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. Свойство Length строки представляет число объектов Char, содержащихся в этой строке, а не число символов Юникода.

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"
```

.

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("Строки равны");
if (color1 != color2)   Console.WriteLine("Строки 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.

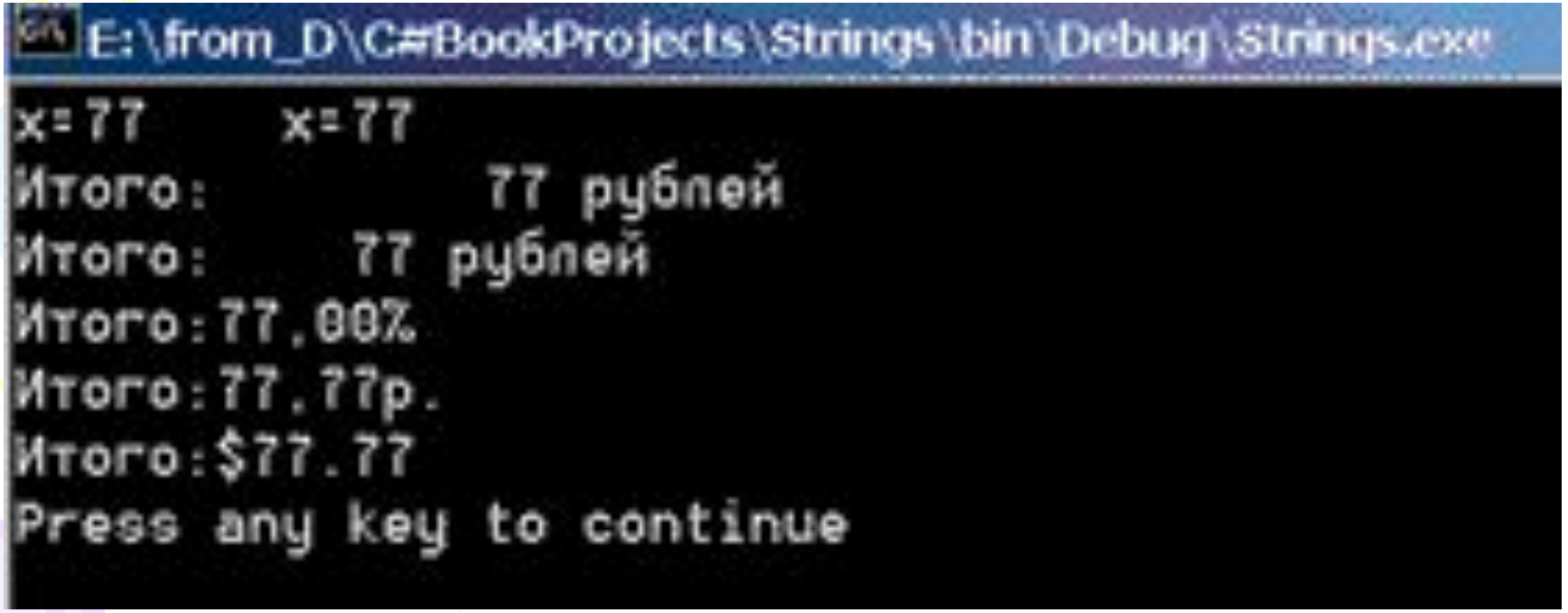
Общий синтаксис, специфицирующий формат, таков:

{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.

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

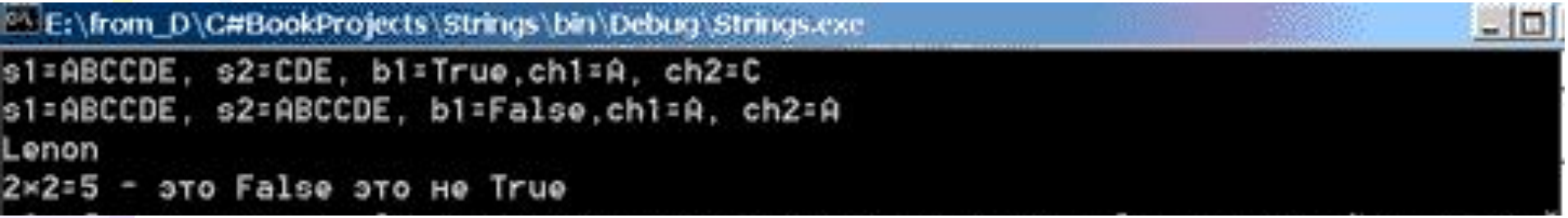
public StringBuilder Insert (int location,<объект>). Метод вставляет строку, полученную из объекта, в позицию, указанную параметром location. Метод Append является частным случаем метода Insert ;

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);
```



Примеры

Символы и массивы символов в C#

Пример 1: Использование методов структуры System.Char

```
B c 2
b C 2
Введите символ: A
Введен символ A, его код - 65
Буква
Введите символ: 5
Введен символ 5, его код - 53
Число
```

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
      { try
          { char b = 'B', c = '\x63', d = '\u0032';
              Console.WriteLine("{0} {1} {2}", b, c, d);
              Console.WriteLine("{0} {1} {2}", char.ToLower(b), char.ToUpper(c),
                  char.GetNumericValue(d));
              char a;
              do
              { Console.Write("Введите символ: "); a = char.Parse(Console.ReadLine());
                Console.WriteLine("Введен символ {0}, его код - {1}", a, (int)a);
                if (char.IsLetter(a)) Console.WriteLine("Буква");
                if (char.IsNumber(a)) Console.WriteLine("Число");
                if (char.IsWhiteSpace(a)) Console.WriteLine("Пробельный символ");
                if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");
              } while(a != 'q');

             // Console.ReadLine();
          }
          catch { Console.WriteLine("Возникло исключение"); }
          Console.ReadLine();
      }
  }
}
```

Символы и массивы символов в C#

Пример 2: Работа с массивом символов

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
      { char[] a = {'m','a','s','s','i','v'};
        char[] b = "abcdba".ToCharArray();
        PrintArray("Исходный массив a:", a);
        int pos = Array.IndexOf(a, 'm');
        a[pos] = 'M';
        PrintArray("Измененный массив a:", a);
        PrintArray("Исходный массив b:", b);
        Array.Reverse(b);
        PrintArray("Измененный массив b:", b);
        Console.ReadLine();
      }
    static void PrintArray(string h, Array a)
      { Console.WriteLine(h);
        foreach (object el in a)
              Console.Write(el);
        Console.WriteLine("\n");
      }
    }
}
```

```
Исходный массив a:
massiv

Измененный массив a:
Massiv

Исходный массив b:
abcdba

Измененный массив b:
abdcba
```

Строки типа string в C#

Пример 4: Объединение и сравнение строк

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
    { string s1 = "Привет"; string s2 = "мир";
      string s3 = s1 + " " + s2;
      string s4 = String.Concat(s3, "!!!");
      Console.WriteLine(s4);
      string s5 = "apple"; string s6 = "a day";
      string s7 = "keeps"; string s8 = "a doctor";
      string s9 = "away";
      string[] values = new string[] { s5, s6, s7, s8, s9 };
      String s10 = String.Join(" ", values);
      Console.WriteLine(s10);
      if (s1.Equals(s2)) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      if (String.Compare(s1, s2) == 0) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      if (s1.CompareTo(s2) == 0) Console.WriteLine("Строки равны");
          else Console.WriteLine("Строки не равны");
      Array.Sort(values);
      String s11 = String.Join(" ", values);
      Console.WriteLine(s11);
      Console.ReadLine();
    }
  }
}
```

```
Привет мир!!!
apple a day keeps a doctor away
Строки не равны
Строки не равны
Строки не равны
a day a doctor apple away keeps
```

Пример 7: Вставка строк, удаление и замена в строке

```
using System;
namespace ConsoleApplication
{ class Program
  { static void Main(string[] args)
    { string text = "Хороший день";
      string subString = "замечательный ";
      text = text.Insert(8, subString);
      Console.WriteLine(text);
      text = "Хороший день";
      // индекс последнего символа
      int ind = text.Length - 1;
      // вырезаем последний символ
      text = text.Remove(ind);
      Console.WriteLine(text);
      // вырезаем первые два символа
      text = text.Remove(0, 2);
      Console.WriteLine(text);
      text = "хороший день";
      text = text.Replace("хороший", "плохой");
      Console.WriteLine(text);
      text = text.Replace("о", "");
      Console.WriteLine(text);
      Console.ReadLine();
    }
  }
}
```

```
Хороший замечательный день
Хороший ден
роший ден
плохой день
плхй день
```

Контрольные вопросы

1. Строки типа string в C#: понятие, способы создания, допустимые операции, основные методы класса System.String. Примеры.
2. Строки класса StringBuider в C#: понятие, способы создания, методы класса StringBuider. Примеры.