

# Знакомство с языками программирования.

## Лекция №1

### Тезисы:

- .NET аналог платформы JAVA.
- Язык программирования - это инструмент.
- Синтаксис учится легко.
- Программист - человек которые пишет код/алгоритм.

### Первые шаги написания программы:

- Получить задачу
- Обдумать условие
- Уточнить условие
- Составить алгоритм

### Программа состоит.

- Интерфейс
- Логика
- Данные

### Синтаксис:

dotnet new console - создать новый проект

dotnet run - запустить программу

write() - вывод в одну строку

writeline() - в конце перейти на новую строку

readline() - считать строку из терминала

.ToLower() - перевод всех символов в строке в нижний регистр

int - целые числа

double - вещественные числа

string - строки

bool - логический тип данных правда/ложь

new Random().Next(min,max) - даст случайное целое число от min до max-1.

## Лекция №2

### Функции:

- Используются в математике, чтобы постоянно не писать один и тот же код.
- В C# нету функций, есть методы.
- Next(1,10) ; WriteLine() - это всё функции.

Функция - часть программного кода, который создает разработчик.

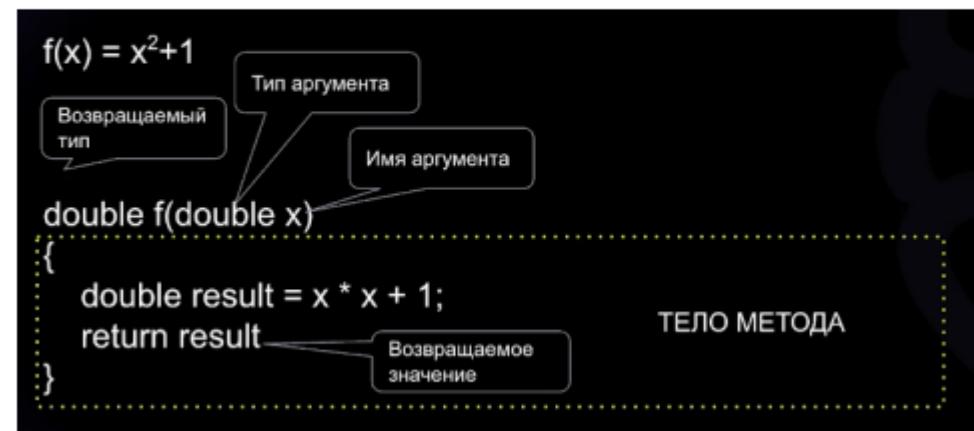
1. Функция имеет имя.
2. Может иметь входные аргументы.
3. Может возвращать значение.

*Общий вид функций. Что пишется в фигурных скобках:*

### Функции в программировании

```
ВозвращаемыйТипДанных ИмяМетода([ТипДанных1 ИмяАргумента1, ... ])  
{  
    Тело Метода  
    return ЗначениеСоответствующееВозвращаемомуТипуДанных;  
}
```

*Тело метода:*



С помощью массивов, можно избавиться от большого количества переменных.

```
ТИП ДАННЫХ[] ИМЯ = {Значение1, Значение2, ...}  
int[] array = { 9, 28, 1, 32, 1990 }

int[] array = { 0, 0, 0, 0, 0 }
int[] array = new int[5];
int[] array = new int[]{ 0, 0, 0, 0, 0 }
int[] array = new int[5]{ 1, 0, 2, 0, 3 }
```

## Лекция №3

Методы бывают двух видов:

1. Вид 1:

Они ничего не принимают и ничего не возвращают.

2. Вид 2:

Что-то принимают, ничего не возвращают.

Вид 1 и Вид 2 это void-методы.

3. Вид 3:

Ничего не принимают, что-то возвращают.

4. Вид 4:

Что-то принимают, что-то возвращают.

**Вот так выглядит ВИД 1 на практике!.**

```
// // // Вид 1. Описание
void Method1();
{
    Console.WriteLine('Автор ...');
}
// Вызов
Method1();
```

**Вот так выглядит ВИД 2 на практике!.**

```
// // // Вид2. Описание
void Method2(string msg)
{
    Console.WriteLine(msg);
}
// Вызов
Method2(msg: "Текст сообщения");

// Описание
void Method21(string msg, int count)
{
    int i = 0;
    while (i < count)
    {
        Console.WriteLine(msg);
        i++;
    }
}
// Вызов
Method21(msg: "Текст", count: 4);
Method21(count: 4, msg: "новый текст");
```

**Вот так выглядит ВИД 3 на практике!.**

```
// // // Вид 3. Описание
int Method3()
{
    return DateTime.Now.Year;
}
// Вызов
int year = Method3();
//Console.WriteLine(year);
```

Если метод что-то возвращает, то надо обязательно указать тип данных, значение которого мы ожидаем.

**Вот так выглядит ВИД 4 на практике!.**

```
// // // Вид4. Описание
// string Method4(int count, string text)
// {
//     int i = 0;
//     string result = String.Empty;

//     while (i < count)
//     {
//         result = result + text;
//         i++;
//     }
//     return result;
// }

string Method4(int count, string text)
{
    string result = String.Empty;
    for (int i = 0; i < count; i++)
    {
        result = result + text;
    }
    return result;
}

string res = Method4(10, "z");
Console.WriteLine(res);
```

**Цикло FOR:**

Вводим в программу for, включаем инициализацию счетчика, далее идет проверка условия, учитывается инкремент (увеличение счетчика).

# Цикл в цикле.

Демонстрация: написание программы по выводу таблицы умножения на экран. Далее демонстрация обновления программы так, чтобы таблица умножения выводилась не одним единым столбцом, а с пробелами.

```
for(int i = 0; i < 10; i++)
{
    for(int j = 0; j < 10; j++)
    {
        Console.WriteLine(i*j)
    }
    Console.WriteLine()
}
```

```
90  for (int i = 2; i <= 10; i++)
91  {
92      for (int j = 2; j <= 10; j++)
93      {
94          Console.WriteLine($"{i} x {j} = {i * j}");
95      }
96 }
```

## Решение задачи на использование цикла for

**Условие задачи:** Дан текст. В тексте нужно все пробелы заменить черточками, маленькие буквы “к” заменить большими “К”, а большие “С” заменить маленькими “с”.

Что нужно сделать первым делом? → Уточнить задачу и понять, ясна ли задача. Что значит “Дан текст”? Что значит “черточками”? Какого алфавита? Маленькие буквы “к” заменить большими “К”, а большие “С” заменить маленькими“с”

```
string Replace(string text, char oldValue, char newValue)
{
    string result = String.Empty;
    int length = text.Length;
    for (int i = 0; i < length; i++)
    {
        if(text[i] == oldValue) result = result + $"{newValue}";
        else result = result + $"{text[i]}";
    }
    return result;
}

string newText = Replace(text, ' ', '|');
Console.WriteLine(newText);
Console.WriteLine();
newText = Replace(newText, 'К', 'K');
Console.WriteLine(newText);
```

### Текст, в котором надо внести изменения:

```
string text = "- Я думал, — сказал князь, улыбаясь, — что, "
+ "если бы вас послали вместо нашего милого Винценгероде,"
+ "вы бы взяли приступом согласие прусского короля."
+ "Вы так красноречивы. Вы дадите мне чаю?"
```





## Задача по упорядочиванию массива

Задача по упорядочиванию данных внутри массива посредством алгоритма сортировки методом выбора (или методом минимакса).

- \* Допустим имеется какая-то последовательность чисел 6 8 3 2 1 4 5 7.
- \* Задача - выбрать самый первый элемент и в оставшихся числах выбрать минимальное.
- \* После того, как найден минимальный элемент, нужно поменять его местами с первым элементом. То есть последовательность теперь будет 1 8 3 2 6 4 5 7.
- \* Далее выбирается второй рабочий элемент (8) и он меняется местами с минимальным числом из оставшихся, то есть последовательность будет 1 2 3 8 6 4 5 7. и т.д.
- \* Таким образом, все числа должны быть в последовательном порядке 1 2 3 4 5 6 7 8.

**Таким образом, общее решение выражено в трех действиях:**

- \* Найти позицию минимального элемента в неотсортированной части массива.
- \* Произвести обмен этого значения со значением первой неотсортированной позиции.
- \* Повторять пока есть не отсортированные элементы.

**Программа по сортировке одномерного массива:**

```
// Сортировка одномерного массива
int[] arr = { 1, 5, 4, 3, 2, 6, 7, 1, 1 };
void PrintArray(int[] array)
{
    int count = array.Length;
    for (int i = 0; i < count; i++)
    {
        Console.WriteLine($"{array[i]} ");
    }
    Console.WriteLine();
}
void SelectionSort(int[] array)
{
    for (int i = 0; i < array.Length - 1; i++)
    {
        int minPosition = i;
        for (int j = i + 1; j < array.Length; j++)
        {
            if (array[j] < array[minPosition]) minPosition = j;
        }
        int temporary = array[i];
        array[i] = array[minPosition];
        array[minPosition] = temporary;
    }
}

PrintArray(arr);
SelectionSort(arr);
PrintArray(arr);
```

## Лекция №4

### Двумерные массивы

#### Как задают двумерные массивы

- \* Указание типа данных (string) – по аналогии с одномерными массивами
- \* Запятая в квадратных скобках – указание на две размерности
- \* Наименование массива (table)
- \* Номер элемента. Первый нумерован [0, 0]

! Первая размерность – это строки (2), вторая – столбцы (3). Запятая (,) – указание размерности массива

#### Двумерные массивы

```
string[,] table = new string[2, 3];
```

```
int[,] matrix = new int[5, 8];
                ↓ строки
                ↑ столбцы
```



**Выводим на экран двумерную матрицу и одновременно заполняем её числами**

Код:

```

void PrintArray(int[,] matr)
{
    for (int i = 0; i < matr.GetLength(0); i++)
    {
        for (int j = 0; j < matr.GetLength(1); j++)
        {
            Console.WriteLine($"{matr[i, j]}");
        }
        Console.WriteLine();
    }
}

void FillArray(int[,] matr)
{
    for (int i = 0; i < matr.GetLength(0); i++)
    {
        for (int j = 0; j < matr.GetLength(1); j++)
        {
            matr[i, j] = new Random().Next(1, 10); // [1; 10]
        }
    }
}

int[,] matrix = new int[3, 4];
PrintArray(matrix);
FillArray(matrix);
Console.WriteLine();
PrintArray(matrix);

```

## Вывод:

```
[base] sergej.kamanecki@Sergejs-MacBook-Air Example013_ RecursionAlgorithm % dotnet run  
8 0 0 0  
8 0 0 0  
8 0 0 0  
8 0 0 0  
  
7 2 3 4  
3 4  
3 0 1 8  
[base] sergej.kamanecki@Sergejs-MacBook-Air Example013_ RecursionAlgorithm %
```

## Закрепляем информацию по работе с массивами

## Как и что описывать:

Тип данных → квадратные скобки с запятой  
внутри → имя массива, оператор присваивания  
→ тип данных → квадратные скобки с  
необходимым числом строк и столбцов внутри.

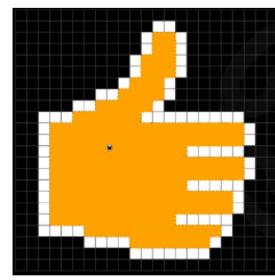
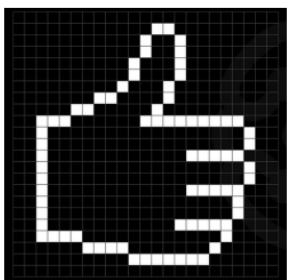
```
    ↓ строки matrix.GetLength(0)  
int[,] matrix = new int[5, 8];  
    ↑ столбцы matrix.GetLength(1)
```



## Представляем изображения в виде массива

Двумерные массивы полезны, в том числе, при работе с изображениями.

Вспомни задачу по информатике: картинка, пустые клетки – «0», заполненные – «1». Как закрасить руку цветом?



## Объясняем решение задачи

## Алгоритм решения:

- \* Договоримся, что у нас 23 строки и 25 столбцов, при этом «0» — закрашенный пиксель, а «1» — незакрашенный.
  - \* Алгоритм закрашивания: определяем точку внутри замкнутого контура, которую хотим закрасить.
  - \* Определяем принцип обхода внутренних точек: если попадаем в незакрашенную точку, закрашиваем её.  
  - \* Определяем порядок обхода: сначала идём вверх, потом влево, вниз и вправо. Когда остановились на точке, смотрим на точку, которая выше (отмечена розовым цветом). Если она не закрашена, красим её и от этой точки снова двигаемся вправо. Далее продолжаем двигаться вправо тем же способом.
  - \* Выясняем, что закрасили все точки с массивом «0» и пришли к точке «1», которая служит контуром, так что использовать (закрашивать) её нельзя.

- \* Возвращаемся к правилу обхода и двигаемся дальше.

## Решение с помощью кода

## Исходные данные:



## Программа:

```

void PrintImage(int[,] image)
{
    for (int i = 0; i < image.GetLength(0); i++)
    {
        for (int j = 0; j < image.GetLength(1); j++)
        {
            //Console.WriteLine($"{image[i, j]} ");
            if (image[i, j] == 0) Console.WriteLine("  ");
            else Console.WriteLine("#");
        }
        Console.WriteLine();
    }
}

void FillImage(int row, int col)
{
    if (pic[row, col] == 0)
    {
        pic[row, col] = 1;
        FillImage(row - 1, col);
        FillImage(row, col - 1);
        FillImage(row + 1, col);
        FillImage(row, col + 1);
    }
}

PrintImage(pic);
FillImage(13, 13);
PrintImage(pic);

```

## Вывод:

## Что такое рекурсия

! Рекурсия — это функция, которая вызывает сама себя.

Факториал  
5! = 5 \* 4 \* 3 \* 2 \* 1  
5 \* 4!  
4 \* 3!  
3 \* 2!  
2 \* 1!

### Пример классической задачи с рекурсией: вычисление факториала

! Факториал числа — произведение чисел от единицы до заданного числа. Обозначают знаком «!»

Факториал можно задать через факториал ( $5! = 5 * 4!$ )

```
int Factorial(int n)
{
    // 1! = 1
    // 0! = 1
    if (n == 1) return 1;
    else return n * Factorial(n - 1);
}

Console.WriteLine(Factorial(3)); // 1*2*3 = 6
```

Проблема в том, что если мы будем вычислять большие числа, то при выводе программы будут появляться отрицательные числа — из-за переполнения типа. Значение « $17!$ » не помещается в тип данных int.

### Как решить эту проблему?

- \* Вариант 1 — заменить int на double.
- \* Вариант 2 — использовать числа Фибоначчи.

### Демонстрация программы

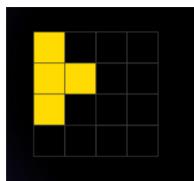
```
// f(1) = 1
// f(2) = 1
// f(n) = f(n-1) + f(n-2)

double Fibonacci(int n)
{
    if (n == 1 || n == 2) return 1;
    else return Fibonacci(n-1) + Fibonacci(n-2);
}

for (int i = 1; i < 50; i++)
{
    Console.WriteLine($"f({i}) = {Fibonacci(i)}");
}
```

## Примеры использования рекурсии

- \* Обход папки (директории) на компьютере
- \* Игра в тетрис
  - \* Основная мысль: поле для тетриса — двумерный массив. Смещение фигуры — это движение 0 и 1. Поворот фигуры — поворот матрицы.



### Подводим итоги

- \* Циклов много не бывает, и двумерные массивы тому подтверждение
- \* Массивов много не бывает
- \* Углубление в методы и практическая рекурсия

# Лекция №5

## Почему важно давать функциям понятные названия

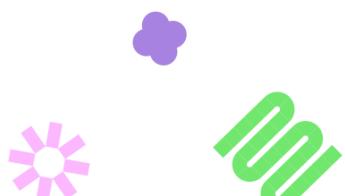
Пример. Для одной и той же задачи были сделаны методы со следующими названиями: Method, ShowNumbers, CreateArray, ShowNums, Ar и Numbers.

Спикер приводит несколько вариантов кода с решением задачи «Показать числа от -N до N».

### Как задают двумерные массивы

- \* Заменить имена переменных.
- \* Дать методам имена, отражающие суть функции.
- \* Именовать аргументы метода.

Доп. материал: [design guidelines](#).



## Общие правила написания хорошего кода

- ◆ Код чаще читают, чем пишут:  
**не пренебрегайте** понятностью и чистотой кода ради скорости.
- ◆ Страйтесь **не использовать сокращения**, кроме общепринятых в разрабатываемом продукте.
- ◆ Когда придумываете название для общедоступной единицы, страйтесь **не использовать имена**, потенциально или явно **конфликтующие со стандартными идентификаторами**.
- ◆ Пишите код только на латинице.

- ◆ Используйте имена с простым написанием. Их проще читать и набирать. **Избегайте** (в разумных пределах) использования слов с двойными буквами или сложным чередованием согласных.
- ◆ **Никогда** не используйте матерные (запрещённые) слова в коде (в том числе в комментариях).



## Про названия переменных:

- \* Обращайте внимание на звучание словосочетаний
- \* Не используйте неявные ссылки внутри названий переменных.
- \* **НЕ ИСПОЛЬЗУЙТЕ** знаки подчёркивания, дефисы и другие символы, которые не являются ни буквами, ни цифрами (подчёркивания допустимы в C#).

- \* **ИЗБЕГАЙТЕ** идентификаторов, совпадающих с ключевыми словами популярных языков программирования: string class = "1B". Дополнительные ссылки [\[1\]](#) [\[2\]](#).
- \* **Страйтесь** не использовать сокращения, если их можно трактовать по-разному.
- \* **НЕ ИСПОЛЬЗУЙТЕ** акронимы (аббревиатуры), которые не являются общепринятыми. По возможности вообще избегайте акронимов.

6

6

- \* **Используйте** универсальные имена платформы, не относящиеся к конкретному языку.

 ConvertToUnsignedChar(string value)  ConvertToByte(string value)

- \* **Используйте** общие, не привязанные к контексту имена, когда это нужно.

 ConvertToByte(string str)  ConvertToByte(string value)

- \* **Придерживайтесь** правил наименования каждой системной единицы (метода, переменной и др.) внутри выбранного языка программирования.

- \* Перед началом работы **договоритесь** с командой о нотациях, чтобы избежать разрозненности при сборке фрагментов кода, написанных разными людьми (Pascal / Camel).



Вот как можно упростить длинный код, сделать его понятнее и лаконичнее:

```
bool Metodi(int chislo)
{
    bool resultati = false;
    if (chislo % 2 == 0)
    {
        resultati = true;
    }
    else
    {
        resultati = false;
    }
    if (resultati == false)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

#### Про **bool**:

Не используйте проверки вида

b == false.

Используйте !b



Совет: когда вы написали код, попробуйте прочитать его на русском языке.

Проанализируйте, насколько код будет понятен читателю.

## Лекция №6



Важно не путать синтаксис языка и сам процесс программирования! Идея цикла — это тема программирования, а оператор for и знаки пунктуации — это синтаксис языка.

## Параметры красивого кода

### \* Гибкий:

- ❖ Функциональность модуля может быть переиспользована.  
Например, можно адаптировать метод заполнения массива разными данными.
- ❖ Но без избыточного обобщения!

### \* Расширяемый:

- ❖ Хорошо, если проектировка системы позволяет докинуть в неё новые модули, например, добавить в мессенджер коллективные чаты.

### \* Модульный:

- ❖ Каждая функция в коде выделена в отдельный модуль.
- ❖ Система отправки картинок в месседжере работает медленно. Модульная система позволяет вытащить модуль, который отвечает за отправку картинок, улучшить и заменить его без изменения других модулей.

### \* Поддерживаемый:

- ❖ Если выкате обновления что-то ломается, важно, чтобы программист, который не участвовал в разработке проекта, мог разобраться в коде и починить его.

### \* Документированный:

- ❖ Код снабжён комментариями.

## Практика 1. Предложения по улучшению кода

```
const Double пи = 3.1415;
int k__BackingField;
void set_MyProperty(int value)
{
    k__BackingField = value;
}
int get_MyProperty()
{
    return k__BackingField;
}
int MyProperty { get; set; }
```



- ❖ Именование кириллицей → в коде используем только латиницу.
- ❖ Double → double (в языке C# принято писать с маленькой буквы).
- ❖ Число Пи → есть встроенные константы.
- ❖ Имена переменной с нижним подчеркиванием → убрать \_.
- ❖ Имена методов с маленькой буквы → начинать с большой.
- ❖ MyProperty → заменить на имя, которое соответствует назначению переменной.
- ❖ Прежде всего проверьте, работает ли вообще код. 😊

## Практика 2: Предложения по улучшению кода

```
void DrawText(string text, int left, int top)
{
    Console.SetCursorPosition(left, top);
    Console.WriteLine(text);
}

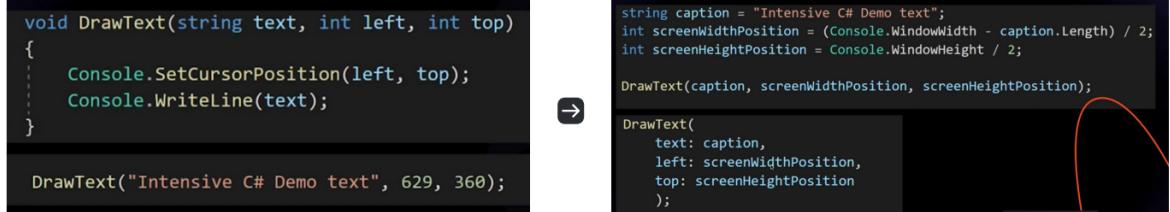
DrawText("Intensive C# Demo text", 629, 360);
```

→

```
string caption = "Intensive C# Demo text";
int screenWidthPosition = (Console.WindowWidth - caption.Length) / 2;
int screenHeightPosition = Console.WindowHeight / 2;

DrawText(caption, screenWidthPosition, screenHeightPosition);

DrawText(
    text: caption,
    left: screenWidthPosition,
    top: screenHeightPosition
);
```



- ❖ Именование методов и переменных — всё хорошо.
- ❖ Магические числа: 629 и 360. Стороннему читателю будет непонятно, откуда взялись эти числа без комментариев → вводить переменные.

## Практика 3. Какой вариант лучше?

```
string label = ""; // 111
string address = String.Empty; // 222
```

Второй вариант лучше, потому что.

- ❖ он понятен человеку, незнакомому с языком C#,
- ❖ нет «магической строки».

## Принципы разработки

1. **Don't repeat yourself** (с англ. «не повторяйтесь») → снижение повторения информации разного рода.
2. **YAGNI** — you aren't gonna need it (с англ. «вам это не понадобится») → отказ от избыточной функциональности: не делайте того, о чём вас не просят. Например, если поставлена задача вывести несколько случайных значений, а код написан для бесконечного вывода этих значений.
3. **KISS** — keep it simple, stupid (с англ. «делайте проще») → отказ от усложнений: простое и понятное описание логики методов.

## Демонстрация парсинга входной строки

Спикер решает задачу: есть координаты четырёх точек фигуры, их необходимо увеличить в два раза и показать пользователю конечный ответ.

! Важно: задание даётся не с целью повторить код, а для демонстрации основных паттернов написания кода.

```
using System.Linq;
string text = "(1.2) (2,3) (4,5) (6,7)"
    .Replace(".", ",")
    .Replace(")", ", ")
    ;
Console.WriteLine(text);

var data = text.Split(" ")
    .Select(item => item.Split(','))
    .Select(e => x: item.Parse(e[0]), y: int.Parse(e[1])))
    .ToArray();
for (int i = 0; i < data.Length; i++)
{
    Console.WriteLine(data[i].x * 10);
}
```

## Небольшой итог и рекомендации

- \* Договоритесь с командой о нотации: Hungarian, CamelCase, Pascal.
- \* Придерживайтесь Code Convention: принципов, оговоренных с командой в проекте.
- \* О комментариях: нужны, но по сути → без лишних «тут объявляем переменные».



/// — автоматический шаблон для комментариев, которые появляются при наведении курсора на переменную внутри кода.

```
/// <summary>
/// Вычисление координаты
/// </summary>
/// <param name="n">Это аргумент n</param>
/// <param name="height">Высоты экрана</param>
/// <returns>Новую позицию</returns>
```

```
height);
on(x, InBoxY(y[x] - l[x], height));
int Program.InBoxY(int n, int height)
Вычисление координаты
Возврат:
Новую позицию
```

- \* Проводите внутреннюю декомпозицию:
  - ❖ отдельных методов и больших структур (классов, например),
  - ❖ на как можно меньшие системные единицы.
- \* Чем больше методов, тем лучше (декомпозиция).
- \* Множество циклов внутри других циклов снижают читаемость.
- \* Пишите тестируемый код и сразу пишите тесты.
- \* Внимательно следите за входными данными, особенно за вводом пользователя.



В работе над кодом чаще спрашивайте себя, можно ли сделать лучше и проще.

# Лекция №7

## Что такое рекурсия и что важно при её описании

Рекурсивная функция — это функция, которая вызывает сама себя: то есть в теле метода (функции) есть вызов её же самой.

### Что важно при описании рекурсии

- \* Описать условие выхода: когда рекурсия закончит себя вызывать.
  - ◆ Код снабжён комментариями.
  - ◆ Неконтролируемая рекурсия тратит слишком много ресурсов, замедляет работу и создаёт впечатление, что программа зависла.

Вместо рекурсии можно использовать итеративный подход: классические ветвление и циклы.

Разберём примеры решения задач с помощью цикла и рекурсии.



## Задача 1. Собрать строку с числами от a до b, a ≤ b

### Решение с помощью цикла (итеративный подход)

- ◆ Цикл будет менять счётчик от значения a до «меньше или равно» b. Результирующее значение пишем в переменную result.

```
string NumbersFor(int a, int b)
{
    string result = String.Empty;
    for (int i = a; i <= b; i++)
    {
        result += $"{i} ";
    }
    return result;
}

Console.WriteLine(NumbersFor(1, 10)); // 1 2 3 4 5 6 7 8 9 10
```

### Решение на основе рекурсии

- ◆ Прописано условие окончания (else return String.Empty;).
- ◆ Если условие выполняется, собираем строку с текущим значением и вызываем новую копию метода с новыми значениями аргумента (значение переменной a увеличится на единицу).

```
string NumbersRec(int a, int b)
{
    if (a <= b) return $"{a} " + NumbersRec(a + 1, b);
    else return String.Empty;
}

Console.WriteLine(NumbersRec(1, 10)); // 1 2 3 4 5 6 7 8 9 10
```

? Как решить задачу так, чтобы стало на один вызов рекурсии меньше?  
Подсказка: подкрутите условие и перепишите ветку else.

## Задача 2. Найти сумму чисел от 1 до n

Решение с помощью цикла  
(итеративный подход):

```
int SumFor(int n)
{
    int result = 0;
    for (int i = 1; i <= n; i++) result += i;
    return result;
}

Console.WriteLine(SumFor(10)); // 55
```



Решение с помощью рекурсии:

◆ условие выхода: n = 0

```
int SumRec(int n)
{
    if (n == 0) return 0;
    else return n + SumRec(n - 1);
}

Console.WriteLine(SumRec(10)); // 55
```

? Попробуйте решить эту задачу  
с помощью математической  
формулы нахождения суммы от 1 до n.



## Задача 3. Найти факториал числа

Факториал — это произведение чисел от 1 до n.

Решение с помощью цикла  
(итеративный подход):

```
int FactorialFor(int n)
{
    int result = 1;
    for (int i = 1; i <= n; i++) result *= i;
    return result;
}

Console.WriteLine(FactorialFor(10)); // 3628800
```

Решение с помощью рекурсии:

```
int FactorialRec(int n)
{
    if (n == 1) return 1;
    else return n * FactorialRec(n - 1);
}

Console.WriteLine(FactorialRec(10)); // 3628800
```

## Задача 4. Вычислить $a^n$

! В следующих двух задачах мы не  
рассматриваем 0; для нас пока  $0^0 = 1$

a — основание степени; n — показатель степени.

**Решение с помощью цикла  
(итеративный подход):**

- perебор счтчика от 1 до n и домножение результата на текущее значение основания степени (a).

**Решение с помощью цикла  
(итеративный подход):**

- если показатель степени = 0, возвращаем 1; иначе запускаем рекурсивный подсчёт, где текущее значение а умножается на вызов функции, аргументы которой – основание степени и показатель на единицу меньше.

## Как сделать меньше строк для рекурсии в языке C#:

- использовать условный (тернарный) оператор и эти две строки:

```
if (n == 0) return 1;  
else return PowerRec(a, n - 1) * a;
```

Записать в одну строку вот так:

```
return n == 0 ? 1 : PowerRec(a, n - 1) * a;
```

Флешбеки из лекций по математике

- \* Формулы сокращенного возведения в степень:
    - ◆ Если показатель степени чётный, сразу умножаем число на само себя и уменьшаем показатель степени в два раза. Как следствие, – сокращение количества операций.

- ❖ Применение этих свойств оптимизирует код решения задачи;

```
int PowerRecMath(int a, int n)
{
    if (n == 0) return 1;
    else if (n % 2 == 0) return PowerRecMath(a * a, n / 2);
    else return PowerRecMath(a, n - 1) * a;
}

Console.WriteLine(PowerRecMath(2, 10)); // 1024
```

# Применение рекурсии в жизни

## 1. Обход директории

- ❖ Алгоритм обхода директории: мы хотим посмотреть все файлы внутри папки. Чтобы увидеть всё её содержимое, нужно заглянуть и во вложенные субдиректории. А это требует рекурсии.
- ❖ Для работы с директориями — класс DirectoryInfo.
- ❖ Код позволяет получить информацию об именах и расширениях файлов внутри папки, их классе и времени создания.

### Решение задачи с помощью рекурсии

```
void CatalogInfo(string path, string indent = "")  
{  
    DirectoryInfo catalogs = new DirectoryInfo(path);  
  
    foreach (var currentCatalog in catalogs.GetDirectories())  
    {  
        Console.WriteLine($"{indent}{currentCatalog.Name}");  
        CatalogInfo(currentCatalog.FullName, indent + " ");  
    }  
  
    foreach (var item in catalogs.GetFiles())  
    {  
        Console.WriteLine($"{indent}{item.Name}");  
    }  
}  
string path = @"/Users/sergejkamaneckij/Projects/HelloCode";  
CatalogInfo(path);
```

```
string path = //путь к папке  
DirectoryInfo di = new DirectoryInfo(path)  
System.Console.WriteLine(di.CreationTime);  
FileInfo[] fi = di.GetFiles();  
for (int i = 0; i < fi.Length; i++)  
{  
    System.Console.WriteLine(fi[i].Name);  
}
```

С помощью рекурсии получаем список файлов:

```
(base) sergejkamaneckij@Sergejs-MacBook-Air Example001_HelloConsole % dotnet run  
obj  
Debug  
net6.0  
ref  
    Example001_HelloConsole.dll  
    Example001_HelloConsole.gencache  
    Example001_HelloConsole.GlobalUsings.g.cs  
    Example001_HelloConsole.pdb  
    aphost  
    Example001_HelloConsole.AssemblyInfoInputs.cache  
    Example001_HelloConsole.csproj  
    Example001_HelloConsole.assets.cache  
    Example001_HelloConsole.csproj.CoreCompileInputs.cache  
    Example001_HelloConsole.csproj.FileListAbsolute.txt  
    Example001_HelloConsole.csproj.AssociationReference.cache  
    Example001_HelloConsole.csproj.AssemblyBuildEditorConfig.editorconfig  
.NETCoreApp,Version=v6.0.AssemblyAttributes.cs  
Example001_HelloConsole.AssemblyInfo.cs  
Example001_HelloConsole.csproj.nuget.dgspec.json  
Example001_HelloConsole.csproj.nuget.dgprops  
project.nuget.cache  
project.assets.json  
Example001_HelloConsole.csproj.nuget.g.targets  
bin  
Debug  
net6.0  
ref  
    Example001_HelloConsole.dll  
    Example001_HelloConsole.pdb  
    Example001_HelloConsole.deps.json  
    Example001_HelloConsole  
    Example001_HelloConsole.runtimeconfig.json  
.DS_Store  
.DS_Store  
Example001_HelloConsole.csproj
```

## 3. Решение арифметических выражений

Есть арифметическое выражение

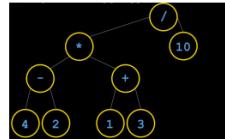
$$(4 - 2) * (1 + 3) / 10$$

- ❖ Арифметическое выражение состоит из арифметических выражений:

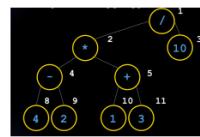
$$(4 - 2) * (1 + 3) / 10 \rightarrow 4 \ 2 \ 1 \ 3 \ 10 \ - \ + \ * \ /$$

— это всё арифметические выражения

- ❖ Разбиваем одно выражение на несколько более мелких и получаем бинарное дерево:



- ❖ Нумеруем операции по математическим правилам

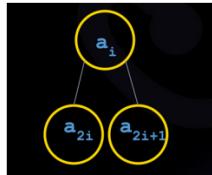


❗ Если бы вместо десяти (узел 3) была операция, внутри неё были бы операции под номерами 6 и 7. Так как в третьем узле только одно число, операций 6 и 7 вообще не будет в этом дереве. Дальше нумерацию начинаем с восьми.



Иерархия подчинённости: узлы, которые находятся выше, называются родительскими по отношению к узлам, которые ниже (дочерние узлы): - (4) и + (5) являются дочерними к \* (2), а /(1) – родительским по отношению к \*(2) и 10 (3).

- \* В общем виде дерево можно записать так:



В коде такое дерево можно представить как обычный одномерный массив:

```
string[] tree = { emp, "/", "*", "10", "+", "*", emp, emp, "4", "2", "1", "3" };
//          0   1   2   3   4   5   6   7   8   9   10  11
```

Логика обхода этого дерева: получить все значения узлов.

Код решения задачи:

```
string[] tree = { emp, "/", "*", "10", "+", "*", emp, emp, "4", "2", "1", "3" };
//          0   1   2   3   4   5   6   7   8   9   10  11
void InOrderTraversal(int pos = 1)
{
    if (pos < tree.Length)
    {
        int left = 2 * pos;
        int right = 2 * pos + 1;
        if (left < tree.Length && !String.IsNullOrEmpty(tree[left])) InOrderTraversal(left);
        Console.WriteLine(tree[pos]);
        if (right < tree.Length && !String.IsNullOrEmpty(tree[right])) InOrderTraversal(right);
    }
}
```

## Какие проблемы возникают при использовании рекурсии

- \* Низкая скорость вычислений.
  - ◆ Пример с кодом для чисел Фибоначчи от 10 до 40:
    - Пример с кодом для чисел Фибоначчи от 10 до 40:
    - с помощью рекурсии – за 20 сек.
- \* Переполнение стека (overflow).
  - ◆ Память и количество вызовов рекурсии не безграничны. Переполнение стека ведёт к ошибкам в программе.

## Во что превращают код компиляторы [1]

Рассмотрим на примере:

```
System.Console.WriteLine("Hello World");
```

Во что этот код превращается? С помощью сервиса [sharplab.io](https://sharplab.io) можно увидеть дополнительные строки кода, которые в шестой версии .Net писать уже не нужно.

## Во что превращают код компиляторы [2]

### Этапы компиляции:

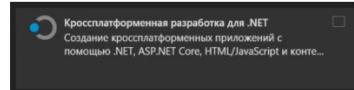
1. Ваш код на языке C# превращается в промежуточный код CIL (Command Intermediate Language).
2. Git-компилятор: превращает CIL-код в Byte-код: в бинарный файл, которым можно поделиться с товарищем.

NB Разобраться в теме поможет [учебник по классам в C#](#)

## Установка Большой Visual Studio Community

При установке [Visual Studio Community](#):

- \* Обязательно поставить галочку тут:



- \* Создать проект → Консольное приложение → Назвать файл → Выбрать платформу .NET 6.0

Раньше требовалось указывать класс. Внутри класса должны находиться методы.

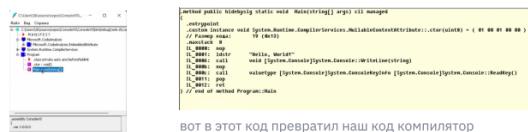
```
static public void Main ()  
или  
static public void Main (string[] args)
```

Получаем

```
class Program  
{  
    static public void Main (string[] args)  
    {  
        Console.WriteLine("Hello World");  
    }  
}
```

## Во что превращается код C# → промежуточный код CIL

- \* Программа IL DASM (в комплекте с Visual Code Studio) → открыть в ней файл .exe



BOT в этот код превратил наш код компилятор

! Когда вы пишете одну строчку кода, компилятор сам дописывает её за вас (например, internal static class или private static void)

Классы можно «паковать» в большие системные единицы — пространства имён (namespace).

```
1  using System;
2  using System.Linq;
3  using System.Text;
4  using System.Net.Http;
5
6  namespace ConsoleVS
7  {
8      class Program
9      {
10         static public void Main(string[] args)
11         {
12             ...
13         }
14     }
15 }
```

Библиотека ListMaster



```
6  class Program
7  {
8      static public void Main(string[] args)
9      {
10         if(args.Length >= 1)
11         {
12             if(args[0] == "-")
13                 Console.WriteLine($"Hello, {args[0].Replace("-", "")}!");
14         }
15     }
16 }
17 }
18 }
19 }
```

Классы можно «паковать» в большие системные единицы — пространства имён (namespace).

! public — открытый модификатор доступа

