

# Базовый C++

## Структуры и классы. Часть I

Кухтичев Антон



13 октября 2025 года

# Напоминание отметиться на портале

и оставить отзыв  
после лекции



# Содержание занятия

- Квиз #4
- Пользовательские типы
- Классы и структуры
- Модификаторы доступа
- RAII (Resource Acquire Is Initialization)
- Константные методы
- Операторы

# КВИЗ #4

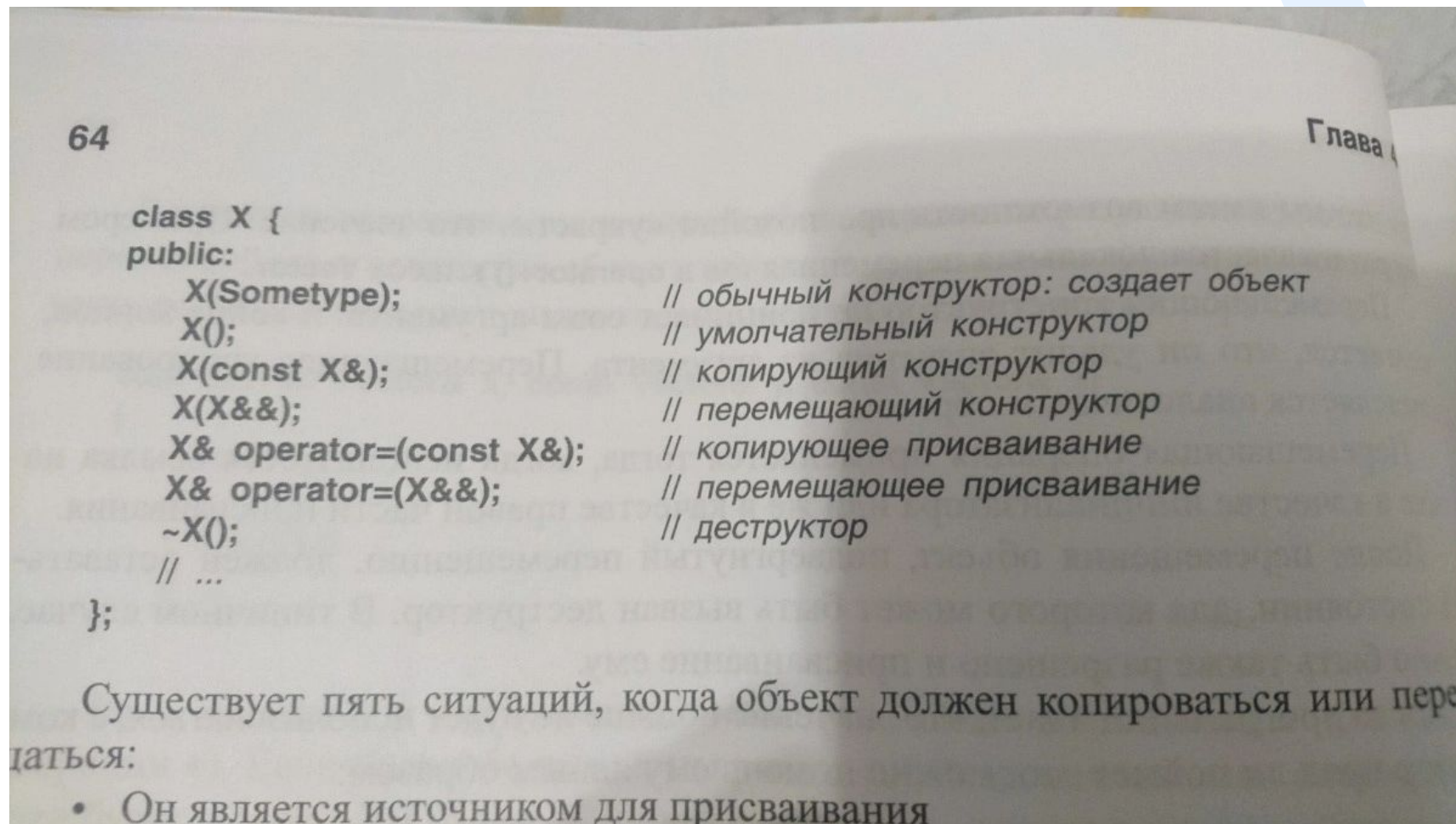


# Минутка юмора



# Язык программирования C++. Стандарт C++11.

## Краткий курс



# Пользовательские ТИПЫ



# Пользовательские типы

- Перечисления

```
enum class Exception {};
```

- Классы

```
class ClassName {};
```

```
struct ClassName {};
```

- Объединения

```
union VariantName {};
```





# Классы и структуры



# Понятие класса

- Объектно-ориентированное программирование построено на понятие класса;
- Объявление класса начинается с ключевого слова `class`;
- По умолчанию члены класса являются закрытыми (`private`-членами);
- Классы и структуры это родственные типы;
- Объект — сущность в адресном пространстве компьютера, появляющаяся при создании класса;
- По определению структура есть класс, все члены которого по умолчанию являются открытыми;

# Конструктор


- Конструктор — это функция, которая вызывается при создании объекта;
- Конструктор вызывается автоматически при создании объекта при помощи `new` (но не при помощи `malloc`!);
- Если конструктор не написан явно, C++ гарантирует, что будет создан конструктор по умолчанию;
- Не возвращает тип;

# Деструктор

- Деструктор — это функция, которая вызывается при разрушении объекта;
- Если деструктор не написан явно, C++ гарантирует, что будет создан деструктор по умолчанию;
- Не возвращает тип;

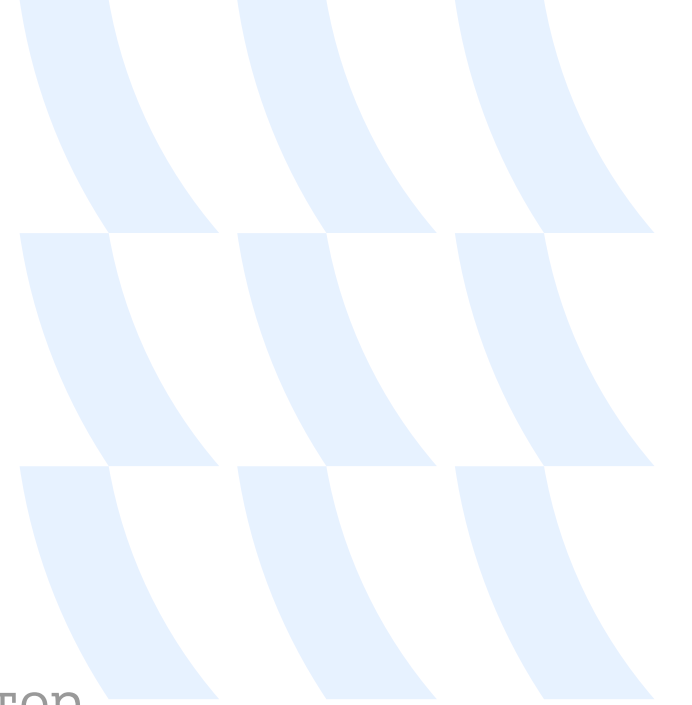
# Модификатора доступа

```
class A
{
public:
    int x_; // доступно всем;
protected:
    int y_; // доступно не только лишь всем;
            // только внутри класса и наследникам;
private:
    int z_; // мало кому доступно; доступно только внутри класса;
};
```



# Пример класса

```
class Dachshund
{
public:
    Dachshund(uint8_t age) { age_ = age; } // конструктор
    ~Dachshund() {} // деструктор
private:
    uint8_t age_ = 0; // закрытый член класса
};
```



# Специальные функции-члены

- В C++98 включает четыре такие функции:
  - конструктор по умолчанию
  - деструктор
  - копирующий конструктор
  - оператор копирующего присваивания
- Эти функции создаются, только если они необходимы, т.е. если некоторый код использует их без их явного объявления в классе;
- Конструктор по умолчанию генерируется только в том случае, если в классе не объявлен ни один конструктор.



1. Скотт Мейерс. Эффективное использование C++. Правило 5. Какие функции C++ создаёт и вызывает молча.

# Специальные функции-члены

- В C++11 приняты два новых игрока:
  - перемещающий конструктор;
  - оператор перемещающего присваивания;

Подробнее о них на следующей лекции.



1. Скотт Мейерс. Эффективный и современный C++. Пункт 3.11  
Генерация специальных функций-членов



# Специальные функции-члены

Если поведение сгенерированных компилятором функций вас устраивает (т.е. почленное копирование нестатических членов-данных), то можно сказать компилятору это:

```
class Seed {  
public:  
    ...  
    ~Seed(); // пользовательский деструктор  
    ...  
    // Поведение копирующего конструктора по умолчанию правильное!  
    Seed(const Seed&) = default ;  
};
```



1. Скотт Мейерс. Эффективный и современный C++. Пункт 3.11  
Генерация специальных функций-членов

# Какие методы генерирует компилятор при наличии различных входных данных

Если явно объявить

Вы получите результат

	Ничего	Деструктор	Конструктор копирования	Присваивание копии	Конструктор перемещения	Присваивание перемещения
Деструктор ~Foo()	✓	✓	✓	✓	✓	✓
Конструктор копирования Foo(const Foo&)	✓	✓	✓	✓		
Присваивание копии Foo& operator=(const Foo &)	✓	✓	✓	✓		
Конструктор перемещения Foo(Foo &&)	✓		Вместо переноса используется копирование		✓	
Присваивание переноса Foo& operator=(Foo &&)	✓					✓



- 1. Джош Лоспинозо. С++ для профи. Глава 4. Жизненный цикл объекта. Методы, генерируемые компилятором. (стр. 192).

# Ссылка на себя

- Каждая (нестатическая) функция-член знает, для какого объекта она вызвана, и **может явно** на него ссылаться при помощи `this`;
- `this` является указателем на объект, для которого вызвана функция;

```
struct A
{
    int x_ = 0;
    void foo([A *this]) {
        this->x_ += 10;
        x_ += 10;
    }
};
```

# RAII (Resource Acquire Is Initialization)

- Захват ресурса есть инициализация.
- В конструкторе объект получает доступ к какому либо ресурсу (например, открывается файл), а при вызове деструктура этот ресурс освобождается (закрывается файл).
- Можно использовать не только для управления ресурсами;
- Класс инкапсулирует владение (захват и освобождение) некоторого ресурса;

# RAII (Resource Acquire Is Initialization)

```
struct Profiler
{
    Profiler() {
        // получаем текущее время
    }
    ~Profiler() {
        // сохраняем время между вызовами конструктора и деструктора
    }
};

void someFunction()
{
    Profiler prof;
    // ...
}
```



# Как узнать имя функции внутри профайлера?

```
struct Profiler
{
    Profiler(const std::string &func_name) {
        func_name_ = func_name;
    }
    std::string func_name_;
};

void someFunction()
{
    Profiler prof(__func__);
    // ...
}
```

# Как узнать имя функции внутри профайлера C++20?

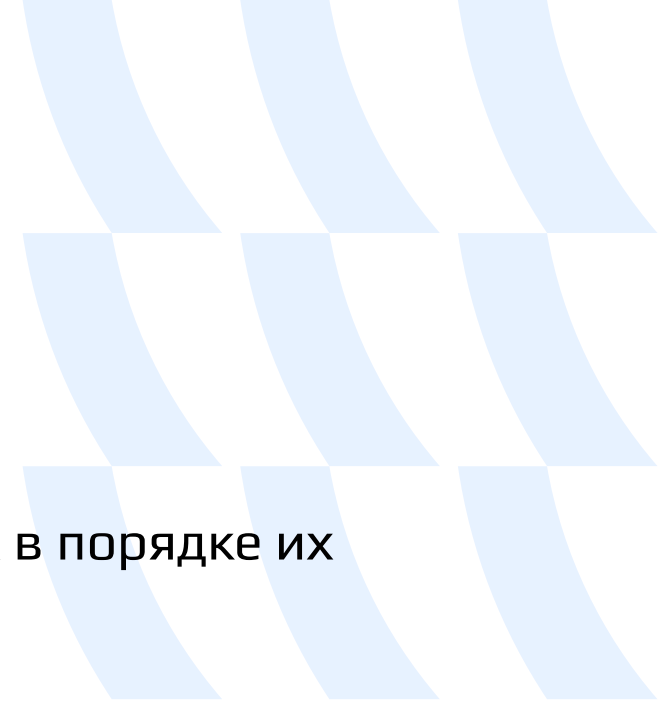
```
#include <source_location>
```

```
struct Profiler
{
    Profiler(const std::source_location &location =
              std::source_location::current()) {
        location_ = location;
    }
    std::source_location location_;
};
```

```
void someFunction()
{
    Profiler prof();
    // ...
}
```

# Конструирование объекта

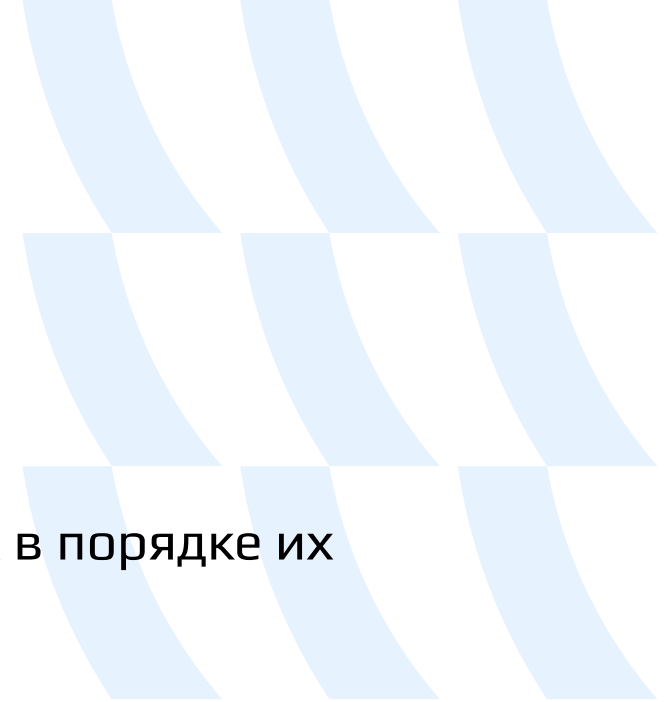
1. Выделяется память под объект;
2. Если есть базовые классы, то конструирование начинается с них в порядке их очередности в списке наследования;
3. Инициализируются поля класса в том порядке, в котором они объявлены в классе;
4. Происходит вызов конструктора.





# Уничтожение объекта

1. Выделяется память под объект;
2. Если есть базовые классы, то конструирование начинается с них в порядке их очередности в списке наследования;
3. Инициализируются поля класса в том порядке, в котором они объявлены в классе;
4. Происходит вызов конструктора.



# Списки инициализаторов членов

1. Все инициализации члена выполняются перед телом конструктора;
  - а. Обеспечивает правильность всех элементов перед выполнением конструктора;
2. Нужно упорядочить инициализаторы членов в том же порядке, в котором они указаны в определении класса, поскольку их конструкторы будут вызываться в этом же порядке;

# Константные методы



# Константные методы

- Любые методы кроме конструктора и деструктора могут быть константными.
- Метод, который гарантирует, что не будет изменять объект или вызывать неконстантные методы класса (поскольку они могут изменить объект).
- Константный метод можно вызывать как для константного, так и для неконстантного объекта, в то время как неконстантный метод можно вызвать только для объекта, не являющегося константой;
- **Рекомендация:** делайте все ваши методы, которые не изменяют данные объекта класса, константными.

# mutable

- Позволяет изменять члены класса внутри `const`-методов;



# Операторы



# Операторы

- `bool operator==(const T& other) const`
- `bool operator!=(const T& other) const`
- `T operator+(const T& other) const`
- `T operator-() const`
- `T& operator++() // ++x`
- `T operator++(int) // x++`
- `const T& operator[](size_t i) const`
- `std::strong_ordering operator<=>(const T&) constC++20`
- Также есть операторы `new`, `delete` и `,` (запятая)
  - `void* operator new (size_t)`
  - `void operator delete (void *)`



1. Скотт Мейерс. Эффективное использование С++. Правило 51: Придерживайтесь принятых соглашений при написании `new` и `delete`.

# Домашнее задание





## Домашнее задание #4 (1)

Нужно написать класс-матрицу, тип элементов `int32_t`. В конструкторе задается количество столбцов и количество строк.

Поддерживаются операции:

- получить количество строк(`rows`)/столбцов(`columns`);
- получить конкретный элемент;
- умножить на число (`*=`);
- сравнение на равенство/неравенство.

В случае ошибки выхода за границы бросать исключение:

```
throw std::out_of_range("")
```

## Домашнее задание #4 (2)

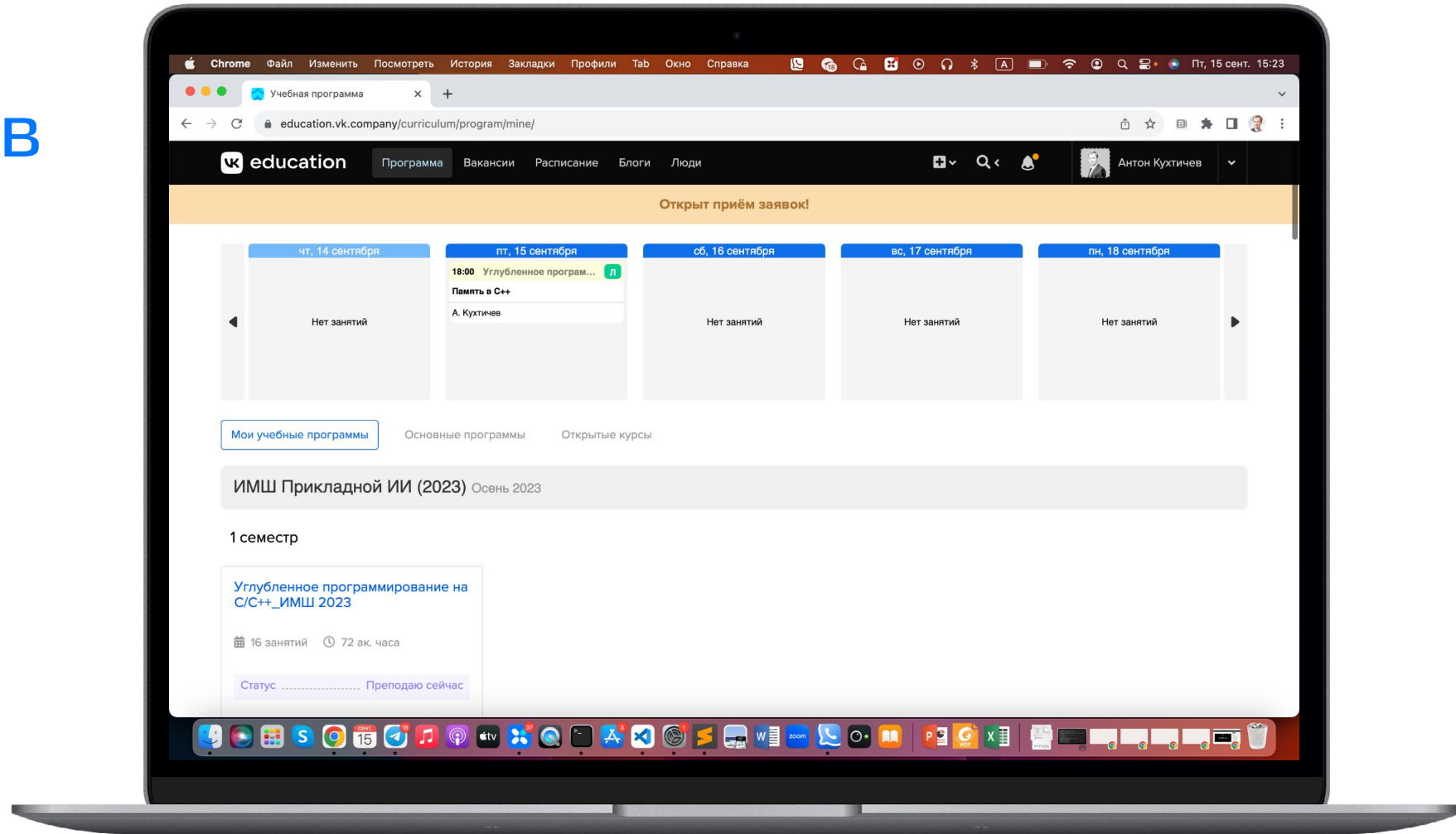
Чтобы реализовать семантику `[] []` понадобится прокси-класс. Оператор матрицы возвращает другой класс, в котором тоже используется оператор `[]` и уже этот класс возвращает значение.

# Полезная литература в помощь

- Джош Ласпинозо «С++ для профи»
- Скотт Мейерс «Эффективный и современный С++»
- Бьерн Страуструп «Язык программирования С++»

# Напоминание оставить отзыв

Это правда важно





Спасибо  
за внимание!