## Класс String

## Построение строк

Самый простой способ построить символьную строку – воспользоваться строковым [литералом](http://professorweb.ru/my/csharp/charp_theory/level3/3_7.php). Например, в следующей строке кода переменной ссылки на строку str присваивается ссылка на строковый литерал:

string str = "Пример строки";

В данном случае переменная str инициализируется последовательностью символов "Пример строки". Объект типа string можно также создать из массива типа char. Например:

char[ ] h ={ ‘С',‘Г',‘К'};

string S2 = new string(h);

Как только объект типа string будет создан, его можно использовать везде, где только требуется строка текста, заключенного в кавычки.

Строки в Си-шарп – это объекты класса String, значением которых является текст. Для работы со строками в этом классе определено множество методов (функций) и в этом уроке мы рассмотрим некоторые из них.

Чтобы использовать строку, ее нужно сначала создать – присвоить какое-либо значение, иначе мы получим ошибку: "Использование локальной переменной "[имя переменной]", которой не присвоено значение". Объявим простую строку и выведем ее на экран:

static void Main(string[] args)

{

string s = "Hello, World!";

Console.WriteLine(s);

}

static void Main(string[] args)

{

string s;

Console.WriteLine(s); // ошибка, строка не создана

}

Для объединения (конкатенации) строк используется оператор "+".

string s = "Hello," + " World!";

Оператор "[]" используется для доступа (только чтение) к символу строки по индексу:

string s = "Hello, World!";

char c = s[1]; // 'e'

Свойство Length возвращает длину строки.

### Спецсимволы

Символ "\" является служебным, поэтому, чтобы использовать символ обратного слэша необходимо указывать его дважды "\\".

Символ табуляции – "\t"

Символ перевода строки – "\r\n"

Двойные кавычки – "\""

## Методы (функции) класса String для работы со строками в Си-шарп

### Как проверить, пуста ли строка?

Метод IsNullOrEmpty() возвращает True, если значение строки равно null, либо когда она пуста (значение равно ""):

static void Main(string[] args)

{

string s1 = null, s2 = "", s3 = "Hello";

String.IsNullOrEmpty(s1); // True

String.IsNullOrEmpty(s2); // True

String.IsNullOrEmpty(s3); // False

}

Метод IsNullOrWhiteSpace() работает как и метод IsNullOrEmpty(), только возвращает True еще и тогда, когда строка представляет собой набор символов пробела и/или табуляции ("\t"):

static void Main(string[] args)

{

string s1 = null, s2 = "\t", s3 = " ", s4 = "Hello";

String.IsNullOrWhiteSpace(s1); // True

String.IsNullOrWhiteSpace(s2); // True

String.IsNullOrWhiteSpace(s3); // True

String.IsNullOrWhiteSpace(s4); // False }

### Как проверить, является ли одна строка "больше" другой?

Для сравнивания строк используется метод Compare(). Суть сравнения строк состоит в том, что проверяется их отношение относительно алфавита. Строка "a" "меньше" строки "b", "bb" "больше" строки "ba". Если обе строки равны – метод возвращает "0", если первая строка меньше второй – "-1", если первая больше второй – "1":

static void Main(string[] args)

{

String.Compare("a", "b"); // возвращает -1

String.Compare("a", "a"); // возвращает 0

String.Compare("b", "a"); // возвращает 1

String.Compare("ab", "abc"); // возвращает -1

String.Compare("Romania", "Russia"); // возвращает -1

String.Compare("Rwanda", "Russia"); // возвращает 1

String.Compare("Rwanda", "Romania"); // возвращает 1

}

Чтобы игнорировать регистр букв, в метод нужно передать, как третий аргумент true.

String.Compare("ab", "Ab"); // возвращает -1

String.Compare("ab", "Ab", true); // возвращает 0

### Как перевести всю строку в верхний/нижний регистр?

Для этого используются методы ToUpper() и ToLower():

static void Main(string[] args)

{

string s = "Hello, SHC";

Console.WriteLine(s.ToUpper()); // выводит "HELLO, SHC"

Console.WriteLine(s.ToLower()); // выводит "hello, shc"

Console.ReadLine();

}

### Как проверить, содержит ли строка подстроку?

Для проверки содержания подстроки строкой используется метод Contains(). Данный метод принимает один аргумент – подстроку. Возвращает True, если строка содержит подстроку, в противном случае – False. Пример:

static void Main(string[] args)

{

string s = "Hello, SHC";

if (s.Contains("Hello"))

Console.WriteLine("Содержит");

Console.ReadLine();

}

Данная программа выводит слово "Содержит", так как "Hello, SHC" содержит подстроку "Hello".

### Как найти индекс первого символа подстроки, которую содержит строка?

Метод IndexOf() возвращает индекс первого символа подстроки, которую содержит строка. Данный метод принимает один аргумент – подстроку. Если строка не содержит подстроки, метод возвращает "-1". Пример:

static void Main(string[] args)

{

string s = "Hello, SHC";

Console.WriteLine(s.IndexOf("H")); // 0

Console.WriteLine(s.IndexOf("SHC")); // 7

Console.WriteLine(s.IndexOf("Zoo")); // -1

Console.ReadLine();

}

### Как узнать, начинается/заканчивается ли строка указанной подстрокой?

Для этого используются соответственно методы StartsWith() и EndsWith(), которые возвращают логическое значение. Пример:

static void Main(string[] args)

{

string s = "Hello, World";

Console.WriteLine(s.StartsWith("Hello")); // True

Console.WriteLine(s.StartsWith("SHC")); // False

Console.WriteLine(s.EndsWith("SHC")); // True

Console.ReadLine();

}

### Как вставить подстроку в строку, начиная с указанной позиции?

Метод Insert() используется для вставки подстроки в строку, начиная с указанной позиции. Данный метод принимает два аргумента – позиция и подстрока. Пример:

static void Main(string[] args)

{

string s = "Hello SHC ";

Console.WriteLine(s.Insert(5,",")); // вставляет запятую на 5 позицию

Console.ReadLine();

}

### Как обрезать строку, начиная с указанной позиции?

Метод Remove() принимает один аргумент – позиция, начиная с которой обрезается строка:

static void Main(string[] args)

{

string s = "Hello, SHC ";

Console.WriteLine(s.Remove(5)); // удаляем все символы, начиная с 5 позиции, на экран выведется "Hello"

Console.ReadLine();

}

В метод Remove() можно передать и второй аргумент – количество обрезаемых символов. Remove(3, 5) – удалит из строки пять символов начиная с 3-го.

### Как получить подстроку из строки, начиная с указанной позиции?

Для этого используется метод Substring(). Он принимает один аргумент – позиция, с которой будет начинаться новая подстрока:

static void Main(string[] args)

{

string s = "Hello, SHC ";

Console.WriteLine(s.Substring(7)); // получаем строку начиная с 7 позиции, выведет " SHC "

Console.ReadLine();

}

В метод Substring(), как в метод Remove() можно передать и второй аргумент – длина подстроки. Substring (3, 5) – возвратит подстроку длиной в 5 символов начиная с 3-й позиции строки.

### Как заменить в строке все подстроки указанной новой подстрокой?

Метод Replace() принимает два аргумента – подстрока, которую нужно заменить и новая подстрока, на которую будет заменена первая:

static void Main(string[] args)

{

string s = "Hello, SHC, Hello";

Console.WriteLine(s.Replace("Hello", "SHC")); //выведет " SHC, SHC, SHC "

Console.ReadLine();

}

### Как преобразовать строку в массив символов?

Метод ToCharArray() возвращает массив символов указанной строки:

static void Main(string[] args)

{

string s = "Hello, World";

char[] array = s.ToCharArray(); // элементы массива – 'H', 'e', 'l', 'l'…

}

### Как разбить строку по указанному символу на массив подстрок?

Метод Split() принимает один аргумент – символ, по которому будет разбита строка. Возвращает массив строк. Пример:

static void Main(string[] args)

{

string s = "Arsenal,Milan,Real Madrid,Barcelona";

string[] array = s.Split(','); // элементы массива – "Arsenal", "Milan", "Real Madrid", "Barcelona"

}

### Неизменяемые строки

Стоит знать, что объекты класса String представляют собой неизменяемые (Immutable) последовательности символов Unicode. Когда вы используете любой метод по изменению строки (например Replace()), он возвращает новую измененную копию строки, исходные же строки остаются неизменными. Так сделано потому, что операция создания новой строки гораздо менее затратна, чем операции копирования и сравнения, что повышает скорость работы программы. В Си-шарп также есть класс StringBuilder, который позволяет изменять строки. Об особенностях работы с ним мы поговорим в одном из дальнейших уроков.

## Класс StringBuilder

Когда строка конструируется классом String, выделяется ровно столько памяти, сколько необходимо для хранения данной строки. Однако, в пространстве имен System.Text имеется класс StringBuilder, который поступает лучше и обычно выделяет больше памяти, чем нужно в данный момент. У вас, как разработчика, есть возможность указать, сколько именно памяти должен выделить StringBuilder, но если вы этого не сделаете, то будет выбран объем по умолчанию, который зависит от размера начального текста, инициализирующего экземпляр StringBuilder. Класс StringBuilder имеет два главных свойства:

* Length, показывающее длину строки, содержащуюся в объекте в данный момент;
* Capacity, указывающее максимальную длину строки, которая может поместиться в выделенную для объекта память.

Любые модификации строки происходят внутри блока памяти, выделенного экземпляру StringBuilder. Это делает добавление подстрок и замену индивидуальных символов строки очень эффективными. Удаление или вставка подстрок неизбежно остаются менее эффективными, потому что при этих операциях приходится перемещать в памяти части строки. Выделять новую память и, возможно, полностью перемещать ее содержимое приходится только при выполнении ряда действий, которые приводят к превышению выделенной емкости строки. В дополнение к избыточной памяти, выделяемой изначально на основе экспериментов, StringBuilder имеет свойствоудваивать свою емкость, когда происходит переполнение, а новое значение емкости не установлено явно.

Рассмотрим пример:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

StringBuilder hello = new StringBuilder("Привет, меня зовут Александр Ерохин",120);

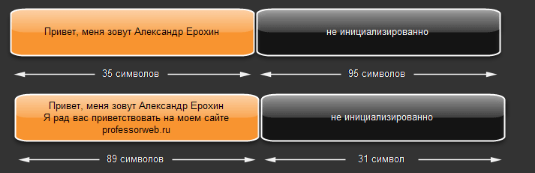
hello.AppendFormat("Я рад вас приветствовать на моем сайте professorweb.ru");

}

}

}

В данном примере начальная емкость StringBuilder равна 120. Всегда лучше сразу указывать емкость, превышающую предполагаемую длину строки, чтобы объекту StringBuilder не приходилось заново выделять память при переполнении. По умолчанию устанавливается емкость в 16 символов. Схематически данный пример работает следующим образом:



Т.е. при вызове метода AppendFormat() остаток текста помещается в пустое пространство без необходимости перераспределения памяти. Однако реальная эффективность, которую несет с собой применение StringBuilder, проявляется при выполнении повторных подстановок текста.

В следующей таблице перечислены основные методы класса StringBuilder:

|  |  |
| --- | --- |
| Методы класса StringBuilder | |
| Метод | Назначение |
| Append() | Добавляет строку к текущей строке |
| AppendFormat() | Добавляет строку, сформированную в соответствии со спецификатором формата |
| Insert() | Вставляет подстроку в строку |
| Remove() | Удаляет символ из текущей строки |
| Replace() | Заменяет все вхождения символа другим символом или вхождения подстроки другой подстрокой |
| ToString() | Возвращает текущую строку в виде объекта System.String (переопределение метода класса System.Object) |

Давайте рассмотрим практический пример использования класса StringBuilder:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

StringBuilder hello = new StringBuilder("Привет, меня зовут Александр Ерохин",120);

hello.AppendFormat(" Я рад вас приветствовать на моем сайте professorweb.ru");

// Зашифруем строку, хранящуюся в переменной hello

Console.WriteLine("Исходная строка: \n {0}\n",hello);

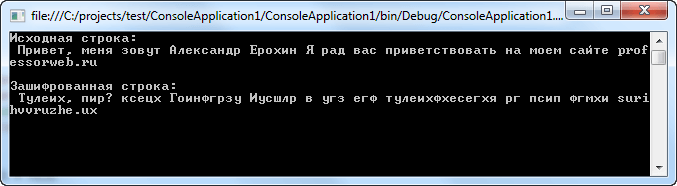
for (int i = 'я'; i >= 'a'; i--)

hello = hello.Replace((char)i,(char)(i+3));

Console.WriteLine("Зашифрованная строка:\n {0}",hello);

Console.ReadLine();

}}}



Форматирующие строки

Если вы хотите, чтобы разрабатываемые классы были дружественными к пользователю, они должны предлагать средства для отображения своих строковых представлений в любом из форматов, которые могут понадобиться пользователю. В исполняющей среде .NET определен стандартный способ достижения этого — интерфейс IFormattable. Описание способа добавления этого важного свойства к пользовательским классам и структурам и представляет собой тему настоящей статьи.

Начнем с рассмотрения того, что происходит, когда форматная строка применяется к примитивному типу, а отсюда станет ясно, как следует включать спецификаторы формата для пользовательских классов и структур:

decimal d = 12.05667m;

int i = 5;

Console.WriteLine("Значение переменной d = {0:C}, а i = {1}",d,i);

Сама строка формата содержит большую часть отображаемого текста, но всякий раз, когда в нее должно быть вставлено значение переменной, в фигурных скобках указывается индекс. В фигурные скобки может быть включена и другая информация, относящаяся к формату данного элемента, например, та, что описана ниже:

Количество символов, которое займет представление элемента, снабженное префиксом-запятой. Отрицательное число указывает, что элемент должен быть выровнен по левой границе, а положительное – по правой. Если элемент на самом деле занимает больше символов, чем ему отведено форматом, он отображается полностью.

Спецификатор формата предваряется двоеточием. Это указывает, каким образом необходимо отформатировать элемент. Например, можно указать, должно ли число быть форматировано как денежное значение, либо его следует отобразить в научной нотации.

В следующей таблице перечислены часто используемые спецификаторы формата для числовых типов:

|  |  |  |  |
| --- | --- | --- | --- |
| Спецификаторы формата для чисел | | | |
| Спецификатор | Применяется к | Значение | Пример |
| C | Числовым типам | Символ местной валюты | $835.50 (США) £835.50 (Великобритания) 835.50р.(Россия) |
| D | Только к целочисленным типам | Обычное целое | 835 |
| E | Числовым типам | Экспоненциальная нотация | 8.35Е+002 |
| F | Числовым типам | С фиксированной десятичной точкой | 835.50 |
| G | Числовым типам | Обычные числа | 835.5 |
| N | Числовым типам | Формат чисел, принятый в данной местности | 4,384.50 (Великобритания/США) 4 384,50 (континентальная Европа) |
| P | Числовым типам | Процентная нотация | 835,000.00% |
| X | Только к целочисленным типам | Шестнадцатеричный формат | 1a1f |

Стоит отметить, что полный список спецификаторов формата значительно длиннее, поскольку другие типы данных добавляют собственные спецификаторы.

### Форматирование даты и времени

Помимо числовых значений, форматированию нередко подлежит и другой тип данных: DateTime. Это структура, представляющая дату и время. Значения даты и времени могут отображаться самыми разными способами.

Форматирование даты и времени осуществляется с помощью спецификаторов формата. Конкретное представление даты и времени может отличаться в силу региональных и языковых особенностей и поэтому зависит от настройки параметров культурной среды. Спецификаторы формата даты и времени сведены в следующей таблице:

|  |  |
| --- | --- |
| Спецификаторы формата для дат | |
| Спецификатор | Формат |
| D | Дата в длинной форме |
| d | Дата в краткой форме |
| F | Дата и время в длинной форме |
| f | Дата и время в краткой форме |
| G | Дата — в краткой форме, время — в длинной |
| g | Дата и время — в краткой форме |
| М | Месяц и день |
| m | То же, что и М |
| O | Формат даты и времени, включая часовой пояс. Строка, составленная в формате О, может быть преобразована обратно в эквивалентную форму вывода даты и времени. Это так называемый "круговой" формат |
| R | Дата и время в стандартной форме по Гринвичу |
| s | Сортируемый формат представления даты и времени |
| T | Время в длинной форме |
| t | Время в краткой форме |
| U | Длинная форма универсального представления даты и времени; время отображается как универсальное синхронизированное время (UTC) |
| u | Краткая форма универсального представления даты и времени |
| Y | Месяц и год |

В приведенном ниже примере программы демонстрируется применение спецификаторов формата даты и времени:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

DateTime myDate = DateTime.Now;

Console.WriteLine("Дата в формате d: {0:d}\nВ формате D: {0:D}",myDate);

Console.WriteLine("Дата в формате f: {0:f}\nВ формате F: {0:F}", myDate);

Console.WriteLine("Дата в формате g: {0:g}\nВ формате G: {0:G}", myDate);

Console.WriteLine("Дата в формате m: {0:m}\nВ формате M: {0:M}", myDate);

Console.WriteLine("Дата в формате r: {0:r}\nВ формате R: {0:R}", myDate);

Console.WriteLine("Дата в формате o: {0:o}\nВ формате O: {0:O}", myDate);

Console.WriteLine("Дата в формате s: {0:s}", myDate);

Console.WriteLine("Дата в формате t: {0:t}\nВ формате T: {0:T}", myDate);

Console.WriteLine("Дата в формате u: {0:u}\nВ формате U: {0:U}", myDate);

Console.WriteLine("Дата в формате y: {0:y}\nВ формате Y: {0:Y}", myDate);

Console.ReadLine();

}

}

}

