StringBuilder a =new StringBuilder();

2) *//инициализация строки и выделение памяти под 4 символа*

   StringBuilder b = new StringBuilder("abcd");

3) *//создание пустой строки и выделение памяти под 100 символов*

   StringBuilder c = new StringBuilder(100);

4) *//инициализация строки и выделение памяти под 100 символов*

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

5) *//инициализация подстрокой "bcd" и выделение памяти под 100 символов*

   StringBuilder d = new StringBuilder("abcdefg", 1, 3,100);

С объектами класса StringBuilder можно работать посимвольно:

```
using System;
//подключили пространство имен для работы с классом StringBuilder
using System.Text;
namespace Example
{
   class Program
   {
      static void Main()
      {
```

```
            StringBuilder a = new StringBuilder("кол около колокола");
            Console.WriteLine("Дана строка: {0}", a);
            char b = 'о';
            int k = 0;
            for (int x = 0; x < a.Length; x++)
            {
               if (a[x] == b)
               {
                  k++;
               }
            }
            Console.WriteLine("Символ {0} содержится в ней {1} раз",
                              b,k );
         }
      }
   }
```

Результат работы программы:

```
Дана строка: кол около колокола
Символ о содержится в ней 7 раз
```

Результат работы программы ничем не отличается от аналогичного примера для работы с типом string. А вот если теперь мы попытаемся заменить в данной строке все буквы о на точки, то такая проблема, как со строкой типа string, уже не возникнет:

```
static void Main()
{
   StringBuilder a = new StringBuilder("кол около колокола");
   Console.WriteLine("Дана строка: {0}", a);
   char b = 'о';
   char c = '.';
   for (int x = 0; x < a.Length; x++)
   {
      if (a[x] == b)
      {
         a[x] = c;
      }
   }
   Console.WriteLine("Преобразованная строка: {0}",a );
}
```

Результат работы программы:

```
Дана строка: кол около колокола
Преобразованная строка: к.л .к.л. к.л.к.ла
```

Для класса StringBuilder реализованы собственные члены, которые приведены в следующей таблице:

Название	Член класса	Описание
Append	Экземплярный метод	Производит добавление данных в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки.

Название	Член класса	Описание
AppendFormat	Экземплярный метод	Производит добавление форматированной строки в конец данной строки.
Capacity	Свойство	Позволяет получить и установить емкость буфера. Если устанавливаемое значение меньше текущей длины строки, или больше максимального, то генерируется исключение ArgumentOutOfRangeException.
Insert	Экземплярный метод	Производит вставку подстроки в заданную позицию.
Length	Изменяемое свойство	Возвращает длину строки. Присвоение ему значения 0 сбрасывает содержимое и очищает строку.
MaxCapacity	Неизменное свойство	Возвращает наибольшее количество символов, которое может быть размещено в строке.
Remove	Экземплярный метод	Производит удаление подстроки из заданной позиции.
Replace	Экземплярный метод	Производит замену всех вхождений заданной подстроки, или символа новой подстрокой, или символом.
ToString	Экземплярный метод	Преобразование в строку типа string.
Chars	Изменяемое свойство	Позволяет обратиться к элементу строки по его номеру. Вместо него можно пользоваться квадратными скобками [].
Equals	Экземплярный метод	Используется для сравнения значения двух строк. Метод возвращает true, только тогда, когда строки имеют одну и ту же длину и состоят из одних и тех же символов.
CopyTo	Экземплярный метод	Копирует подмножество символов строки в массив char.

Рассмотрим примеры использования данных членов класса.

```
class Program
{
   static void Main()
   {
      StringBuilder str=new StringBuilder("Площадь");
      Console.WriteLine("Максимальный объем буфера: {0} \n ",
                                 str.MaxCapacity);
      Print(str);
      str.Append(" треугольника равна");
      Print(str);
      str.AppendFormat(" {0:f2} см ", 123.456);
      Print(str);
```

```
            str.Insert(8, "данного ");
            Print(str);
            str.Remove(7, 21);
            Print(str);
            str.Replace("а", "…");
            Print(str);
            str.Length=0;
            Print(str);
         }
         static void Print(StringBuilder a)
         {
            Console.WriteLine("Строка: {0} ", a);
            Console.WriteLine("Текущая длина строки: {0} ", a.Length);
            Console.WriteLine("Объем буфера:  {0} ", a.Capacity);
            Console.WriteLine();
         }
      }
```

Результат работы программы:

```
Максимальный объем буфера: 2147483647
Строка: Площадь
Текущая длина строки: 7
Объем буфера: 16
Строка: Площадь треугольника равна
Текущая длина строки: 26
Объем буфера: 32
Строка: Площадь треугольника равна 123,46 см
Текущая длина строки: 37
Объем буфера: 64
Строка: Площадь данного треугольника равна 123,46 см
Текущая длина строки: 45
Объем буфера: 64
Строка: Площадь равна 123,46 см
Текущая длина строки: 24
Объем буфера: 64
Строка: Площ…дь р…вн… 123,46 см
Текущая длина строки: 30
Объем буфера: 64
Строка:
Текущая длина строки: 0
Объем буфера: 64
```

Все выполняемые действия относились только к одному объекту str. Никаких дополнительных объектов не создавалось.

Следует обратить внимание на то, что при увеличении текущей длины строки возможно изменение объема буфера, отводимого для хранения значения строки. А именно: если длина строки превышает объем буфера, то он увеличивается в два раза. Обратное не верно, т.е., при уменьшении длины строки буфер остается неизменным.

Задание

Самостоятельно изучите члены класса ToString, Chars, Equals, CopyTo.

Члены класса StringBuilder не содержат методы IndexOf, LastIndexOf, реализованные для класса string. Попробуем разработать их самостоятельно, взяв за основу алгоритм последовательного поиска.

```
// Метод IndexOf возвращает индекс первого вхождения
// подстроки str2 в строке str1 начиная с данной позиции.
// Если подстрока str2 в строке str1не найдена, то метод
// возвращает -1.
static int IndexOf (StringBuilder str1, StringBuilder str2, int n)
{
   int i,j;
   for(i = n; i < str1.Length;)
   {
      if (str1[i] == str2[0])
      {
         for (j = 1;j < str2.Length && i + j < str1.Length; j++)
         {
            if (str2[j] != str1[i+j])
            {
               break;
            }
         }
         if (j == str2.Length)
         {
            return i;
         }
         else
         {
            i++;
         }
      }
      else
      {
         i++;
      }
   }
   return -1;
}
```

Теперь, используя разработанный метод, можно после каждого вхождения подстроки x в строку str вставить подстроку y. Например, это можно сделать с помощью следующего фрагмента программы:

```
StringBuilder str = new StringBuilder("кол около колокола");
StringBuilder x = new StringBuilder("кол");
StringBuilder y = new StringBuilder("!!!");
int n = IndexOf(str, x, 0);
while (n != -1)
{
   str.Insert(n+x.Length, y);
   n = IndexOf(str, x, n + x.Length + y.Length);
}
Console.WriteLine(a);
```

Результат работы:

```
кол!!! окол!!!о кол!!!окол!!!а
```

Задания

1) *проверьте, будет ли данный фрагмент программы работать корректно, если строка x будет являтся подстрокой y и почему.*
2) *разработайте аналог метода LastIndexOf для работы с объектами StringBuilder.*

На практике часто комбинируют работу с изменяемыми и неизменяемыми строками. В качестве примера рассмотрим решение следующей задачи: дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания. Необходимо вывести все слова сообщения, которые начинаются и заканчиваются на одну и ту же букву.

```
class Program
{
   static void Main()
   {
      Console.WriteLine("Введите строку: ");
      StringBuilder a = new StringBuilder(Console.ReadLine());
      //удаляем из строк все знаки пунктуации
      for (int i = 0; i < a.Length;)
      {
         if (char.IsPunctuation(a[i]))
         {
            a.Remove(i, 1);
         }
         else
         {
            ++i;
         }
      }
      //преобразуем объект StringBuilder к типу string
      //и разбиваем его на массив слов
      string[] s = a.ToString().Split(' ');
      Console.WriteLine("Искомые слова: ");
      //перебираем все слова в массиве слов и выводим
      //на экран те, которые начинаются и
      //заканчиваются на одну и туже букву
      foreach (string str in s)
      {
         if (str[0] == str[str.Length-1])
         {
            Console.WriteLine(str);
         }
      }
   }
}
```

Результат работы программы:

```
Введите строку:
кол около колокола, а шалаш у ручья.
```

```
искомые слова:
около
а
шалаш
у
```

Задание

Измените программу так, чтобы она корректно работала и для случая, когда в исходной строке встречаются лишние пробелы.

Сравнение классов String и StringBuilder

Основное отличие классов String и StringBuilder заключается в том, что при создании строки типа String выделяется ровно столько памяти, сколько необходимо для хранения инициализирующего значения. Если создается строка как объект класса StringBuilder, то выделение памяти происходит с некоторым запасом. По умолчанию, под каждую строку выделяется объем памяти, равный минимальной степени двойки, необходимой для хранения инициализирующего значения, хотя можно задать эту величину по своему усмотрению. Например, для инициализирующего значения ”это текст” под строку типа String будет выделена память под 9 символов, а под строку типа StringBuilder –под 16 символов, из которых 9 будут использованы непосредственно для хранения данных, а еще 7 составят запас, который можно будет использовать в случае необходимости.

Если разработчик решит добавить в строку типа StringBuilder еще один символ, например, точку, то новая строка создаваться не будет. Вместо этого будет изменен уже существующий символ, находящийся в конце используемой части строки. Аналогичные действия будут выполнены при удалении, или изменении строки. Если объем добавлений превысит объем созданного запаса, произойдет создание новой строки, длина которой будет в два раза больше, чем длина предыдущей; в нее будет скопировано содержимое старой строки, и добавление продолжится.

У строки типа String такого запаса нет, поэтому при каждой попытке добавить новый символ будет создаваться новая строка требуемой длины.

Существование запаса памяти привело к тому, что у любого объекта класса StringBuilder есть два свойства, отвечающих за длину строки.

- Свойство Length действует по аналогии со свойством Length класса String и возвращает длину хранящихся в объекте текстовых данных.
- Свойство Capacity возвращает реальный объем, который занимает в памяти объект класса StringBuilder (без учета служебной информации).

Следующий пример демонстрирует различие между результатами, которые возвращают эти свойства:

```
class Program
{
   static void Main(string[] args)
   {
      string s = "Это текст";
      StringBuilder sb = new StringBuilder(s);
      Console.WriteLine("Длина строки \"{0}\" = {1}", s, s.Length);
      Console.WriteLine("Длина этой строки в StringBuilder = {0}",
                              sb.Length);
```

```
            Console.WriteLine("Реальная длина StringBuilder = {0}",
                                        sb.Capacity);
            Console.WriteLine("Максимальная длина StringBuilder = {0}",
                                      sb.MaxCapacity);
            Console.Read();
        }
    }
```

Результат работы программы:

```
Длина строки "Это текст" = 9
Длина этой строки в StringBuilder = 9
Реальная длина StringBuilder = 16
Максимальная длина StringBuilder = 2147483647
```

Использование StringBuilder позволяет сократить затраты памяти и времени центрального процессора при операциях, связанных с изменением строк. В то же время, на создание объектов класса StringBuilder также тратится некоторое время и память. Как следствие, в некоторых случаях операции по изменению строк оказывается «дешевле» производить непосредственно с самими строками типа String, а в некоторых – выгоднее использовать StringBuilder.

Пусть нам необходимо получить строку, состоящую из нескольких слов «текст» идущих подряд. Сделать это можно двумя способами.

Первый способ (прямое сложение строк):

```
string str = "";
for (int j = 0; j < count; j++)
{
   str += "текст";
}
```

Второй способ (использование StringBuilder):

```
StringBuilder sb = new StringBuilder();
for (int j = 0; j < count; j++)
{
   sb.Append("текст");
}
```

Затраты времени и памяти для каждого из случаев представлены в следующей таблице:.

Число слов	Длина строки	String		StringBuilder	
		Время	Затраты памяти	Время	Затраты памяти
2	10	0.091	15	0.18	16
3	15	0.163	30	0.22	16
4	20	0.252	50	0.373	48
5	25	0.336	75	0.39	48
6	30	0.464	105	0.463	48

Число слов	Длина строки	String		StringBuilder	
		Время	Затраты памяти	Время	Затраты памяти
7	35	0.565	140	0.591	112
8	40	0.695	180	0.663	112
9	45	0.809	225	0.692	112
10	50	0.965	275	0.731	112
15	75	1.779	600	1.125	240
20	100	2.697	1050	1.354	240
25	125	3.811	1625	1.571	240
30	150	5.045	2325	2.144	496
35	175	6.441	3150	2.359	496
40	200	7.992	4100	2.615	496
45	225	9.59	5175	2.799	496
50	250	11.45	6375	3.03	496
60	300	15.57	9150	4.074	1008
70	350	20.55	12425	4.683	1008
80	400	26.15	16200	5.036	1008
90	450	32.13	20475	5.419	1008

Замечание

Для получения более достоверных результатов каждое действие было выполнено 1000000 раз.

Первая колонка таблицы содержит в себе число слов, из которых состоит итоговой строка. Вторая колонка показывает длину итоговой строки в символах. Третья строка таблицы показывает, за какое время было осуществлено 1000000 операций создания строки заданной длины, а четвертая – количество памяти, которое было при этом затрачено. Пятая и шестая колонки показывают те же самые значения, но для случая использования класса StringBuilder.

В колонках «Затраты памяти» в обоих случаях стоят вычисленные эмпирически значения общего объема памяти, который был затрачен на создание строк соответствующей длины.

Как видно из этой таблицы, при числе операций сложения не больше пяти, выгоднее использовать прямое сложение строк, а если число операций сложения больше – имеет смысл использовать класс StringBuilder.

Практикум №8

Задание 1

Разработать программу, которая для заданной строки s:

1) подсчитывает общее число вхождений символов x и y;
2) определяет, какой из двух заданных символов встречается в строке чаще всего;
3) выводит на экран символы, которые наиболее часто встречается в строке;
4) выводит на экран символы, которые встречаются в строке только один раз;
5) определяет, имеются ли в строке два соседствующих одинаковых символа;
6) определяет, является ли строка палиндромом;
7) определяет, упорядочены ли по алфавиту символы строки;
8) подсчитывает количество букв в строке;
9) подсчитывает количество цифр в строке;
10) подсчитывает сумму всех содержащихся в строке цифр;
11) выводит на экран последовательность символов, расположенных до первого двоеточия;
12) выводит на экран последовательность символов, расположенных после последнего двоеточия;
13) выводит на экран последовательность символов, расположенных между круглыми скобками (считается, что в строке ровно одна пара круглых скобок);
14) находит самую длинную подстроку, состоящую только из цифр;
15) находит самую длинную подстроку, состоящую из повторяющегося символа.

Замечание

При решении задач использовать тип string.

Задание 2

Разработать программу, которая:

1) вставляет в строку символ x после каждого вхождения символа y;
2) вставляет в строку подстроку x после каждого вхождения подстроки y;
3) удваивает каждое вхождение заданного символа x;
4) удваивает каждое вхождение заданной подстроки x;
5) удаляет среднюю букву, если длина строки нечетная, и две средних, если длина строки четная;
6) удаляет все символы x;
7) удаляет из строки все цифры;
8) удаляет все подстроки substr;
9) заменяет все вхождения подстроки str1 на подстроку str2 (при этом str1 может являться частью str2);
10) заменяет все группы стоящих рядом точек на многоточие;
11) меняет местами первую букву со второй, третью с четвертой и т.д.;
12) меняет местами первую букву с последней, вторую с предпоследней и т.д.;
13) определяет, сколько различных символов встречается в строке;
14) удаляет из строки все подстроки, состоящие из цифр;
15) удаляет из строки самую длинную подстроку, состоящую из повторяющегося символа.

Замечание

При решении задач использовать класс StringBuilder.

Задание 3

Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания.

1. Вывести только те слова сообщения, в которых содержится заданная подстрока.
2. Вывести только те слова сообщения, которые содержат не более чем n букв.
3. Вывести только те слова сообщения, которые начинаются с прописной буквы.
4. Вывести только те слова сообщения, которые содержат хотя бы одну цифру.
5. Удалить из сообщения все слова, которые заканчиваются на заданный символ.
6. Удалить из сообщения все слова, содержащие данный символ (без учета регистра).
7. Удалить из сообщения все однобуквенные слова (вместе с лишними пробелами).
8. Удалить из сообщения все повторяющиеся слова (без учета регистра).
9. Подсчитать сколько раз заданное слово встречается в сообщении.
10. Подсчитать сколько слов, состоящих только из прописных букв, содержится в сообщении.
11. Найти самое длинное слово сообщения.
12. Найти все самые длинные слова сообщения.
13. Найти самое короткое слово сообщения.
14. Найти все самые короткие слова сообщения.
15. Вывести на экран все слова-палиндромы, содержащиеся в сообщении.
16. По правилу расстановки знаков препинания перед каждым знаком препинания пробел отсутствует, а после него обязательно стоит пробел. Учитывая данное правило, проверьте текст на правильность расстановки знаков препинания и, если необходимо, внесите в текст изменения.
17. Вывести только те слова, которые встречаются в тексте ровно один раз.
18. Вывести только те слова, которые встречаются более n раз.
19. Вывести слова сообщения в алфавитном порядке.
20. Вывести слова сообщения в порядке возрастания их длин.

Самостоятельная работа №5

Задание 1

Известны фамилия, имя и отчество пользователя. Найти его код личности. Правило получения кода личности: каждой букве ставится в соответствие число – порядковый номер буквы в алфавите. Эти числа складываются. Если полученная сумма не является однозначным числом, то цифры числа снова складываются до тех пор, пока не будет получено однозначное число. Например:

Исходные данные: Александр Сергеевич Пушкин
Код личности: (1+13+6+12+19+1+15+5+18) + (19+6+18+4+6+6+3+10+25) + (17+21+26+12+10+15) = 288 ⇒ 2+8+8=18 ⇒ 1+8=9

Задание 2

В шифре Цезаря алфавит размещается на круге по часовой стрелке. За последней буквой алфавита идет первая буква алфавита, т.е., после буквы «я» идет буква «а». При шифровании текста буквы заменяются другими буквами, отстоящими по кругу на заданное количество позиций (сдвиг) дальше по часовой стрелке. Например, если сдвиг равен 3, то буква «а» заменяется на букву «г», буква «б» на букву «д», а буква «я» на букву «в». Зашифровать сообщение, используя шифр Цезаря со сдвигом k.