



# Объектно-ориентированное программирование

---

2020

# Кто я?

---

Старший преподаватель кафедры 806

Дзюба Дмитрий Владимирович

[ddzuba@yandex.ru](mailto:ddzuba@yandex.ru)



Где найти  
материалы  
к курсу

---

<https://qrgo.page.link/1JEXf>

# Базовые требования к слушателям

---

1. Знание языка программирования C  
при изложении материала будем считать, что слушатель знает основные конструкции языка C, типы данных и правила написания программ
2. Знание операционной системы Microsoft Windows 7/8/10  
практические занятия будут проходить на компьютерах, работающих под управлением Microsoft Windows 7
3. Знание среды разработки Microsoft Visual Studio 2013/2015/2017 или Visual Studio Code + GCC/CLANG  
лабораторные работы должны делаться в Microsoft Visual Studio/VS Code, мы будем создавать консольные приложения с unmanaged кодом

# Отчетность по курсу рейтинг

5-бальная система	Рейтинговая система	Европейская система
5 - Отлично	90-100	A
4 – Хорошо	82-89	B
	75-81	C
3 - Удовлетворительно	67-74	D
	60-66	E
2 - Неудовлетворительно	Менее 60	F

## Балы даются:

1. Сделанная и сданная Лабораторная работа (8 шт) – от 5 до 15 баллов.
2. Зачет (два задания) по 15 баллов за задание (итого до 30).

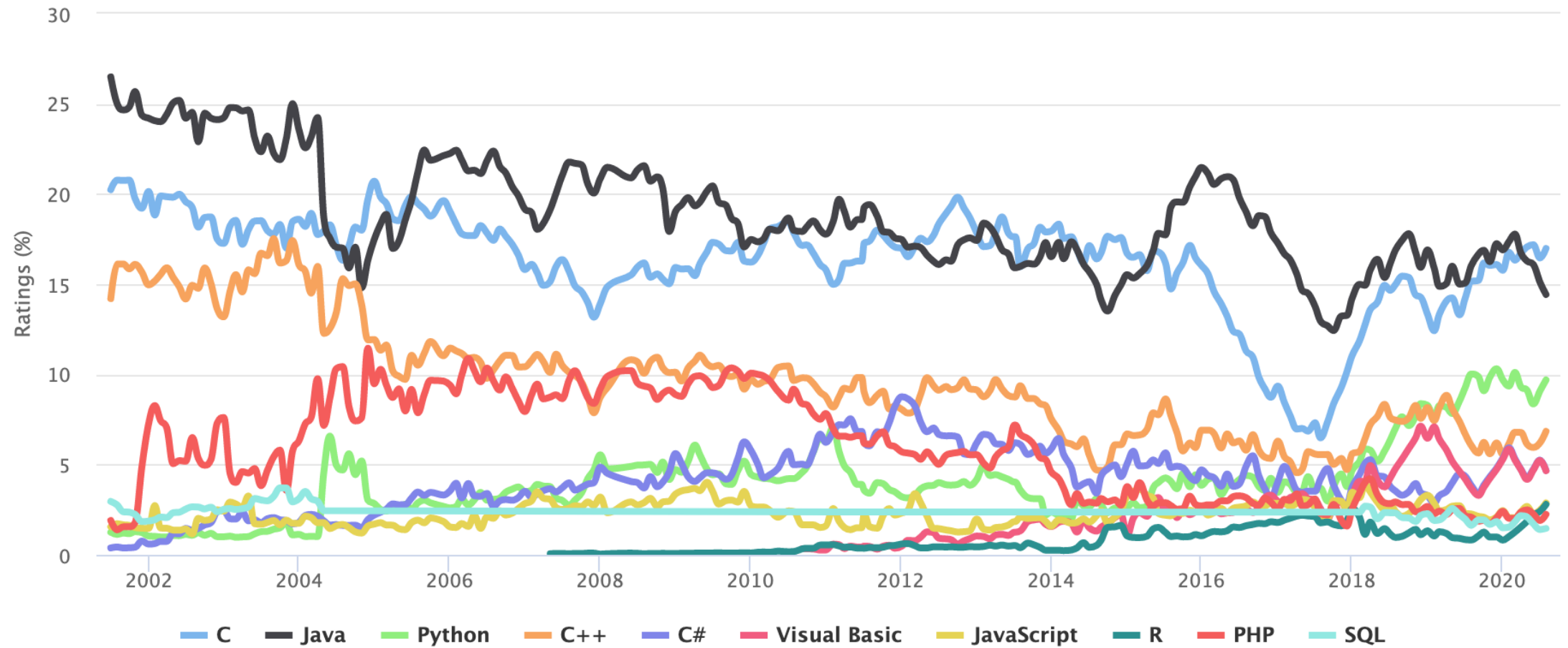
# Расписание занятий

---

1. Лекции проходят каждую неделю по вторникам с 10.45 до 12.15. В 314 ГАК.
2. Всего 16 занятий + консультация к зачету

# TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)





## WEB DEVELOPMENT

PHP C JAVASCRIPT C++ JAVA PYTHON RUBY



## GAME DEVELOPMENT

C# C C++ JAVA PYTHON RUBY



## MOBILE APP DEVELOPMENT

C# C++ JAVA



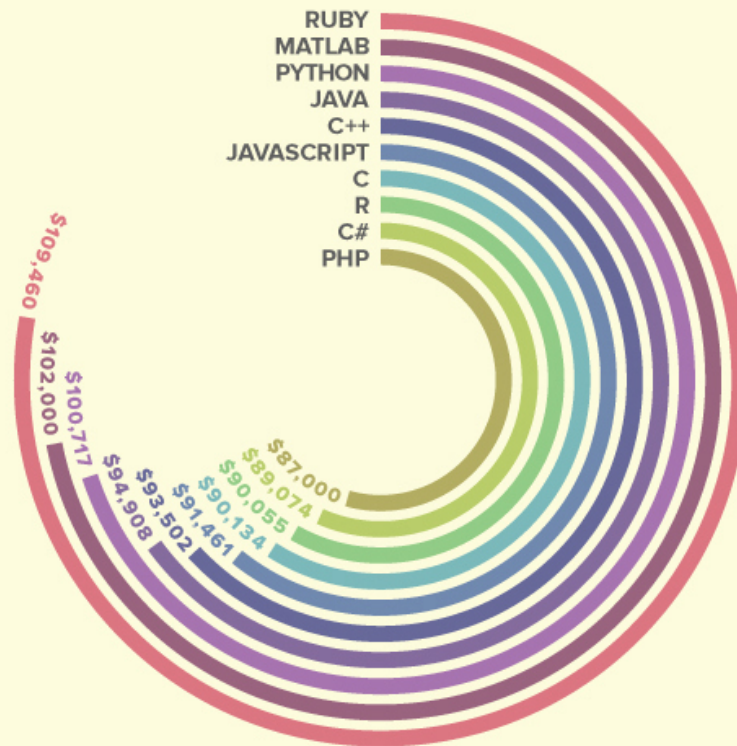
## DATA ANALYSIS

R PYTHON MATLAB



## EMBEDDED SYSTEM PROGRAMMING

C C++ PYTHON





# Среда разработки

---

Допускается использование следующих сред разработки/компиляторов:

- Microsoft Visual Studio 2017/19 для MS Windows 7/8.1/10
- X-Code (clang) для MacOS X 10.x
- gcc для Linux (например, Ubuntu).

Допускается использование других компиляторов C++ поддерживающих стандарт C++ 17 и выше.

# Общие понятия

---

ЛЕКЦИЯ №1

# Семейство языков С

---

## С

1972 , Dennis Ritchie @ Bell Labs.

Императивный язык программирования

## С++

1979 , Bjarne Stroustrup @ Bell Labs.

Императивный, объектно-ориентированный язык программирования

# Язык С

---

1. Компилируемый.
2. Императивный.
3. Ручное управление памятью.
4. Используется когда нужно написать программу, которая:
  - Эффективна по скорости работы.
  - Эффективна по потребляемой памяти.

# Ручное управление памятью в С

---

## Цели процесса

Позволить программе помечать области памяти как «занятые» полезной информацией.

Позволить программы помечать области памяти как «не занятые» по окончании работы. Что бы эти области могли использовать другие алгоритмы и программы.

## Что есть в С

В библиотеке `stdlib.h` есть функции `malloc` и `free`.

# Управление памятью: Куча (heap)

---

1. Куча – это область памяти, который может использовать программа.
2. Кучу можно сравнить с гигантским массивом.
3. Для работы с кучей используется специальный синтаксис указателей.
4. Вся программа может получить доступ к куче.

Addr.	Contents
:	:
0xbee	0xbeef
0xbf4	0xfed
:	:

# Пример работы с кучей в С

malloc.cpp

---

```
#include "stdlib.h" // работа с памятью
#include "stdio.h" // работа с вводом и выводом
#include "string.h" // работа со строками

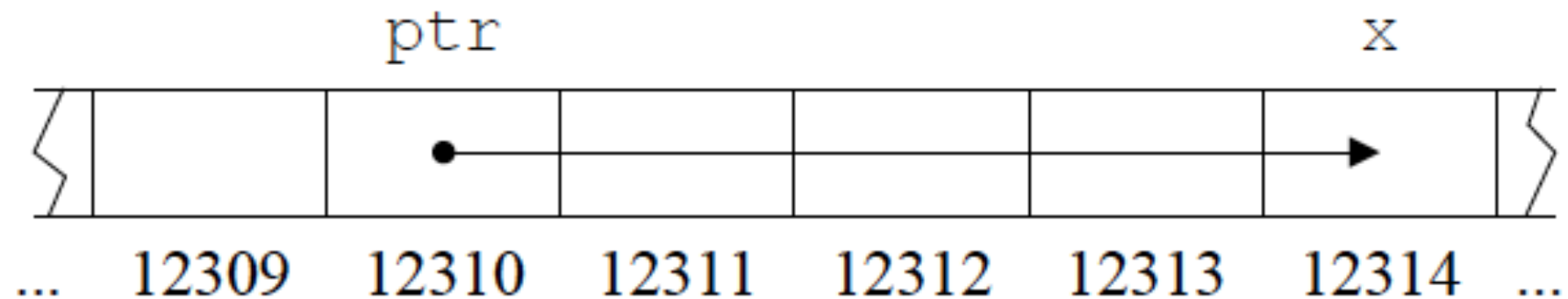
int main()
{
    // выделяем память
    char *pointer = (char*) malloc(sizeof(char)*100);
    // копируем в память данные
    strcpy(pointer, "Hello World!");
    // получаем данные из памяти
    printf("%s", pointer);
    // освобождаем указатель
    free(pointer);
    // ждем нажатия любой клавиши
    getchar();
    return 0;
}
```

# Указатель — это число

Указатель хранит адрес переменной!

```
int x = 5;
```

```
int * ptr = &x;
```





# Что считает функция?

square.cpp

---

```
#include <iostream>

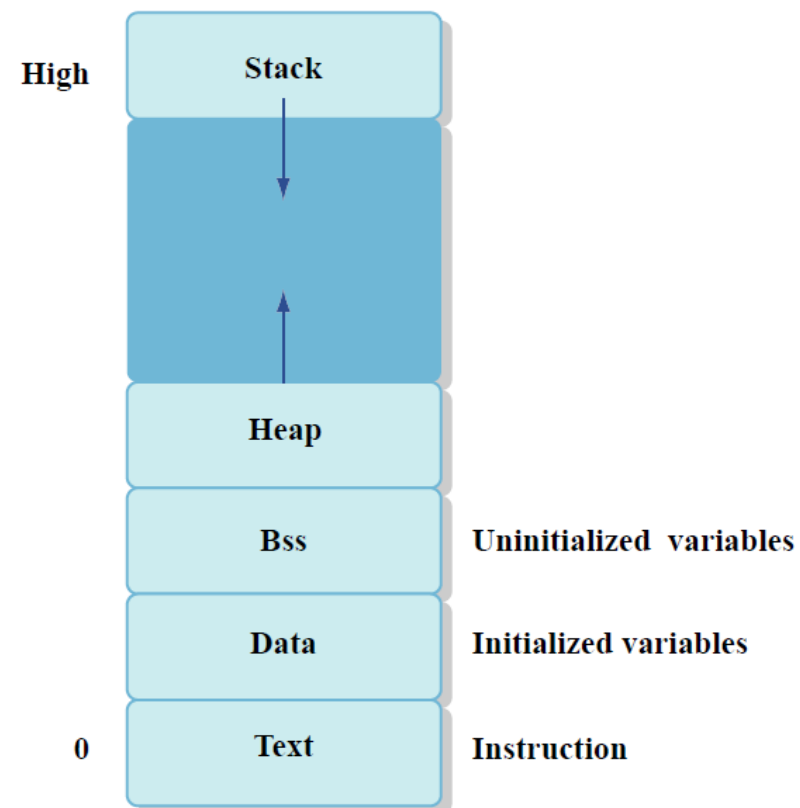
void squareByPtr ( int * numPtr ) {
    *numPtr = *numPtr * *numPtr ;
}

int main () {
    int x = 5;
    squareByPtr (& x);
    std::cout << x;
}
```

# Выделение памяти в стеке (stack)

Функции C размещаются в стеке:

1. Функции помещаются в стек, в момент вызова.
2. Функции удаляются из стека в момент когда вызывается return.
3. Функция может использовать любую память в пределах стека.



# Переменные «на стеке»

stackoverflow.cpp

---

```
1. #include "stdlib.h" // работа с памятью
2. #include "stdio.h" // работа с вводом и выводом
3. #include "string.h" // работа со строками
4. int a[10];
5. int *array_func(int val) {
6. // int a[10];
7. for (int i = 0; i < 10; i++) a[i] = val;
8. return a;
9. }
10. int main(int argc, char** argv) {
11. int *array = array_func(3);
12. for (int i = 0; i < 10; i++)          printf("Array is %d\n", *(array+i));
13. return 0;
14. }
```



# Основные понятия ООП

---

# Объект

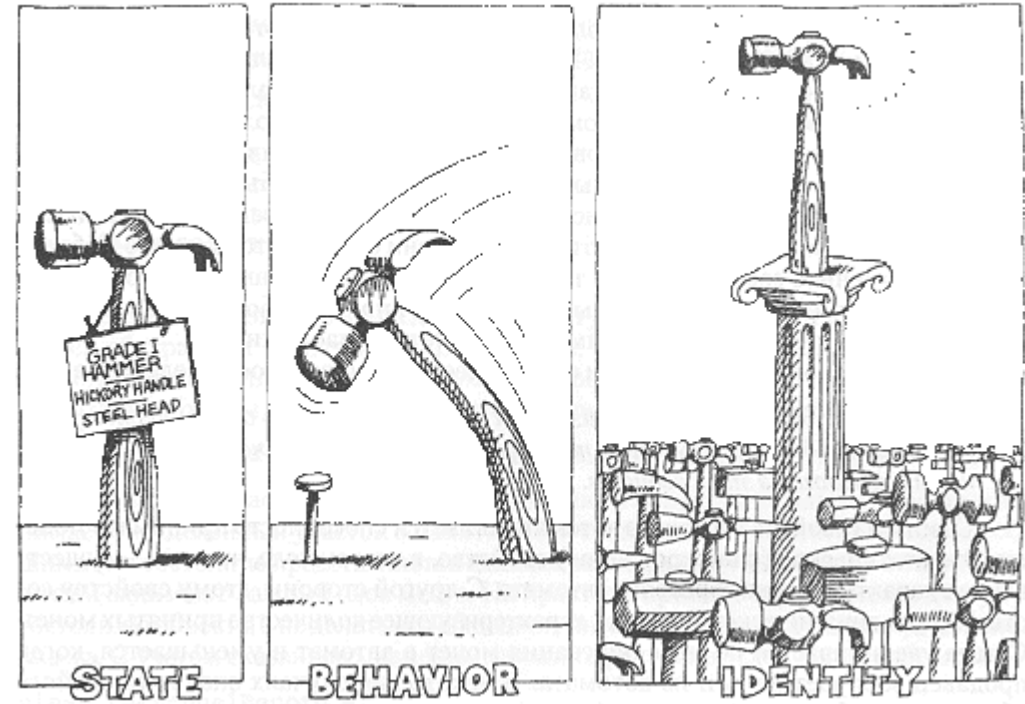
---

«Объект представляет собой конкретный опознаваемый предмет, единицу или сущность (реальную или абстрактную), имеющую четко определенное функциональное назначение в данной предметной области»

Smith, M. and Tockey, S. 1988. An Integrated Approach to Software Requirements Definition Using Objects. Seattle, WA: Boeing Commercial Airplane Support Division, p.132.

# Свойства объекта

1. **Состояние**  
в любой момент времени объект находится в каком-либо состоянии, которое можно измерить / сравнить / скопировать
2. **Поведение**  
объект может реагировать на внешние события либо меняя свое состояние, либо создавая новые события
3. **Идентификация**  
объект всегда можно отличить от другого объекта



# Класс

---

## 1. Определение.

Классом будем называть группу объектов, с общей структурой и поведением.

## 2. Смысл программы на С++ это описание классов!

## 3. Даже если нужен всего один объект – мы будем описывать класс.

# Очень простой класс объектов

first\_class.cpp

---

```
class MyClass  
{  
public:  
    int Number;  
    void doSomething();  
};
```

**class** – ключевое слово

**public** – область видимости атрибутов и методов класса

**int** Number – атрибут класса

**void** doSomething() - метод класса



# Как работать с вводом/выводом в C++?

<http://www.cplusplus.com/reference/iostream/>

---

Механизм для ввода-вывода в Си++ называется потоком . Название произошло от того, что информация вводится и выводится в виде потока байтов – символ за символом.

- Класс **istream** реализует поток ввода,
- Класс **ostream** – поток вывода.

Библиотека потоков ввода-вывода определяет три глобальных объекта: `cout`, `cin` и `cerr`.

- **cout** называется стандартным выводом,
- **cin** – стандартным вводом,
- **cerr** – стандартным потоком сообщений об ошибках.

**cout** и **cerr** выводят на терминал и принадлежат к классу **ostream**, **cin** имеет тип **istream** и вводит с терминала. Разница между **cout** и **cerr** существенна в **Unix** – они используют разные дескрипторы для вывода. В других системах они существуют больше для совместимости.

Вывод осуществляется с помощью операции `<<`, ввод с помощью операции `>>`.

```
int x; cin >> x; // ввод числа X
```



# Печатаем на экран

---

```
#include <iostream>
```

```
int main() {
```

```
    int a = 0;
```

```
    std::cin >> a;
```

```
    std::cout << "Значение переменной: " << a << std::endl;
```

```
    return 1;
```

```
}
```

# Пример по работе с потоками

stream.cpp

---

```
1. #include <cstdlib>
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. int main(int argc, char** argv) {
6.     std::string file_name;
7.     std::string file_text;
8.     std::cout << "Please enter file name:";
9.     std::cin >> file_name;
10.    std::ofstream out_file(file_name, std::ofstream::out);
11.    std::cout << "Please enter file text:";
12.    while (std::cin >> file_text) out_file << file_text << std::endl;
13.    out_file.close();
14.    std::cout << "Result:" << std::endl;
15.    std::ifstream in_file(file_name);
16.    while (in_file >> file_text) std::cout << file_text << std::endl;
17.    in_file.close();
18.    return 0;
19. }
```

# Класс на C++

second\_class.cpp

---

```
class SquareEquation {  
public: SquareEquation(double, double, double);  
        double FindX1();  
        double FindX2();  
private:  
        double a;  
        double b;  
        double c;  
};
```

```
#include "SquareEquation.h"  
  
#include <math.h>  
  
double SquareEquation::FindX1() {  
    return (-b-sqrt(b*b-4*a*c))/(2*a);  
};  
  
double SquareEquation::FindX2() {  
    return (-b-sqrt(b*b-4*a*c))/(2*a);  
};
```

# Конструктор

---

Если у класса есть конструктор, он вызывается всякий раз при создании объекта этого класса. Если у класса есть деструктор, он вызывается всякий раз, когда уничтожается объект этого класса.

Объект может создаваться как:

1. автоматический, который создается каждый раз, когда его описание встречается при выполнении программы, и уничтожается по выходе из блока, в котором он описан;
2. статический, который создается один раз при запуске программы и уничтожается при ее завершении;
3. объект в свободной памяти, который создается операцией `new` и уничтожается операцией `delete`;
4. объект-член, который создается в процессе создания другого класса или при создании массива, элементом которого он является.

# Жизненный цикл данных

Example07\_Life

---

```
class Life{
public:

    // Конструкторы
    Life() { std::cout << "I'm alive" << std::endl;}
    Life(const char* n) : Life() { name=n; std::cout<< "My name is " << name << std::endl;};

    // Деструкторы
    ~Life() { std::cout << "Oh no! I'm dead!" << std::endl;}
private:
    std::string name;
};
```

# Сколько раз вызовется конструктор?

---

```
class Integer {  
public:  
  int val;  
  Integer() {  
    val = 0;  
    cout << "default constructor" << endl;  
  }  
};  
  
int main() {  
  Integer arr[3];  
}
```

# Система сборки

---

- Вызов компилятора из командной строки
- Использование утилиты make с самостоятельным написанием MakeFile
- Использование утилиты CMake



# CMake

---

<https://cmake.org/cmake-tutorial/>

1. Не зависит от компилятора/платформы
2. Позволяет собирать выполняемые файлы, библиотеки, unit-тесты

# Простой пример

CMakeLists.txt

---

```
cmake_minimum_required (VERSION 3.2)
```

```
project (MyProject)
```

```
add_executable(main main.cpp)
```

```
set_target_properties(main PROPERTIES
```

- CXX\_STANDARD 14
- CXX\_STANDARD\_REQUIRED ON
- )

# Системы управления исходным кодом

---

1. Предоставляет возможности управления изменениями исходного кода (сохранение истории, возможность ее просмотра, откат к предыдущим версиям);
2. Позволяет работать с репозиториями
  - Централизованными (SVN, CVS)
  - Распределенными (Git)
3. Позволяет взаимодействовать командам разработчиков
4. Интегрироваться со средствами Continuous Delivery/Continuous Integration

# Git <https://git-scm.com/>

Распределенная система  
контроля исходным кодом с  
открытым кодом (open-source).



<https://habr.com/ru/company/playrix/blog/345732/>

# ОСНОВНЫЕ КОМАНДЫ

---

`git init`

`git clone`

`git add`

`git rm`

`git commit`

`git push`


`git pull`


<https://dzone.com/articles/top-20-git-commands-with-examples>




# Join GitHub

The best way to design, build, and ship software.

 **Step 1:**  
Set up your account

 **Step 2:**  
Choose your subscription

 **Step 3:**  
Tailor your experience

## Create your personal account

**Username \***

Username OOP4MAI is not available.  
OOP4MAI-spec, OOP4MAI-dev, or OOP4MAI133 are available.

**Email address \***

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

**Password \***

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

**You'll love GitHub**

**Unlimited** public repositories

**Unlimited** private repositories

✓

 Limitless collaboration

✓

 Frictionless development

✓


 Open source community

# Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner

Repository name \*

 OOP4MAI ▾

 / 

Great repository names are short and memorable. Need inspiration? How about **effective-octo-parakeet**?

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.



**Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾





Спасибо!

---

НА СЕГОДНЯ ВСЕ