

Лекция 7.

Классы

```
struct Point {  
    int x;  
    int y;  
};
```

Пусть есть структура Rect :

```
struct BadRect {  
    int width = 0;  
    int height = 0;  
    int area = 0;  
};  
  
int GetRectAreaSlow(const BadRect& rect) {  
    // ...  
}
```

Структуры – это всего лишь набор переменных, которые никак не связаны и независимы, но если нам потребуется какая-то связь, то надо использовать классы.

```
class Rect {  
public:  
    int SetWidth(int new_width) {  
        width_ = new_width;  
        UpdateArea();  
    }  
  
    int Area() const { // const не позволяет менять приватные переменные  
        return area_;  
    }  
  
private:  
    void UpdateArea() {  
        area_ = width_ * height_;  
    }  
  
private:  
    int width_ = 0;  
    int height_ = 0;  
    int area_ = 0;  
};
```

Классы практически не отличаются от Структур. Единственное отличие – люди договорились, что структура считается набором полей, а классы еще имеют связи между полями.

Модификаторы доступа.

- **private** : поля и функции можно менять/читать/вызывать только изнутри класса. Другими словами, к ним нет доступа снаружи.
- **public** : можно менять извне; методы в этом разделе можно вызывать снаружи.
- **protected** : ...

Константные поля;

```
public:
```

В константной функции нельзя менять приватные переменные.

```
int foo() const {  
  
}
```

Если в разделе `private` объявить переменную как `mutable int area`, то ее можно менять даже в `const` функции.

```
// код про кэш; пример для использования mutable переменных.
```

Таким образом мы можем обращаться к `RectId` как к типу данных, равному по свойствам `uint64_t`

```
class RectDataBase {  
public:  
    using RectId = uint64_t;  
  
    Rect GetRect(RectId) {  
        // ...  
    }  
}
```

Инициализация класса

Конструктор вызывается автоматически, сразу при создании объекта. Код просто не скомпилируется, пока в конструктор не передадут все важные переменные.

```
class Rect {  
public:  
    Rect(int width, int height) { // конструктор  
        width_ = width;  
        height_ = height;  
        UpdateArea();  
    }  
  
private:  
    void UpdateArea() {  
        area_ = width_ * height_;  
    }  
  
private:  
    int width_ = 0;  
    int height_ = 0;  
    int area_ = 0;  
}
```

```
Rect(int width, int height) // другой вид конструктора  
    : width_{width}  
    , height_{height}  
    {}
```

Можно пользоваться также и перегрузкой функций и тогда компилятор сам будет выбирать нужный конструктор.

Если есть функция, принимает в себя объект класса, то вызов функции может выглядеть неочевидно:

```
void foo(Rect r){
    ...
}

int main() {
    foo(123);
}
```

Тогда в конструкторе можно указать ключевое слово `explicit`, и тогда мы будем обязаны в функции указать иначе

```
int main() {
    foo(Rect{123})
}
```

Деструкторы

Как понятно из названия, вызывается в конце существования объекта.

```
class Noisy {
public:
    Noisy(int idx) {
        // что выполнится при создании объекта
    }
    ~Noisy() {
        // что выполнится в конце существования объекта
    }
}
```

Объект удаляется, как и переменные, в конце блока с `{...}`.