Лекция 1

Программирование на C

Массивы. Работа со строками.

Преподаватель:

Потапова Елена Владимировна - доцент кафедры КИМ, к. т.н.

potapova1hv@mail.ru

https://vk.com/id581424425

Литература, Internet-источники

- **Герберт Шилдт** - C# 4.0. Полное руководство
- **Марченко А. Л.** Основы программирования на C# 2.0 : учеб. пособие / А. Л. Марченко, 2016. - 551 с.
- **Колдаев В. Д.** Основы алгоритмизации и программирования : учеб. Пособие / В. Д. Колдаев ; ред. Л. Г. Гагарина, 2015. - 413 с.
- **Биллиг В.А.** Основы программирования на C# . (Электронный ресурс: «Интуит» Национальный открытый университет. https://www.intuit.ru/studies/courses/2247/18/info)
- **Зиборов В.В.** - Visual C# 2012 на примерах
- **Культин Н.** "Microsoft Visual C# в задачах и примерах"

Сайты с теорией, примерами и обратной связью!

- https://metanit.com/sharp/tutorial/1.1.php
- http://mycsharp.ru/post/10/2013_04_19_massivy_v_si-sharp_klass_list.html

Объявление массива

тип[] имя_массива = new тип[размер массива];

Пример:

```
int[] my_array = new int[5]; // создаем массив целых чисел
string[] seasons = new string[4] {"зима","весна","лето","осень"};
//объявление массива строк и его инициализация значениями
```

- можно разделить на два отдельных оператора. Например:

```
int[] sample;
sample = new int[10];
```

- В какой ситуации это используется ?

Двумерные массивы

- Простейшей формой многомерного массива является двумерный массив.
- Местоположение любого элемента в двумерном массиве обозначается двумя индексами.
- Такой массив можно представить в виде таблицы, на строки которой указывает один индекс, а на столбцы - другой.
- В следующей строке кода объявляется двумерный массив integer размерами 3 x 4.

```
int[,] table = new int[ 3, 4];
```

Двумерные массивы: пример заполнения

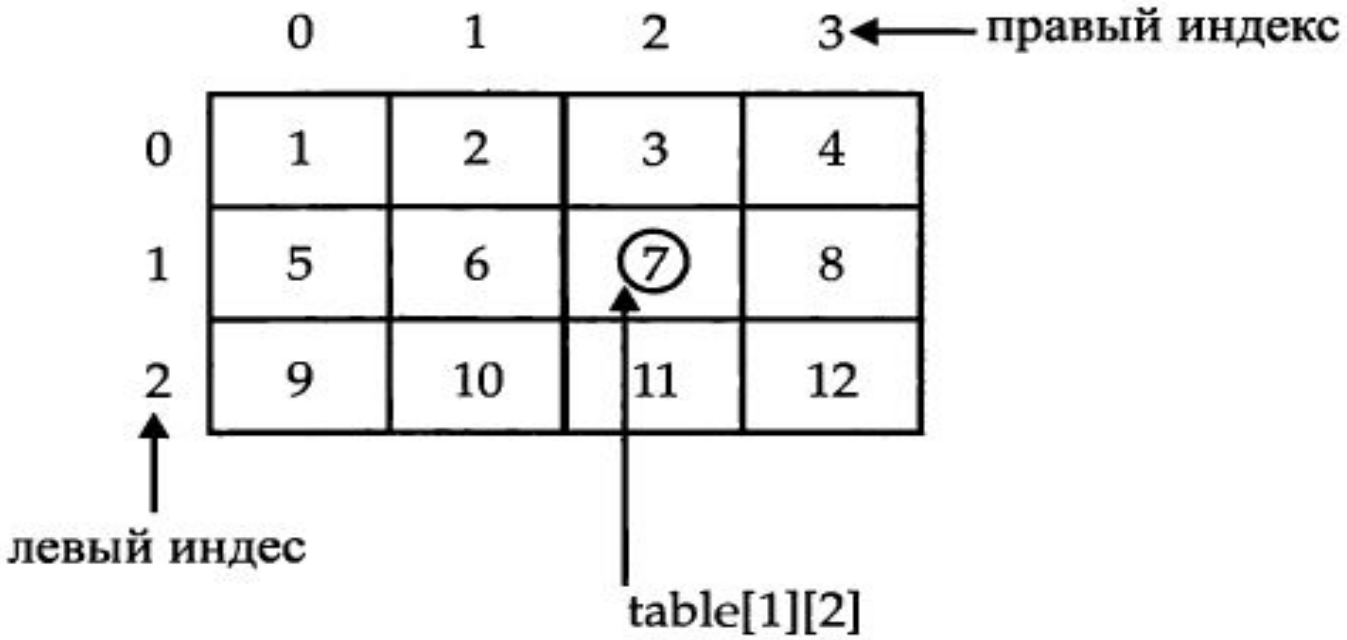

```
static void Main(string[] args)
{
    int i, j;
    int[,] table = new int[3, 4];
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            table[i, j] = (i * 4) + j + 1;
            Console.Write(table[i, j] + " ");
        }
        Console.WriteLine();
    }
}
```

В данном примере элемент массива table [0, 0] будет иметь значение 1,

элемент массива table [0,1] = 2,

элемент массива table [0, 2] = 3 и т.д.

А значение элемента массива table [2,3] окажется равным 12.



Циклы **for** и **foreach** для перебора элементов массива


```
int sum = 0;
int[] nums = new int[10];

// Задать первоначальные значения элементов массива nums.
for(int i = 0; i < 10; i++)
  nums[i] = i;

// Использовать цикл foreach для вывода значений
// элементов массива и подсчета их суммы.
foreach(int x in nums) {
  Console.WriteLine("Значение элемента равно: " + x);
  sum += x;
}

Console.WriteLine("Сумма равна: " + sum);
```

```
Значение элемента равно: 0
Значение элемента равно: 1
Значение элемента равно: 2
Значение элемента равно: 3
Значение элемента равно: 4
Значение элемента равно: 5
Значение элемента равно: 6
Значение элемента равно: 7
Значение элемента равно: 8
Значение элемента равно: 9
Сумма равна: 45
```

Класс System.Array : основные методы и свойства

- Свойство **Length** возвращает длину массива
- Свойство **Rank** возвращает размерность массива
- Статический метод **BinarySearch()** выполняет бинарный поиск в отсортированном массиве
- Статический метод **Clear()** очищает массив, устанавливая для всех его элементов значение по умолчанию
- Статический метод **Copy()** копирует часть одного массива в другой массив
- Статический метод **Exists()** проверяет, содержит ли массив определенный элемент
- Статический метод **Find()** находит элемент, который удовлетворяет определенному условию
- Статический метод **FindAll()** находит все элементы, которые удовлетворяют определенному условию
- Статический метод **IndexOf()** возвращает индекс элемента
- Статический метод **Resize()** изменяет размер одномерного массива
- Статический метод **Reverse()** располагает элементы массива в обратном порядке
- Статический метод **Sort()** сортирует элементы одномерного массива

Пример 1. Класс Array : его методы и свойства

```
int[] numbers = { -4, -3, -2, -1,0, 1, 2, 3, 4 };

// расположим в обратном порядке
Array.Reverse(numbers);

// уменьшим массив до 4 элементов
Array.Resize(ref numbers, 4);

foreach(int number in numbers)
{
    Console.Write($"{number} \t");
}
```

Результат: 4,3,2,1

Пример 2. Класс Array : его методы и свойства

```
int[] numbers = { -4, -3, -2, -1,0, 1, 2, 3, 4 };
int[] numbers2 = new int[5];

// копируем из numbers с 2-го индекса 5 элементов
// и поместим их в массив numbers2, начиная с 0-го индекса
Array.Copy(numbers, 2, numbers2, 0, 5);

foreach(int number in numbers2)
{
    Console.Write($"{number} \t");
}
```

Результат: -2, -1, 0, 1, 2

Пример 3. Класс Array : его методы и свойства

```
int[] numbers = { -3, 10, 0, -5, 12, 1, 22, 3};

Array.Sort(numbers);

foreach(int number in numbers)
{
    Console.Write($"{number} \t");
}
```

Результат: -5, -3, 0, 1, 3, 10, 12, 22

Класс Array : его методы и свойства

сумма элементов массива

```
int [] numbers = new int [] { 0, 1, 2, 3, 4, 5 };
int sum = 0;
foreach (int x in numbers)
{
sum += x;
}
//Выводим результат
Console.WriteLine("Сумма элементов массива: " + sum);
```

Или вообще без расчета!

```
Console.WriteLine("Сумма: " + numbers.Sum());
```

Соблюдение границ массива

- Границы массива в C# строго соблюдаются.
- Если границы массива не достигаются или же превышаются, то возникает ошибка при выполнении.
- Как только значение переменной **i** достигнет **10**, возникнет исключительная ситуация типа **IndexOutOfRangeException**, связанная с выходом за пределы индексирования массива, и программа преждевременно завершится.
- (обработка исключительных ситуаций)

```
static void Main(string[] args)
{
    int[] sample = new int[10];
    int i;

    //воссоздать превышение границ массива
    for (i = 0; i < 100; i++)
    {
        sample[i] = i;
    }

}
```

Создание динамического массива

- В языке c# под словом массив подразумевается объект с заранее заданным неизменяемым размером!
- Для создания **динамического массива** - метод Resize класса System.Array. **(Способ 1)**

```
static void Main(string [] args)
{
        int length = 0;
        int [] numbers = new int [5] {1,2,3,4,5};
        length = numbers.Length;                    //  5
        Array.Resize (ref numbers, numbers.Length + 2);
        length = numbers.Length;                    //  7
}                                  //Результат: {1,2,3,4,5,0,0} – 7 элементов
```

- Данный метод принимает два параметра: имя массива и его новый размер.
- В ходе выполнения метода, будет создан новый массив указанной нами длины, после чего все значения из старого массива будут скопированы в него. В результате мы получаем динамический массив.

Создание динамического массива

(Способ 2)

- Для создания **динамического массива** - обобщенный список **List<T>** класса **System.Collections.Generic**

```
static void Main(string [] args)
{
        int count = 0;
        List<int> numbers = new List<int>  { 1, 2, 3, 4, 5 };
        count = numbers.Count; //5
        numbers.Add(6);
        numbers.Add(7);
        count = numbers.Count; //7
        //преобразуем список в массив
        int [] numbers2 = numbers.ToArray<int>();
}
```

- Вместо массива используем список, который по умолчанию является динамическим.
- С помощью него выполняем всю необходимую работу и как только добиваемся нужного результата, то с помощью метода **ToArray** преобразуем список в массив.

Класс List

- Предназначен для работы со списками: можно добавлять, вставлять и удалять элементы (динамический массив).

```
static void Main(string[] args)
{
   List<string> teams = new List<string>();     // создание списка
   teams.Add("Barcelona");                       // добавление элемента
   teams.Add("Chelsea");
   teams.Add("Arsenal");
   List<string> teams2 = new List<string>() {"Dynamo", "CSKA" };   // инициализация
}
```

Класс List: добавление и удаление элементов

Метод	Описание
Add([элемент])	добавляет элемент в конец списка
AddRange([список элементов])	добавляет в конец списка элементы указанного списка
Insert([индекс],[элемент])	вставляет элемент на позицию соответствующую индексу, все элементы «правее» будут сдвинуты на одну позицию
InsertRange([индекс], [список элементов])	то же самое, только вставляется множество элементов

Remove([элемент])	удаляет первое вхождение указанного элемента из списка
RemoveRange([индекс], [количество])	удаляет указанное количество элементов, начиная с указанной позиции
RemoveAt([индекс])	удаляет элемент, который находится на указанной позиции
Clear()	удаляет все элементы списка

Класс List: пример

```
static void Main(string[] args)
{
   List<string> teams = new List<string>() { "Inter", "Milan", "Bayern", "Juventus"};
   teams.Insert(2,"Barcelona"); // вставляем в список элемент "Barcelona" на позицию 2
   teams.Remove("Milan"); // удаляем первое вхождение элемента "Milan" из списка
   List<string> newTeams = new List<string>() { "Liverpool", "Roma", "Borussia",
"Valencia" };
   teams.AddRange(newTeams); // добавляем в конец списка элементы списка newTeams
}
```

- Свойство Count соответствует свойству обычного массива – Length – количество элементов.

- Простые массивы работают быстрее, чем списки List !

```
enum Days
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}

enum Time : byte
{
    Morning = 1,
    Afternoon,
    Evening,
    Night
}
```

Кроме простых типов данных в C# есть такой тип как **enum** или ***перечисление.***

Перечисления представляют набор логически связанных **константа.**

Объявление перечисления происходит с помощью оператора **enum**.

Далее идет название перечисления, после которого указывается тип перечисления - он обязательно должен представлять целочисленный тип (byte, int, short, long).

Если тип явным образом не указан, то умолчанию используется тип int. Затем идет список элементов перечисления через запятую.

Каждому элементу перечисления присваивается целочисленное значение, причем первый элемент будет иметь значение 0, второй - 1 и так далее.

Мы можем также явным образом указать значения элементов, либо указать значение первого элемента, в этом случае каждый следующий элемент по умолчанию увеличивается на единицу

Кортеж - удобный способ для работы с набором значений!


```
var tuple = (5, 10);
```

кортеж tuple, который имеет два значения: 5 и 10. В дальнейшем мы можем обращаться к каждому из этих значений через поля с названиями Item[порядковый_номер_поля_в_кортеже]. Например:

```
static void Main(string[] args)
{
    var tuple = (5, 10);
    Console.WriteLine(tuple.Item1); // 5
    Console.WriteLine(tuple.Item2); // 10
    tuple.Item1 += 26;
    Console.WriteLine(tuple.Item1); // 31
    Console.Read();
}
```

Примеры объявления кортежей:

```
(int, int) tuple = (5, 10);

(string, int, double) person = ("Tom", 25, 81.23);
```

```
var tuple = (count:5, sum:10);
Console.WriteLine(tuple.count); // 5
Console.WriteLine(tuple.sum); // 10
```

Строки и класс System.String

Создание строк:

- 1 способ – через строковую переменную

```
string s1 = “Строки в C# весьма эффективны.”;
```

Строка str инициализируется последовательностью символов или пустой строкой.

```
string s2 = “ ”;
```

- 2 способ – через конструктор класса String создать строку из массива типа char.

```
string s3 = new String('a', 6); // результатом будет строка "aaaaaa“

char[ ] charray = {‘t’, ‘e’, ‘s’, ‘t’};
string s4  = new string (charray);
```

Объекты класса **System.String** представляют текст как последовательность символов Unicode. Максимальный размер объекта String может составлять в памяти 2 ГБ, или около 1 миллиарда символов.

Строки в C#

Строки являются объектами =>> имеют свойства и методы!

String – это имя стандартного для среды .NET строкового типа **System.String**

Это означает, что в C# строке как объекту доступны все методы, свойства, поля и операторы, определенные в классе **String.**

(!) После создания строки последовательность составляющих ее символов не может быть изменена.

Когда требуется получить строку как производную от уже существующей строки, создаем новую строку с помощью **методов преобразования исходной строки**.

Методы класса System.String

- **Compare**: сравнивает две строки
- **Contains**: определяет, содержится ли подстрока в строке
- **Concat**: соединяет строки
- **CopyTo**: копирует часть строки или всю строку в другую строку
- **EndsWith**: определяет, совпадает ли конец строки с подстрокой
- **Format**: форматирует строку
- **IndexOf**: находит индекс первого вхождения символа или подстроки в строке
- **Insert**: вставляет в строку подстроку
- **Join**: соединяет элементы массива строк
- **LastIndexOf**: находит индекс последнего вхождения символа или подстроки в строке
- **Replace**: замещает в строке символ или подстроку другим символом или подстрокой
- **Split**: разделяет одну строку на массив строк
- **Substring**: извлекает из строки подстроку, начиная с указанной позиции
- **ToLower**: переводит все символы строки в нижний регистр
- **ToUpper**: переводит все символы строки в верхний регистр
- **Trim**: удаляет начальные и конечные пробелы из строки

- **(неполный список !)**

Длина строки, посимвольный вывод строки


```
Console.WriteLine("str1: " + str1);
Console.WriteLine("Длина строки str1: " + str1.Length);
```

```
// Вывести строку str1 посимвольно.
Console.WriteLine("Вывод строки str1 посимвольно.");
for(int i=0; i < str1.Length; i++)
  Console.Write(str1[i]);
```

Что получим, если заменим оператор **Console.Write** на **Console.WriteLine ??**

Методы класса System.String

Метод **Split()** - разделит строку на части.

```
string str = “Один на суше, другой на море.”;
char[] seps = {‘ ‘, ‘.’, ‘,’};   // массив разделителей
string [] parts = str.Split(seps);
Console.WriteLine(“Результат разделения строки: “);
for (int i=0; i < parts.Length; i++)
    Console.WriteLine(parts[i]);
```

Выдача на Консоль:

```
Результат разделения строки:
Один
на
суше

другой
на
море
```

Замена символов или строк – метод **Replace()**:

```
string text = "хороший день";

text = text.Replace("хороший", "плохой");
Console.WriteLine(text);

text = text.Replace(‘о’, ‘*’);
Console.WriteLine(text);
```

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
}
```

Конкатенация

Конкатенация строк или объединение может производиться как с помощью операции +, так и с помощью метода **Concat:**

```
string s1 = "hello";
string s2 = "world";
string s3 = s1 + " " + s2; // результат: строка "hello world"
string s4 = String.Concat(s3, "!!!"); // результат: строка "hello world!!!"

Console.WriteLine(s4);
```

Обрезка строки - функция **Trim**:

```
string text = " hello world ";

text = text.Trim(); // результат "hello world"
text = text.Trim(new char[] { 'd', 'h' }); // результат "ello worl"
```

Функция **Trim** без параметров обрезает начальные и конечные пробелы и возвращает обрезанную строку.
Чтобы явным образом указать, какие начальные и конечные символы следует обрезать, мы можем передать в функцию массив этих символов.

Эта функция имеет частичные аналоги:
функция **TrimStart** обрезает начальные символы,
а функция **TrimEnd** обрезает конечные символы.

Заполнение и обрезка строк : методы PadLeft(), PadRight(), Trim()

```
static void Main(string[] args)
{
    string str = "тест";
    Console.WriteLine("Исходная строка: " + str);

    //Заполнить строку пробелами слева
    str = str.PadLeft(10);
    Console.WriteLine("|" + str + "|");

    //Обрезать пробелы
    str = str.Trim();
    Console.WriteLine("|" + str + "|");

    //Заполнить строку символами # слева
    str = str.PadLeft(10,'#');
    Console.WriteLine("|" + str + "|");

    //Заполнить строку символами # справа
    str = str.PadRight(20, '#');
    Console.WriteLine("|" + str + "|");

    //Обрезать символы #
    str = str.Trim('#');
    Console.WriteLine("|" + str + "|");
}
```

Вывод на экран:

```
Исходная строка: тест
|      тест|
|тест|
|##########тест|
|##########тест##########|
|тест|
```

Вставка, удаление и замена строк Методы Insert(), Remove(), Replace()

public string Insert(int *startIndex*, string *value*)

public string Remove(int *startIndex*)

public string Remove(int *startIndex*, int *count*)

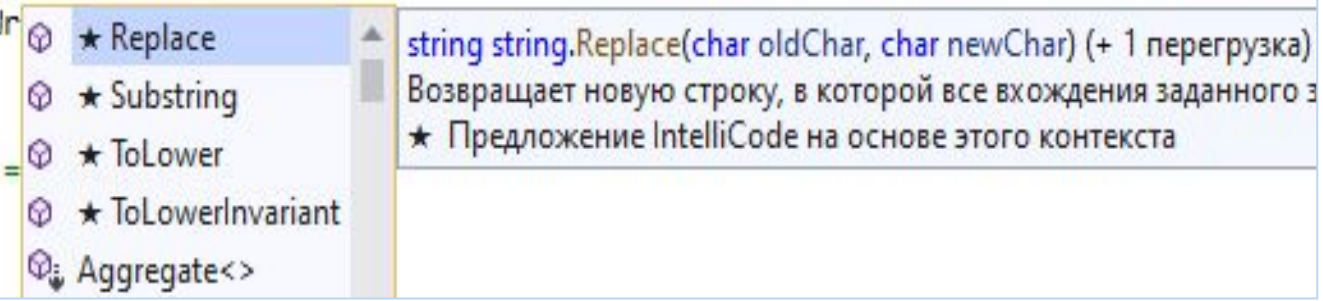
public string Replace(char *oldChar*, char *newChar*)

public string Replace(string *oldValue*, string *newValue*)

```
    //Заменить строку
    str = str.Replace("простой","непростой");
    Console.WriteLine(str);

    //Заменить символы в строке
    str = str.Replace('т','X');
    Console.WriteLine(str);

    //Удалить строку (часть строки)
    str = str.
    Console.Wr
}
/*
int[] sample =
int i;
```



Вставка, удаление и замена строк


```
static void Main(string[] args)
{
    string str = "Это тест";
    Console.WriteLine("Исходная строка: " + str);

    //Вставить строку
    str = str.Insert(4,"простой");
    Console.WriteLine(str);

    //Заменить строку
    str = str.Replace("простой","непростой");
    Console.WriteLine(str);

    //Заменить символы в строке
    str = str.Replace('т','Х');
    Console.WriteLine(str);

    //Удалить строку (часть строки)
    str = str.Remove(4, 5);
    Console.WriteLine(str);
}
```

Результат выполнения этой программы:

Исходная строка: Это тест
Это простой тест
Это непростой тест
Эхо непросХой ХесХ
Эхо сХой ХесХ

Поиск в строке - метод IndexOf()

определить индекс первого вхождения отдельного символа или подстроки в строке:

```
string s1 = "hello world";
char ch = 'o';
int indexOfChar = s1.IndexOf(ch); // равно 4
Console.WriteLine(indexOfChar);

string subString = "wor";
int indexOfSubstring = s1.IndexOf(subString); // равно 6
Console.WriteLine(indexOfSubstring);
```

метод **LastIndexOf** – находит индекс последнего вхождения символа или подстроки в строку

Поиск в строке - методы StartsWith и EndsWith

позволяют узнать начинается или заканчивается ли строка на определенную подстроку

Например, у нас есть задача удалить из папки все файлы с расширением exe:

```
string path = @"C:\SomeDir";

string[] files = Directory.GetFiles(path);

for (int i = 0; i < files.Length; i++)
{
    if(files[i].EndsWith(".exe"))
        File.Delete(files[i]);
}
```

Обратите внимание на использование слешей в именах файлов. (Работа с файлами будет в Лекции 3)
Либо мы используем двойной слеш: `"C:\\"`, либо одинарный, но тогда перед всем путем ставим знак @: `@"C:\Program Files"`

Форматирование строк : вывод в консоль


```
class Program
{
    static void Main(string[] args)
    {
        Person person = new Person { Name = "Tom", Age = 23 };

        Console.WriteLine("Имя: {0}  Возраст: {1}", person.Name, person.Age);
        Console.Read();
    }
}

class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

В строке "Имя: {0} Возраст: {1}" на место `{0}` и `{1}` затем будут вставляться в порядке следования `person.Name` и `person.Age`

Форматирование строк: метод String.Format


```
string output = String.Format("Имя: {0}  Возраст: {1}",
                        person.Name, person.Age);
Console.WriteLine(output);
```

Вывод отформатированной строки может осуществляться как в консоль, так и в текстовые элементы управления формы.

Метод `Format` принимает строку с плейсхолдерами типа {0}, {1} и т.д., а также набор аргументов, которые вставляются на место данных плейсхолдеров. В итоге генерируется новая строка.

В методе Format могут использоваться различные спецификаторы и описатели, которые позволяют настроить вывод данных.

Таблица форматов : https://metanit.com/sharp/tutorial/7.5.php

Например, для форматирования валюты используется описатель "C":

```
double number = 23.7;
string result = String.Format("{0:C}", number);
Console.WriteLine(result); // $ 23.7
string result2 = String.Format("{0:C2}", number);
Console.WriteLine(result2); // $ 23.70
```

Интерполяция строк

Начиная с версии языка C# 6.0 была добавлена такая функциональность, как интерполяция строк. Эта функциональность призвана заменить форматирование строк.

```
Person person = new Person { Name = "Tom", Age = 23 };

Console.WriteLine($"Имя: {person.Name}  Возраст: {person.Age}");
```

Знак доллара перед строкой указывает, что будет осуществляться интерполяция строк. Внутри строки опять же используются плейсхолдеры {...}, только внутри фигурных скобок уже можно напрямую писать те выражения, которые мы хотим вывести.

```
int x = 8;
int y = 7;
string result = $"{x} + {y} = {x + y}";
Console.WriteLine(result); // 8 + 7 = 15
```

Задачи:

- Лаб 1 : Создать программу, для анализа текстовой строки с условием квадратного уравнения. Исходное уравнение вводится в виде текстовой строки определенного формата: **a*x*x + b*x + c =0.**

- Лаб 2 : Создать программу для обработки текста, подгружаемого из внешнего файла в формате .txt.

По выбору пользователя должна выполняться одна из четырех задач:

- (1) найти самое короткое предложение;
- (2) заменить цифры от 1 до 9 на слова;
- (3) найти все слова, содержащие букву ‘а’
- (4) разбить самое короткое предложение так, чтобы каждое слово было записано в отдельной строке.