В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

06 Часть 1

Строки.

Глобализация и Локализация

Благодарность Д.Д. Сагалову за помощь в переработке презентаций

Строки в C#

Строка (string, System.String) – это последовательная *неизменяемая* коллекция символов (типа char) в кодировке UTF-16, используемая для представления текста.

Строки в C# являются *ссылочными типами*.

Обратите внимание: максимальный размер строки в C# – 2Гб или в районе миллиарда символов.

Строковые литералы в C# записываются в двойных кавычках:

```
string line = "This is a string.";
```

https://docs.microsoft.com/ru-ru/dotnet/api/system.string?view=net-5.0

Создание Объектов Типа String

```
string line1 = "Это строка 1";
char[] chars = { 'М', 'а', 'с', 'с', 'и', 'в' };
string line2 = new string(chars);                    // "Массив"
string line3 = new string('W', 1);                   // "W"
string line4 = new string('7', 3);                   // "777"
string line5 = 242.ToString();                       // "242"
```

Такой код некорректен:

```
string line;
line += "asdfg"; // Ошибка компиляции – line не инициализирована.
```

https://docs.microsoft.com/ru-ru/dotnet/api/system.string.-ctor?view=net-5.0

Индексация Строк

Аналогично массивам, допускается обращение к отдельным символам строки по индексу, однако полученный символ ***доступен только для чтения***.

Дополнительно строки полностью поддерживают индексы и диапазоны:

```
using System;


string indexingDemo = "Yet another string...";
Console.WriteLine(indexingDemo[4]);
Console.WriteLine(indexingDemo[^4]);
Console.WriteLine(indexingDemo[4..]);
Console.WriteLine(indexingDemo[..^18]);
Console.WriteLine(indexingDemo[4..11]);
```

Вывод:

a
g
another string...
Yet
another

https://docs.microsoft.com/en-us/dotnet/api/system.string.chars?view=net-5.0

Конкатенация Строк

Конкатенация (объединение) строк может осуществляться с помощью **операций +** и **+=**:

```
string strConcat = "Hello";
strConcat = strConcat + " Software";
strConcat += " Engineering!";
Console.WriteLine(strConcat);
```

Вывод:
Hello Software Engineering!

Важно: т. к. строки в C# неизменяемы, конкатенация всегда ведёт к созданию новой строки. По этой причине крайне НЕ рекомендуется выполнять конкатенацию в цикле.

Строка как Контейнер Символов

Так как строка фактически является *неизменяемым* контейнером для символов, её можно обходить посимвольно с помощью циклов for и foreach:

```
using System;

string line1 = "example1";
for (int i = 1; i <= line1.Length; ++i) {
    Console.Write(line1[^i] + " ");
}
Console.WriteLine();
string line2 = "0123";
foreach (char letter in line2) {
    Console.Write(letter + "->" + (int)letter + " ");
}
string incorrect = "example3";
incorrect[0] = 'E'; // Ошибка компиляции - строка неизменяема.
```

Вывод:
1 e l p m a x e
0->48 1->49 2->50 3->51

Обход с использованием индексации с конца.

Массивы Строк

Как и в случае с другими типами в C#, Вы можете создавать массивы строк:

```
string[] lines = new string[] { "ноль", "один", "два", "три",
                 "четыре", "пять", "шесть", "семь", "восемь", "девять"};
```

Пример:

```
using System;

string[] starArray = new string[4];
for (int i = 0; i < starArray.Length; i++)
{
    starArray[i] = new string('*', i + 1);
}
foreach (string startSet in starArray)
{
    Console.Write("\t" + starSet);
}
```

Вывод:

```
*     **     ***     ****
```

Параметр Main() – Массив Строк

У Main может быть параметр – массив строк, передаваемый приложению в момент запуска.

При желании вы можете указать параметры командной строки в Visual Studio:
Project → имя_проекта Properties → Debug → Command line arguments

```
class Program {
    static void Main(string[] args) {
        int sum = 0;
        if (args.Length == 0) {
            System.Console.WriteLine("Нет аргументов в командной строке!");
            return;
        }
        for (int i = 0; i < args.Length; i++) {
            sum += int.Parse(args[i]);   // Осторожно: возможны исключения.
        }
        System.Console.WriteLine("Сумма чисел = " + sum);
    }
}
```

Строки в switch-Выражениях

Аналогично другим типам данных строки могут использоваться в выражениях switch:

```
static string TranslateDay (string day) => day.ToLower() switch
{
    "monday" => "понедельник",
    "tuesday" => "вторник",
    "wednesday" => "среда",
    "thursday" => "четверг",
    "friday" => "пятница",
    "saturday" => "суббота",
    "sunday" => "воскресенье",
    _ => throw new ArgumentException("This is not a valid day.");
};
```

switch-выражения доступны с C# 8.0 как вариант более лаконичного синтаксиса.

Символ «_» служит аналогом метки default при использовании данного синтаксиса.

Форматирование и Интерполяция Строк: Пример

```
string s = "Hello there.";
System.Console.WriteLine($"{s.ToUpper()}");   // К верхнему регистру.
System.Console.WriteLine("{0}", s);           // Исходная строка.
```

Вывод:
HELLO THERE.
Hello there.

```
using System;

string myPet = "Spot";
int age = 4;
string color = "black and white";
Console.WriteLine("My dog's name is {0}. He is {1} years old. His color is {2}.", myPet, age,
color);
Console.WriteLine($"My dog's name is {myPet}. He is {age} years old. His color is {color}.");
```

Вывод:
My dog's name is Spot. He is 4 years old. His color is black and white.
My dog's name is Spot. He is 4 years old. His color is black and white.

Спецификация Формата Строк

C# поддерживает форматирование строк через специальный синтаксис с фигурными скобками:

{ index[,alignment][:formatString[precisionSpecifier]] }
index – номер аргумента;
alignment – ширина поля;
formatString – спецификатор формата;
precisionSpecifier – спецификатор точности.

```
using System;
int num = 23, den = 6;
string result, // ссылка на строку, с результатом
form = "Числитель: {0}, знаменатель: {1}, дробь: {0}/{1} == {2}";
result = string.Format(form, num, den, num / den);
Console.WriteLine(result);
```

Вывод:
Числитель: 23, знаменатель: 6, дробь: 23/6 == 3

Спецификаторы Формата и Точности

Спецификатор	Формат	Роль спецификатора точности
C или c	Валютный	Количество десятичных разрядов.
D или d	Целочисленный	Минимальное число цифр.
E или e	Экспоненциальный	Число разрядов после точки.
F или f	С фиксированной точкой	Число разрядов после точки.
G или g	Наиболее короткий между E и F	Аналогично E и F
N или n	Численный с отступами	Минимальное число цифр.
P или p	Процентный	Количество десятичных разрядов.
R или r	Строка, посимвольно равная такому же числу	Игнорируется.
X или x	Шестнадцатеричный	Минимальное число цифр.

https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings

Пример Форматирования 1

```
double dou = 1234.567;
string form = "Спецификация E4: {0:E4}, спецификация F4: {0:F4}";
string result = string.Format(form, dou);
System.Console.WriteLine(result);
```

Вывод:
Спецификация E4: 1,2346E+003, спецификация F4: 1234,5670

```
int num = 23;
string form = "Основание 10: {0,-4:D}\r\nОснование 16: {0,4:X}";
System.Console.WriteLine(form, num);
```

Вывод:
Основание 10: 23
Основание 16: 17

```
int num = 23;
string form = "Основание 10: {0,-4:D3}\r\nОснование 16: {0,4:X3}";
System.Console.WriteLine(form, num);
```

Вывод:
Основание 10: 023
Основание 16: 017

Пример Форматирования 2

```
double completion = 0.57;
string form = "Процесс загрузки: {0:P4}";
System.Console.WriteLine(form, completion);
```

Вывод:
Процесс загрузки: 57,0000 %

```
decimal currency = 2365700000;
string form = "На Вашем счету: {0,3:C2}";
System.Console.WriteLine(form, currency);
```

Вывод:
На Вашем счету: 2 365 700 000,00 ₽

Члены Класса String-1

Член	Тип	Краткое описание
Length	Свойство	Возвращает количество символов в данной строке.
Contains	Метод	Возвращает true, если указанная подстрока присутствует в данной строке, иначе false.
ToLower	Метод	Возвращает копию строки, все символы которой переведены в нижний регистр.
ToUpper	Метод	Возвращает копию строки, все символы которой переведены в верхний регистр.
Insert	Метод	Возвращает копию строки, для которой в заданное место вставлена указанная строка.
Remove	Метод	Возвращает копию строки, из которой удалена указанная подстрока.
Replace	Метод	Возвращает копию строки, в которой заданная подстрока заменена другой подстрокой.
Substring	Метод	Возвращает обрезанную с заданной позиции копию строки.

Члены Класса String-2

Член	Тип	Краткое описание
Split	Метод	Возвращает массив строк, полученный путём деления указанной строки на подстроки с использованием заданных разделителей.
Join	Статический Метод	Объединяет несколько строк в одну через указанный разделитель.
Concat	Статический Метод	Производит конкатенацию двух и более строк, может конкатенировать строковое представление произвольных объектов.
Trim	Метод	Возвращает копию строки, из которой удалены все вхождения указанного(ых) символа(ов) в начале и в конце.
Format	Статический Метод	Возвращает отформатированную строку (подробнее).
Compare	Статический Метод	Осуществляет сравнение строк по заданным правилам.
Equals	Метод	Осуществляет проверку строк на равенство по заданным правилам.
ToCharArray	Метод	Копирует символы данной строки в массив символов.

Члены Класса String-3

Член	Тип	Краткое описание
IndexOf	Метод	Возвращает индекс первого вхождения подстроки в строку или -1.
LastIndexOf	Метод	Возвращает индекс последнего вхождения подстроки в строку или -1.
IndexOfAny	Метод	Возвращает индекс первого вхождения одного из перечисленных символов или -1.
StartsWith	Метод	Возвращает true, если указанная строка начинается с заданного символа/строки или false в противном случае.
EndsWith	Метод	Возвращает true, если указанная строка заканчивается заданным символом/строкой или false в противном случае.
PadLeft	Метод	Возвращает копию строки, в которую добавлено указанное количество символов слева.
PadRight	Метод	Возвращает копию строки, в которую добавлено указанное количество символов справа.

Метод Split() – Пример 1

```
using System;
string example = "5 great men attacks of 1997 happened over 20 years ago.";
Console.WriteLine(CountIntsInSplitString(example, " "));

static int CountIntsInSplitString(string s, params string[] delimiters) {
    int count = 0;
    if (delimiters.Length == 0) {
        delimiters = new[] { " " };
    }
    string[] splitString = s.Split(delimiters, StringSplitOptions.RemoveEmptyEntries);
    foreach (string word in splitString) {
        if (int.TryParse(word, out _)) {
            ++count;
        }
    }
    return count;
}
```

Пустые вхождения строки будут удаляться в результате разделения.

Если Вам не нужно сохранять результат в out-переменную, Вы можете таким образом проигнорировать его.

Метод Split() – Пример 2

```
using System;

string line = "hi there! this, is: a string.";
char[] delimiters = { ' ', '!', ',', ':', '.' };
string[] words = line.Split(delimiters,
StringSplitOptions.RemoveEmptyEntries);
Console.WriteLine($"Word Count: {words.Length}\r\nThe Words...");
foreach (string word in words)
{
    Console.WriteLine($"{word}");
}
```

Вывод:

Word Count: 6
The Words...
hi
there
this
is
a
string

Метод Join() – Пример

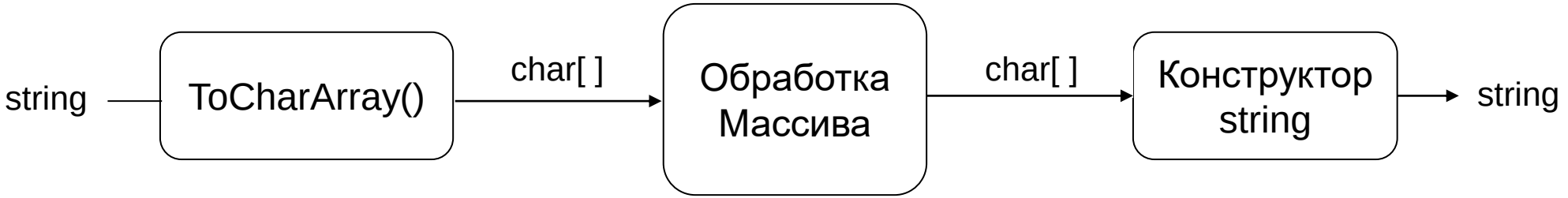
```
using System;

string sent = "De gustibus non est disputandum"; // О вкусах не спорят.
string[] words = sent.Split(' ');
Console.WriteLine("word.Length = " + words.Length);
foreach (string word in words)
{
    Console.Write("{0}\t", word);
}
sent = string.Join("-:-", words);
Console.WriteLine("\n" + sent);
```

Вывод:

word.Length = 5
De gustibus non est disputandum
De-:-gustibus-:-non-:-est-:-disputandum

https://docs.microsoft.com/ru-ru/dotnet/api/system.string.split?view=net-5.0
https://docs.microsoft.com/ru-ru/dotnet/api/system.string.join?view=net-5.0

Работа с Символами Строки как с Массивом char



Один из способов редактирования символов строки – использование метода ToCharArray() с последующим конструированием новой строки:

```
using System;

string row = "0123456789";
char[] reversed;
reversed = row.ToCharArray();
Array.Reverse(reversed);
row = new string(reversed);
Console.WriteLine(row);
```

Вывод:
9876543210

Сравнение Строк: Пример 1

```
public static int Compare (string strA, string strB)
```

```
using System;

string[] eng = { "one", "two", "three", "four" };
string res = eng[0];
foreach (string num in eng)
{
    if (string.Compare(res, num) < 0)
    {
        res = num;
    }
}
Console.WriteLine(res);
```

Вывод:
two

https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compare?view=net-5.0

Сравнение Строк: Пример 2

```
public static int Compare(string sA, string sB, bool ignoreCase, CultureInfo);
```

```
using System;
using System.Globalization;

string[] hens = {"Куропатка белая", "Куропатка тундровая",
                 "Тетерев", "Глухарь", "Рябчик" };
string res = hens[0];
foreach (string hen in hens)
{
    if (string.Compare(res, hen, true, new CultureInfo("ru")) > 0)
    {
        res = hen;
    }
}
Console.WriteLine(res);
```

Вывод:
Глухарь

StringBuilder в C#

StringBuilder – тип из пространства имён *System.Text*, предназначенный для динамического редактирования строк.

StringBuilder имеет преимущества над string в следующих ситуациях:

- Если строка многократно изменяется неизвестное количество раз (например, внутри цикла);
- Если необходимо вносить большое количество изменений в строку.

https://docs.microsoft.com/ru-ru/dotnet/api/system.text.stringbuilder?view=net-5.0

string: Генерация Строк (Плохая Реализация)

```
using System;

Random rnd = new();
// Генерация длинных строк через конкатенацию
// работает невероятно медленно и затратно по памяти.
for (int i = 0; i < 10; ++i) {
    Console.WriteLine(GetRandomString(1_000_000));
}
string GetRandomString(int len) {
    string result = "";
    result += (char)rnd.Next('A', 'Z' + 1);
    for (int i = 0; i < len - 1; ++i) {
        result += (char)rnd.Next('a', 'z' + 1);
    }
    return result;
}
```

Плохо! На каждой итерации создаётся новая строка, программа работает очень неэффективно для длинных строк.

StringBuilder: Генерация Строк (Хорошая Реализация)

```
using System.Text;

System.Random rnd = new();
// С StringBuilder генерация работает быстро даже для
// миллиона объектов - на каждой итерации строки не пересоздаются.
for (int i = 0; i < 10; ++i) {
    System.Console.WriteLine(GetRandomString(1_000_000));
}
string GetRandomString(int len) {
    StringBuilder sBuilder = new(len);
    sBuilder.Append((char)rnd.Next('A', 'Z' + 1));
    for (int i = 0; i < len - 1; ++i) {
        sBuilder.Append((char)rnd.Next('a', 'z' + 1));
    }
    return sBuilder.ToString();
}
```

В отличие от string, StringBuilder расширяется динамически, а не пересоздаётся на каждой итерации.

Связь Формата Вещественных Чисел и Локализации

```
static bool ReadDouble(string st, out double res) {
    if (double.TryParse(st, out res)) {
        return true;
    }
    string temp = null;
    int delimPos = st.IndexOf('.');          // Ищем точку.
    if (delimPos >= 0) {                     // В строке есть точка.
        temp = st.Replace('.', ',');         // Заменяем точку запятой.
    }
    if (double.TryParse(temp, out res)) {
        return true;
    }
    delimPos = st.IndexOf(',');              // Ищем запятую.
    if (delimPos >= 0) {                     // В строке есть запятая
        temp = st.Replace(',', '.');         // Заменяем запятую точкой.
    }
    return double.TryParse(temp, out res);
}
```

Метод для чтения вещественного американской (точка) и европейской (запятая) записей чисел.

Глобализация и Локализация

Локализация – адаптация приложения к национальным особенностям страны (региональные настройки).

Глобализация – адаптация приложения к работе с разными языками и региональными настройками.

Региональные настройки (региональные стандарты / culture) определяют:

- Язык;
- Символ валюты;
- Формат даты;
- Формат вывода чисел (точка/запятая).

В .Net для этих целей предусмотрены:

- механизм сателлитных сборок (satellite assemblies);
- классы из пространств имен:
  - **System.Globalization**;
  - **System.Resources**;
  - **System.Text**.

Региональные Настройки (Culture)

Региональные настройки (culture) идентифицируются строкой, содержащей *главный* (primary) и *вспомогательный* (secondary) тэги, или Int32 кодом (LCID - Local Culture IDentifier).

✓ Коды определены в стандарте Internet RFC 1766.

✓ Имена “язык” – “страна/регион” определены в стандартах ISO (International Standards Organization)

Региональные настройки делятся на три группы:

- **Invariant** – не зависят от языка и страны, имя “” (код != 0);
- **Neutral** – определяют только язык, например, “ru”, “en” (два символа в нижнем регистре);
- **Specific** – определяют язык и страну/регион, например, “en-CA”, “en-GB”, “ru-RU”, “tt-RU”.

Класс CultureInfo из System.Globalization

✓ Текущие региональные настройки определяются значениями двух свойств выполняемого потока:

Thread.CurrentCulture – формат даты/чисел/валюты Thread.CurrentUICulture – загружаемые ресурсы

✓ Свойства имеют тип CultureInfo.

```
[Serializable]
public class CultureInfo : ICloneable, IFormatProvider
```

✓ Ссылка на объекты класса CultureInfo передается как параметр методам, использующим информацию о региональных настройках (culture).

✓ Конструкторы класса System.Globalization.CultureInfo (4)

```
public CultureInfo( int culture );
public CultureInfo( string name );
public CultureInfo( int culture, bool useUserOverride );
public CultureInfo( string name, bool useUserOverride );
```

https://docs.microsoft.com/ru-ru/dotnet/api/system.globalization.cultureinfo?view=net-5.0

Класс CultureInfo

✓ С каждым региональным стандартом (culture) по умолчанию связаны конкретные форматы даты/числа/валюты. Пользователь может изменить эти установки в ControlPanel (Date, Time, Language, and Regional Options).

✓ Конструкторы с одним параметром инициализируют CultureInfo пользовательскими значениями форматов даты/числа/валюты.

✓ Конструкторы с двумя параметрами могут инициализировать CultureInfo как пользовательскими значениями форматов даты/числа/валюты (true), так и значениями по умолчанию (false).

✓ В классе CultureInfo определены свойства (более 20), связанные с региональными настройками и форматами даты/числа/валюты, например,

```
public virtual DateTimeFormatInfo DateTimeFormat {get; set;}
public virtual NumberFormatInfo NumberFormat {get; set;}
public virtual TextInfo TextInfo {get;}
public bool UseUserOverride {get;}
```

https://docs.microsoft.com/ru-ru/dotnet/api/system.globalization.cultureinfo.-ctor?view=net-5.0

Региональные Настройки и Метод ToString()

✓ В некоторых классах (например, Int32, Double) метод ToString() перегружен и использует значения свойства класса CultureInfo с информацией о текущих значениях региональных настроек (culture):

```
public virtual string ToString ( IFormatProvider provider );
public virtual string ToString ( string format,IFormatProvider provider );
```

✓ Интерфейс IFormatProvider.

```
public interface IFormatProvider
{ object GetFormat ( Type formatType ); }
```

реализуют классы CultureInfo, DateTimeFormatInfo, NumberFormatInfo, в которых есть информация о региональных стандартах (culture).

https://docs.microsoft.com/ru-ru/dotnet/api/system.globalization.cultureinfo?view=net-5.0

Вызовы ToString() с Использованием CultureInfo

```
using System;
using System.Globalization;

// CultureInfo для английского языка в USA.
Console.WriteLine(100.ToString("c", new CultureInfo("en-US")));

// CultureInfo для России, форматы по умолчанию.
Console.WriteLine(100.ToString("c", new CultureInfo("ru-RU", false)));

// CultureInfo для России, форматы из установок пользователя.
Console.WriteLine(100.ToString("c", new CultureInfo("ru-RU", true)));
```

Вывод:

$100.00
100,00 ₽
100,00 ₽

Класс CultureInfo и Настройки Потока

✓ Региональные настройки (culture) – это свойство потока.

✓ По умолчанию региональные настройки потока определяются значениями, заданными в профиле пользователя. Чтобы программно изменить региональные настройки, необходимо присвоить значение свойству потока.

✓ Свойства класса CultureInfo с информацией о текущих значениях региональных настроек (culture):

```
// Установки CurrentThread.
public static CultureInfo CurrentCulture {get;}

// Установки, которые использует ResourceManager.
public static CultureInfo CurrentUICulture {get;}

// Являются ли региональные настройки CurrentThread нейтральными?
public virtual bool IsNeutralCulture {get;}
```