

# 1 Клуб фанатов Manowar

1. Каждая секция - отдельная тема у Сорокина ([sorokin.github.io/cpp-course](http://sorokin.github.io/cpp-course)).
2. Для списков используйте `\begin{enumerate}` или `\begin{itemize}`
3. Очень помогает latex cheatsheet.
4. Первые пять билетов пропущены - там асм и прочая дичь. Пока что в приоритете билеты по плюсам.
5. Может попасться всё до 19 билета включительно.
6. Писать максимально подробно по каждому пункту - мало ли, что могут спросить.
7. ГачиБасс228.

## 2 Билет 6

### 2.1 Структуры

С помощью структур можно создавать пользовательские типы. По дефолту все поля структуры публичны, в то время как поля классов по дефолту приватны.

Доступ к полям структур осуществляется через оператор `'.'`, для указателей на структуры используется оператор `'->'`.

```
struct point {
    double x;
    double y;
};
...
point p = {11.4514, 810.931};
p.x = -19.19;
point * pp = new point;
pp->y = 4545;
```

Инициализировать структуры можно, прописав фигурные скобки.

Методы — функции, определяемые в полях структур (non C-style). Они реализованы с неявным параметром *this*, который является указателем на текущий экземпляр структуры. Объявление и определение методов:

```
struct point {
    // declaration
    void shift(double x, double y);
    double x;
    double y;
};
...
// definition
void point::shift(double x, double y) {...}
```

Определение структуры не генерирует никакого кода на ассемблере. Структуры существуют только на момент компиляции, они определяют то, как данные адресуются в памяти и располагаются, но после того, как код скомпилировался, никакой информации о структурах уже нет. Конструкторы — методы для инициализации структур.

```
point() {x = y= 0;}
```

```
    point(double x, double y) {
        this->x = x;
        this->y = y;
    }
...
// empty constructor call
point p1;
// calling constructor with 2 args
point p2 = point(1, 2);
```

Список инициализации позволяет проинициализировать поле до входа в тело конструктора. Инициализация полей в списке инициализации происходит в порядке объявления полей. То есть при написании  $y(0), x(y)$  произойдёт не то, что ожидалось.

```
point() : x(0), y(0) {}
point(double x, double y) : x(x), y(y) {}
```

В функциях можно определить значения по умолчанию.

```
point(double x = 0, double y = 0) : x(x), y(y) {}
...
point p1; // x = 0, y = 0
point p2(2); // x = 2, y = 0
point p3(3, 4); // x = 3, y = 4
```

Конструктор одного параметра задаёт неявное пользовательское преобразование. Запретить это можно при помощи ключевого слова *explicit*.

```
explicit point(double x = 0, double y = 0) : x(x), y(y) {}
...
point p = 2; // error
```

Конструктор по умолчанию генерируется компилятором, если он пользовательски не задан.

```
struct segment {
    segment(point p1, point p2) : p1(p1), p2(p2) {}
    point p1;
    point p2;
};
...
segment s; // error
```

Деструктор — метод, вызывающийся при удалении структуры и по умолчанию генерируется компилятором. У него может быть только 0 аргументов, и он всегда один на одну структуру. Объявляется как `<struct name>`.

Конструкторы и деструктор позволяют чётко определять время жизни структуры. Обращение к некоторым значениям до вызова конструктора или после вызова деструктора приводит к undefined behavior.