Обработка и форматирование строк

Классы для обработки символьных данных:

- Char – символьный тип данных
- String – строковый тип данных
- StringBuilder — построитель строк
- Regex – регулярное выражение

Char

Особенности:

- представляет **символ UTF-16** в Юникоде
- **тип-значение** (хранится в стеке)
- значение по умолчанию: **'\0'**
- размер: **2 байта**
- псевдоним: **char** (используется при создании переменных)
- при приведении к числовому типу возвращает код символа

Способы описания:

Способ	**Примеры**
символ	'Z', '0','я'
escape-последовательность (управляющая последовательность): начинаются с \	'\0', '\t', '\n'
escape-последовательность Юникода: символы **\u** после \u — шестнадцатеричное представление кода символа из четырех символов	'\u006A'
шестнадцатеричная escape-последовательность: символы **\x** после \x — шестнадцатеричное представление кода символа	'\x006A', '\x6A'
код символа, приведенный к char	(char)65 // 'A'

Методы типа данных Char:

1) методы проверки принадлежности к определенной категории символов:

- **Char.IsКатегория(**символ**)** — возвращает истину, если символ относится к указанной категории.

Категории: буква, регистр, цифра, число, разделитель, знак препинания, пробельный, управляющий

Проверка, что символ x — буква в нижнем регистре:

```
if (Char.IsLower(x))
{
        …
}
```

Проверка, что символ x — не цифра:

```
if (!Char.IsDigit(x))
{
        …
}
```

2) методы приведения к требуемому регистру:

- **Char.ToLower(**символ**)** — возвращает символ в нижнем регистре.
- **Char.ToUpper(**символ**)** — возвращает символ в верхнем регистре.

String

Особенности:

- **неизменяемый** массив символов
- **ссылочный** тип
- значение по умолчанию: **null** (область памяти под переменную не выделена)
- размер: **до 2 ГБ** (1 000 000 000 символов)
- псевдоним: **string** (используется при создании переменных)
- к символам строки можно обратиться как к элементу массива: **s[индекс элемента]**
- строки можно объединять, используя сложение: **s1+s2+...+sn**
- если строка начинается с @, то можно не экранировать \ (@"c:\Temp\pcs" вместо "c:\\Temp\\pcs")

Методы типа данных String:

Методы класса String <u>не изменяют</u> исходную строку.
Для изменения требуется присваивать строке значение, которое вернул метод:
строка = строка.Метод(параметры);

Сравнение строк	
s1.**CompareTo**(s2)	1 (s1>s2), 0 (s1=s2), -1 (s1<s2)
s1.**Equals**(s2)	Проверка равенства строк
s.**IsNullOrEmpty**()	Проверка, что строка null или пустая (String.Empty или "")
s.**IsNullOrWhiteSpace**()	Проверка, что строка null или пустая или состоит из пробельных символов
Поиск в строке	
s1.**Contains**(s2)	Проверка, что s1 содержит подстроку s2
s1.**StartsWith**(s2) / s1.**EndsWith**(s2)	Проверка, что подстрока s2 — начало / конец s1
s.**IndexOf**(…) / s.**LastIndexOf**(…)	Возвращает индекс первого / последнего вхожения символа или подстроки в s. Можно указать с какого и в скольки символах искать. Если совпадений нет, возвращает -1
s.**IndexOfAny**(…) / s.**LastIndexOfAny**(…)	Аналог IndexOf и LastIndexOf, но ищет любой символов из указанного массива символов
Разделение и соединение строк	
массив = s.**Split**(разделитель, …);	Возвращает массив, разделенный сепаратором (символом, строкой, массивом символов/строк). По умолчанию разделитель — пробел. Параметр StringSplitOptions позволяет исключить пустые строки и обрезать пробелы по краям. Можно указать, сколько элементов требуется вернуть
s = **String.Join**(разделитель, массив, …);	Объединяет элементы массива в строку, используя указанный разделитель (символ или строку)
s = **String.Concat**(строки);	Объединяет указанные строки в одну
s = s.**Substring**(длина);	Возвращает из строки подстроку указанной длины
Вставка, удаление и замена строк	
s.**Insert**(индекс, подстрока)	Возвращает строку, в которую вставлена подстрока
s.**Remove**(индекс, количество)	Возвращает строку, из которой удалены символы с указанного индекса в указанном количестве
s.**Replace**(s1, s2)	Возвращает строку, в которой подстроки или символы s1 заменены на s2
s.**ToLower**() / s.**ToUpper**()	Возвращает строку в нижнем / верхнем регистре
s.**Trim**() / s.**TrimStart**() / s.**TrimEnd**()	Возвращает строку без пробелов по краям / в начале / в конце строки. При указании параметров — удаляет указанные символы
s.**PadLeft**(длина) / s.**PadRight**длина)	Дополняет строку до указанной длины пробелами слева / справа. При указании символа использует его вместо пробелов

Форматирование строк:
String.Format(" {0} {1}", параметр0, параметр1)
Настройки формата: {номер параметра,количество символов выравнивания:спецификатор формата}

Примеры применения спецификаторов:
```
String.Format("{0:dddd MMMM}", DateTime.Now) // день и месяц текущей даты
Console.WriteLine("{0:C}", n); // вывод значения n в денежном виде (2 знака после запятой и валюта)
String.Format("часы = {0:hh}, минуты = {1:mm}", DateTime.Now) // часы и минуты текущей даты
```
Примеры применения выравнивания:
```
string myFName = "Fred";
string name1 = String.Format("|{0,10}|", myFName);      // |      Fred|
string name2 = String.Format("|{0,-10}|", myFName);     // |Fred      |
```

Интерполяция строк:
Если строка начинается с $, то вместо номера параметра можно указывать сам параметр или выражение.

```
Person person = new Person { Name = "Tom", Age = 23 };
Console.WriteLine($"Имя: {person.Name}  Возраст: {person.Age}");
string result = $"{x} + {y} = {x + y}";
```

StringBuilder

Особенности:
- **изменяемая** строка символов
- **ссылочный** тип
- находится в пространстве имен **System.Text**
- позволяет выделить область памяти (Capacity) больше текущей длины строки (Length)
- в отличие от string в StringBuilder **можно изменить символ** по его индексу

Пример работы с StringBuilder:
```
StringBuilder sb = new StringBuilder("привет мир");
sb[0] = 'П';                    // изменение первого символа
string s =  sb.ToString();      // s = Привет мир
```

Методы типа данных StringBuilder:
Методы класса StringBuilder изменяют исходную строку.

Методы	Пояснение
sb.**Insert**(индекс, подстрока);	Вставляет в строку в sb подстроку
sb.**Remove**(индекс, количество);	Удаляет из строки в sb символы с указанного индекса в указанном количестве
sb.**Replace**(s1, s2);	Заменяет в строке в sb все подстроки или символы s1 на s2
sb.**Append**(строка);	Добавляет строку в конец текущей строки в sb
sb.**AppendFormat**(строка);	Добавляет строку, сформированную в соответствии со спецификатором формата, в конец текущей строки в sb
string s = sb.**ToString**();	Возвращает строку из sb

Класс String рекомендуется использовать в следующих случаях:
- при небольшом количестве операций и изменений над строками
- при выполнении фиксированного количества операций объединения. В этом случае компилятор может объединить все операции объединения в одну
- когда надо выполнять масштабные операции поиска при построении строки.

Класс StringBuilder рекомендуется использовать в следующих случаях:
- при неизвестном количестве операций и изменений над строками во время работы программы
- когда предполагается, что приложению придется сделать множество подобных операций