СИМВОЛЫ И СТРОКИ в C#

Обработка текстовой информации является одной из самых распространенных задач современного программирования. C# предоставляет для ее решения широкий набор средств: символы char, неизменяемые строки string и изменяемые строки StringBuider. В данном разделе мы рассмотрим работу с символами, неизменяемыми и изменяемыми строками.

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.*

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

Строковый тип string

Тип string, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый тип класса System.String библиотеки .Net. Тип string относится к ссылочным типам.

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.

Создать объект типа string можно несколькими способами:

```
1) string s;                              // инициализация отложена
2) strings="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
   strings = x.ToString();                //преобразовали ее к типу string
5) strings=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
   string v=new string (a, 0, 2)          // этом:  0 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. Рассмотрим пример программы, в которой задается строка и подсчитывается количество букв (символов) 'o' в этой строке.

```
class Program
{
static void Main()
 {
     string a ="кол около колокола";
     Console.WriteLine("Дана строка: {0}", a);
        char b='o';
        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 );
   Console.ReadLine();
 }
}
```

Результат работы программы:
Дана строка: кол около колокола
Символ о содержится в ней 7 раз

Задание. Измените данную программу так, чтобы она запрашивала с клавиатуры исходную строку и символ, количество которых нужно определить. Если символа нет, то программа должна вывести сообщение «Символ не найден».

Изменим исходную задачу. Пусть теперь нужно заменить все символы о на букву а.

Попытка заменить в данной строке все вхождения буквы о на букву а (см. код ниже) ожидаемого результата не даст:

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

Относительно строки 1 компилятор выдаст сообщение об ошибке:
Property or indexer string.this[int] cannot be assigned to – it is read only

Обратите внимание! Компилятор запрещает напрямую изменять значение строки.

Для решения проблемы нужно создать новую строку и копировать в нее нужные символы из исходной.

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

Название	***Описание***	***Вид***
Compare	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с учетом или без учета регистра.	Статический метод
CompareTo	Сравнение текущего экземпляра строки с другой строкой.	Экземплярный метод
Concat	Слияние произвольного числа строк.	Статический метод
Copy	Создание копии строки.	Статический метод
Empty	Открытое статическое поле, представляющее пустую строку.	Статическое поле
Format	Форматирование строки в соответствии с заданным форматом.	Статический метод
IndexOf, LastIndexOf	Определение индекса первого или, соответственно, последнего вхождения подстроки в данной строке.	Экземплярные методы
IndexOfAny, LastIndexOfAny	Определение индекса первого или, соответственно, последнего вхождения любого символа из подстроки в данной строке.	Экземплярные методы
Insert	Вставка подстроки в заданную позицию.	Экземплярный метод
Join	Слияние массива строк в единую строку. Между элементами массива вставляются разделители.	Статический метод
Length	Возвращает длину строки.	Свойство
PadLeft, PadRigth	Выравнивают строки по левому или, соответственно, правому краю путем вставки нужного числа пробелов в начале или в конце строки.	Экземплярные методы
Remove	Удаление подстроки из заданной позиции.	Экземплярный метод
Replace	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом.	Экземплярный метод
Split	Разделяет строку на элементы, используя разные разделители. Результаты помещаются в массив строк.	Экземплярный метод
StartWith, EndWith	Возвращают true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой.	Экземплярные методы
Substring	Выделение подстроки, начиная с заданной позиции.	Экземплярный метод
ToCharArray	Преобразует строку в массив символов.	Экземплярный метод
ToLower, ToUpper	Преобразование строки к нижнему или, соответственно, к верхнему регистру.	Экземплярные методы
Trim, TrimStart, TrimEnd	Удаление пробелов в начале и конце строки или только с начала или только с конца соответственно.	Экземплярные методы

***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.

Например, если выполнить следующий фрагмент программы:

```
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

то компилятор никаких сообщений не выдаст, но мы не увидим никаких преобразований со строкой.
А вот в результате работы следующего фрагмента программы мы получим следующий результат:

```
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. Бороться 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. А вот если нужно изменять строку, то лучше пользоваться классом StringBuilder.

Подробнее о методах работы со строкам типа string см. С. Фролов «Самоучитель по C#» Глава 12. Работа с текстовыми строками. Стр. 368-405.

Пример. Приведенная ниже программа вводит строку символов с клавиатуры и формирует новую строку, в которой символы введенной строки переписаны в обратном порядке.

```
class Program
{
    static void Main(string[] args)
    {
        string str;   //описываем исходную строку str
        string str1="";   //описываем строку-результат str1 и задаем
                          // ей начальное значение – пустая строка
        Console.Write("Введите строку ");
        str = Console.ReadLine();       //вводим строку с клавиатуры
        int i;
        string bukva;
        for (i = 0; i < str.Length; i++)
        {
            bukva = str.Substring(i, 1);  // выделяем один символ из строки
            str1 = bukva + str1;          // добавляем этот символ к новой строке слева
        }
        Console.WriteLine("\nНовая строка = "+str1); //выводим результат
        Console.Read();
        }
}
}
```

Задание. Измените данную программу так, чтобы она проверяла, является ли введенное слово (фраза) палиндромом (т.е. читается одинаково слева направо и справа налево. Примеры палиндромов – ТОПОТ, РОТОР, ШАЛАШ …).