



Web-разработка на C# и платформе Microsoft .NET

Работа со строками

План занятия

- Символы
- Строки
- StringBuilder
- Форматирование вывода

Ключевые методы класса Char

Метод

Действие

IsDigit()

А не цифра ли этот символ?

IsLetter()

А не буква ли этот символ?

IsPunctuation()

А не символ ли это пунктуации?

IsSeparator()

А не разделитель ли этот символ?

IsLower()

А не строчный ли это символ

IsUpper()

А не заглавный ли это символ?

ToLower()

Преобразовать к строчному

ToUpper()

Преобразовать к заглавному

```
bool b1 = char.IsDigit('1'); // true
```

```
char c1 = char.ToUpper('t'); // 'T'
```

```
bool b2 = char.IsPunctuation("Это строка", 3); // false
```

Способы создания строки

```
// объявление с инициализацией значением Сайт  
string word = "Сайт";
```

```
// строка в куче из пяти повторяющихся символов  
string sssss = new string('s', 5);
```

```
// string s2 = new string("s1"); так нельзя  
// string s2 = new string(); так тоже нельзя  
// а так можно:  
string s2; // это и есть объявление пустой строки в куче
```

```
// преобразование в символьный массив явно заданной строки  
char[] charArr= "Привет посетителям портала!".ToCharArray();
```

```
// объявление строки с инициализацией символами из массива  
string stryes = new string(charArr);
```

```
// объявление с копированием подстроки Привет  
// (с нулевого шесть символов)  
string strye = new string(charArr, 0, 6);
```

Ключевые свойства и методы

Метод

Действие

Length

Свойство, возвращает длину строки.

Contains()

Проверяет, содержится ли в строке подстрока

Format()

Позволяет задать форматирование строки.

Insert()

Вставляет в строку подстроку

Remove()

Удаляет из строки символы

Replace()

Заменяет в строке все вхождения подстроки на какой-либо другой текст

Substring()

Вырезает из строки подстроку

StartsWith()

Проверяет, начинается ли строка с подстроки

EndsWith()

Проверяет, заканчивается ли строка на подстроку

Ключевые свойства и методы

Метод	Действие
<code>ToCharArray()</code>	Конвертирует строку к массиву символов
<code>ToUpper()</code>	Переводит все символы строки в верхний регистр (в т.ч. символы нац. алфавитов).
<code>ToLower()</code>	Переводит все символы строки в нижний регистр. Корректно работает в том числе и с русскими символами.
<code>Trim()</code>	Удаляет из начала и конца строки пробельные, либо другие спецсимволы
<code>IndexOf()</code>	Определяет номер позиции первого вхождения подстроки в строку
<code>LastIndexOf()</code>	Определяет номер позиции последнего вхождения подстроки в строку

Строки – неизменяемые объекты!

- Потокбезопасность
- Неизменность
- Сокращение затрат памяти

```
string s = "Hello world!";  
s.Replace(' ', '_');  
Console.WriteLine(s);
```

```
string s = "Hello world!";  
s = s.Replace(' ', '_');  
Console.WriteLine(s);
```

Неправильная модификация строк

При каждой модификации строки создается отдельный объект

Будет создано n строк:

```
int n = 10000;  
string str = string.Empty;  
for (int i = 0; i < n; i++)  
{  
    str += "*";  
}
```



Правильная модификация строк

Для сложения большого числа строк используйте класс `StringBuilder`:

```
int n = 100000;  
var sb = new StringBuilder();  
for (int i = 0; i < n; i++)  
{  
    sb.Append("*");  
}  
string s = sb.ToString();
```

Сравнение String и StringBuilder

Число слов		Длина		String	String	StringBuilder	StringBuilder
Время	Затраты памяти	Время	Затраты памяти	Время	Затраты памяти	Время	Затраты памяти
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		
7	35	0.565	140	0.591	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		
50	250	11.45	6375	3.03	496		
90	450	32.13	20475	5.419	1008		

Методы и свойства класса StringBuilder

Метод	Действие
Length	Длина строки
Capacity	Емкость строки
Append()	Добавляет строку к текущей строке.
AppendFormat()	Добавляет форматированную строку
Insert()	Вставляет подстроку в строку
Remove()	Удаляет символы из текущей строки
Replace()	Заменяет в строке все вхождения подстроки на какой-либо другой текст
ToString()	Преобразует объект в строку

Когда сложение строк – не преступление

Литеральные строки

```
string s = "Это " +  
    "длинная текстовая " +  
    "строка " +  
    "которую не удобно " +  
    "записать в одной " +  
    "строке программы.";
```

```
string s  
= "Это длинная текстовая строка которую не удобно зап
```

Оператор @

Строка, помеченная
@ воспринимается буквально,
без учета управляющих символов:

```
string s1 = "c:\\temp\\myfile.txt";  
string s2 = @"c:\temp\myfile.txt";
```

```
string s3 = "Это текст\r\nна несколько\r\nстрок";  
string s4 = @"Это текст
```

```
на несколько  
строк";
```

Вывод данных

Отдельные значения

```
Console.WriteLine(x);
```

Ручное объединение строк

```
Console.WriteLine("x = " + x + ", y = " + y);
```

Форматирование

```
Console.WriteLine("x = {0}, y = {1}", x, y);
```

Управляющие символы

Вид	Наименование
<code>\a</code>	Звуковой сигнал
<code>\b</code>	Возврат на шаг назад
<code>\f</code>	Перевод страницы
<code>\n</code>	Перевод строки
<code>\r</code>	Возврат каретки
<code>\t</code>	Горизонтальная табуляция
<code>\v</code>	Вертикальная табуляция
<code>\\</code>	Обратная косая черта
<code>\'</code>	Апостроф
<code>\"</code>	Кавычки

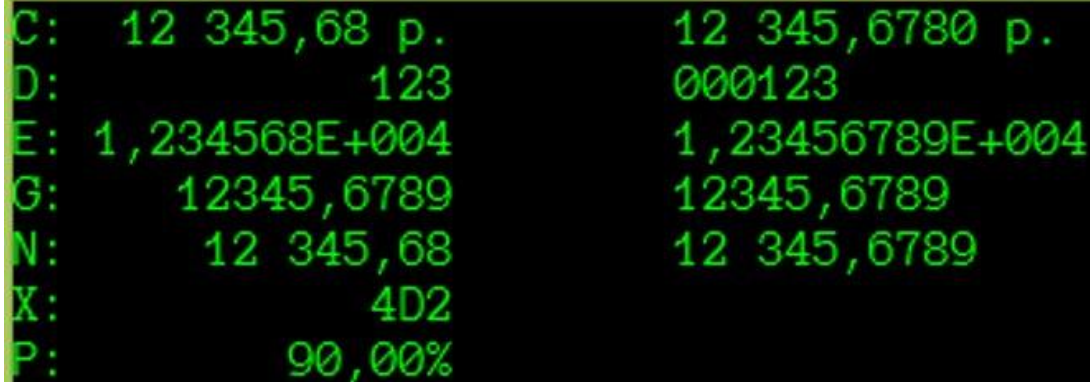
Параметры форматирования

Параметр	Формат
C	Финансовый (\$, €, p.)
D	Целочисленный
E	Экспоненциальный (научный)
F	Вещественный
G	Общий числовой
N	Стандартное форматирование
P	Процентный
X	Шестнадцатеричный

Параметры являются нечувствительными к регистру, то есть d и D – это одно и то же.

Пример форматирования

```
Console.WriteLine("C: {0,14:C} \t {0:C4}", 12345.678);  
Console.WriteLine("D: {0,14:D} \t {0:D6}", 123);  
Console.WriteLine("E: {0,14:E} \t {0:E8}", 12345.6789);  
Console.WriteLine("G: {0,14:G} \t {0:G10}", 12345.6789);  
Console.WriteLine("N: {0,14:N} \t {0:N4}", 12345.6789);  
Console.WriteLine("X: {0,14:X}", 1234);  
Console.WriteLine("P: {0,14:P}", 0.9);
```



C:	12 345,68 p.	12 345,6780 p.
D:	123	000123
E:	1,234568E+004	1,23456789E+004
G:	12345,6789	12345,6789
N:	12 345,68	12 345,6789
X:	4D2	
P:	90,00%	

Настройки форматирования

Описатель формата	Имя
0	Знак-заместитель нуля
#	Заместитель цифры
.	Разделитель
,	Разделитель групп и масштабирование чисел
'строка'	Разделитель строк-литералов
;	Разделитель секций

Пример расширенного форматирования

```
Console.WriteLine("{0:plus #####.0000;minus 0000.####;zero}",  
    123.456);  
Console.WriteLine("{0:plus #####.0000;minus 0000.####;zero}",  
    -123.456);  
Console.WriteLine("{0:plus #####.0000;minus 0000.####;zero}",  
    0.0);  
Console.WriteLine("{0,50:#### тысяч 000 целых.###}",  
    2222123.4);  
Console.WriteLine("{0,-50:#### тысяч 000 целых.###}",  
    123.4896318);
```

```
plus 123,4560  
minus 0123,456  
zero
```

```
2222 тысяч 123 целых,4
```

```
тысяч 123 целых,49
```

Пример форматирования даты

```
DateTime dt = new DateTime(2011, 3, 9, 16, 5, 7, 123);  
Console.WriteLine("t: {0:t}", dt);  
Console.WriteLine("T: {0:T}", dt);  
Console.WriteLine("d: {0:d}", dt);  
Console.WriteLine("D: {0:D}", dt);  
Console.WriteLine("f: {0:f}", dt);  
Console.WriteLine("F: {0:F}", dt);  
Console.WriteLine("g: {0:g}", dt);  
Console.WriteLine("G: {0:G}", dt);  
Console.WriteLine("m: {0:m}", dt);  
Console.WriteLine("y: {0:y}", dt);  
Console.WriteLine("o: {0:o}", dt);  
Console.WriteLine("s: {0:s}", dt);  
Console.WriteLine("u: {0:u}", dt);  
Console.WriteLine("U: {0:U}", dt);
```

```
t: 16:05  
T: 16:05:07  
d: 2011-03-09  
D: 9 марта 2011 г.  
f: 9 марта 2011 г. 16:05  
F: 9 марта 2011 г. 16:05:07  
g: 2011-03-09 16:05  
G: 2011-03-09 16:05:07  
m: 9 марта  
y: Март 2011  
o: 2011-03-09T16:05:07.1230000  
s: 2011-03-09T16:05:07  
u: 2011-03-09 16:05:07Z  
U: 9 марта 2011 г. 12:05:07
```

Настройки форматирования даты

Описатель формата	Описание
d	День
f	Доли секунд
F	Доли секунд (без нулей)
h	Часы в 12-часовом формате
H	Часы в 24-часовом формате
m	Минуты
M	Месяц
s	Секунды
y	Год
z	Смещение времени

Пример настройки форматирования даты

```
DateTime dt = new DateTime(2008, 3, 9, 16, 5, 7, 123);  
Console.WriteLine("{0:\\y-\\t y yy yyy yyyy}", dt);  
Console.WriteLine("{0:\\M-\\t M MM MMM MMMM}", dt);  
Console.WriteLine("{0:\\d-\\t d dd ddd dddd}", dt);  
Console.WriteLine("{0:\\h\\H-\\t h hh H HH}", dt);  
Console.WriteLine("{0:\\m-\\t m mm}", dt);  
Console.WriteLine("{0:\\s-\\t s ss}", dt);  
Console.WriteLine("{0:\\f-\\t f ff fff ffff}", dt);  
Console.WriteLine("{0:\\F-\\t F FF FFF FFFF}", dt);  
Console.WriteLine("{0:\\z-\\t z zz zzz}", dt);
```

```
y-      8 08 2008 2008  
M-      3 03 мар Март  
d-      9 09 Вс воскресенье  
hH-     4 04 16 16  
m-      5 05  
s-      7 07  
f-      1 12 123 1230  
F-      1 12 123 123  
z-      +4 +04 +04:00
```


Регулярные выражения

- Отладчик:

<http://regexr.com/>

- Классы:

Regex,
Match,
MatchCollection

- Полезные ссылки:

<http://www.regular-expressions.info>

[http://ru.wikipedia.org/wiki/Регулярные выражения](http://ru.wikipedia.org/wiki/Регулярные_выражения)

<http://debugger.ru/articles/nativeregexp>

Специальные символы (Метасимволы)

()

[]

{ }

\

^

-

\$

|

?

*

+

.

Метасимвол – это символ со специальным значением при поиске соответствий.

Чтобы использовать метасимвол как обычный, его нужно экранировать с помощью символа «обратный слеш» \

Или заключить в символы \Q ... \E

Перечисления

Код	Значение
[]	Перечисление символов
- (внутри перечисления)	интервал
	альтернатива
^ (в начале перечисления)	кроме

Коды количества

Код	Значение
*	Любое количество элементов. В том числе, ни одного.
*?	Не жадная *
+	Любое количество элементов, но хотя бы один должен быть.
+?	Не жадный +
?	Или есть или нет.
{n}, {n,m}, {n,}	Интервал или точное число символов

Предопределенные классы символов

Код	Значение
.	Любой символ
\d	Цифра ([0-9])
\D	Не цифра ([^0-9])
\s	Пробельный символ ([\t\n\x0B\f\r])
\S	Не пробельный символ ([^\s])
\w	«Цифробуква» ([a-zA-Z_\d])
\W	Не «цифробуква» ([^\w])

Позиции внутри строк

Код	Значение
^	Начало строки (в начале строки)
\$	Конец строки
\b	Граница слова
\B	Не граница слова
\G	Предыдущий успешный поиск

```

\ n)?[ \ t])*)(?:\. (?: (?: \ r
\ r \ n)?[ \ t])+|\Z| (?:[ \ " (
]))*"(?: (?: \ r \ n)?[ \ t])* \
]+(?: (?: (?: \ r \ n)?[ \ t])+|\
?: (?: \ r \ n)?[ \ t])*)(?: \. (?:
?: (?: \ r \ n)?[ \ t])+|\Z| (?:
?: \ r \ n)?[ \ t])*))*| (?:[ ^ ( ) <
?=[ \ " ( ) < @, ; ; \ \ " . \ ( \ ) ])|
\ t))*)* \< (?: (?: \ r \ n)?[ \ t
n)?[ \ t])+|\Z| (?:[ \ " ( ) < @
t))*)(?: \. (?: (?: \ r \ n)?[ \ t
\ t])+|\Z| (?:[ \ " ( ) < @, ; ; \
))*(?:, @ (?: (?: \ r \ n)?[ \ t]
t))+|\Z| (?:[ \ " ( ) < @, ; ; \ \
?: \ (?: (?: \ r \ n)?[ \ t])*)(?:

```

Если вы считаете, что можете решить
проблему с помощью регулярных
выражений, значит теперь у вас 2
проблемы... 😊

Пример – Проверка E-Mail (RFC 822)

СПАСИБО ЗА ВНИМАНИЕ