

# Структуры и классы

## Эпизод II

# Атака полиморфизма

Антон Кухтичев



# Напоминание отметиться на портале

и оставить отзыв  
после лекции



# Содержание занятия

- Наследование
- Перегрузка методов
- Виртуальные функции

# Перегрузка методов

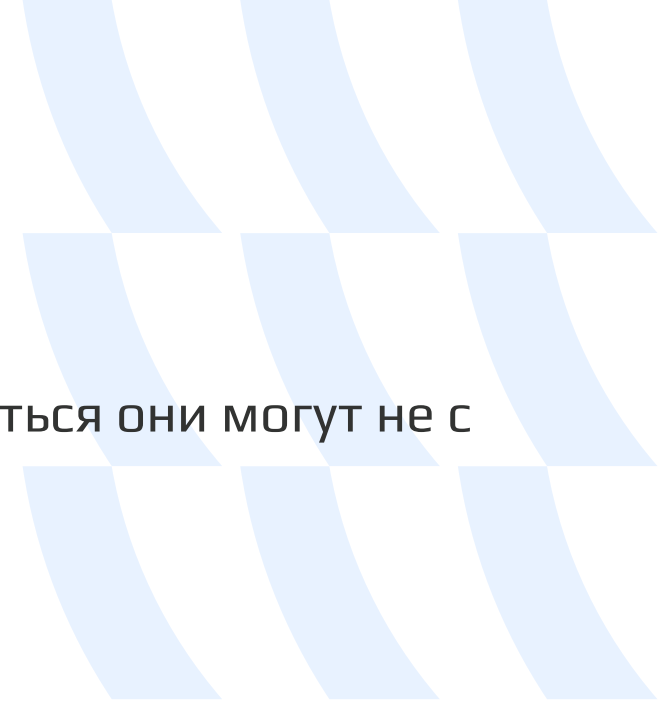


# Перегрузка методов

- Методы классов - это просто функции, в которые неявно передается указатель на сам класс;
- Конструкторы - это тоже функции и их тоже можно перегружать.
- Деструкторы - тоже функции, но перегружать нельзя.

# Параметры по умолчанию

- Пропусков в параметрах по умолчанию быть не должно, начинаться они могут не с первого аргумента, но заканчиваться должны на последнем.



# Наследование



# Наследование

- Возможность порождать класс на основе другого с сохранением всех свойств класса-предка.
- Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником, дочерним или производным классом.
- Наследование моделирует отношение «является».
- Требуется для создания иерархичности – свойства реального мира.



# Приведение вверх и вниз по иерархии

- Приведение вверх (к базовому классу) всегда безопасно;
- Приведение вниз может быть опасным;

```
struct A {};  
struct B : public A {};  
struct C : public A {};
```

```
B* b = new B();  
A* a = b;  
C* c = a; // Ошибка компиляции  
C* c = static_cast<C*>(b); // Ошибка компиляции  
C* c = static_cast<C*>(a); // !!!
```

# Приведение вверх и вниз по иерархии

- Приведение вверх (к базовому классу) всегда безопасно;
- Приведение вниз может быть опасным;

```
struct A {};  
struct B : public A {};  
struct C : public A {};
```

```
B* b = new B();  
A* a = b;  
C* c = a; // Ошибка компиляции  
C* c = static_cast<C*>(b); // Ошибка компиляции  
C* c = static_cast<C*>(a); // !!!
```

# Множественное наследование

- Сначала выделяется память под базовый класс
- Затем – под наследуемый
- Подобъекты являются полноценными объектами!

```
class Wolf
{
};
class Dog
{
};
class Husky : public Wolf, public Dog
{
};
```



# Представление в памяти при наследовании



# Инструменты для исследования

- В целях повышения быстродействия данные в памяти должны быть выровнены, то есть размещены определенным образом;

- Предпочтительное выравнивание можно узнать:

```
std::cout << alignof(char) << std::endl;    // 1
```

```
std::cout << alignof(double) << std::endl; // 8
```

- `sizeof(T)` - размер типа в байтах
- `offsetof(T, M)` - смещение поля M от начала типа T

# Инструменты для исследования

```
struct S
{
    char m1;
    double m2;
};
```

```
sizeof(char) == 1
sizeof(double) == 8
sizeof(S) == 16
offsetof(S, m1) == 0
offsetof(S, m2) == 8
```

```
[          char          ][          double          ]
[c][.][.][.][.][.][.][.][d][d][d][d][d][d][d][d]
```

# Инструменты для исследования

```
#pragma pack(push, 1)
class S
{
public:
    char m1;
    double m2;
};
#pragma pack(pop)
```

```
offsetof(S, m1) == 0
offsetof(S, m2) == 1
sizeof(S) == 9
```

# Простые типы (POD, Plain old data)

1. Скалярные типы (`bool`, числа, указатели, перечисления (`enum`), `nullptr_t`)
2. `class` или `struct` которые:
  - a. Имеют только **тривиальные** (сгенерированные компилятором) конструктор, деструктор, конструктор копирования;
  - b. Нет виртуальных функций и базового класса;
  - c. Все нестатические поля с модификатором доступа `public`;
  - d. Не содержит статических полей не POD типа.
  - e. В C++20 это понятие изменилось



1. Разбор понятий: trivial type, standard layout, POD  
<https://habr.com/ru/articles/532972/>



# Простые типы (POD, Plain old data)<sup>C++20</sup>

1. Класс со стандартным выравниваем (standard layout) это класс, который
  - a. не имеет нестатических элементов данных типа класса с нестандартным выравниванием,
  - b. не имеет виртуальных функций и виртуальных базовых классов,
  - c. имеет одинаковый контроль доступа для всех нестатических элементов данных,
  - d. не имеет базовых классов с нестандартным выравниваем,
  - e. ещё кое-что (см. ссылку)
2. То есть то, что C может понять.



1. [https://en.cppreference.com/w/cpp/language/classes#Standard-layout\\_class](https://en.cppreference.com/w/cpp/language/classes#Standard-layout_class)

# Простые типы (POD, Plain old data)

```
class NotPOD
{
public:
    NotPOD(int x)
    {
    }
};
```

```
class NotPOD
    : public Base
{
};
```

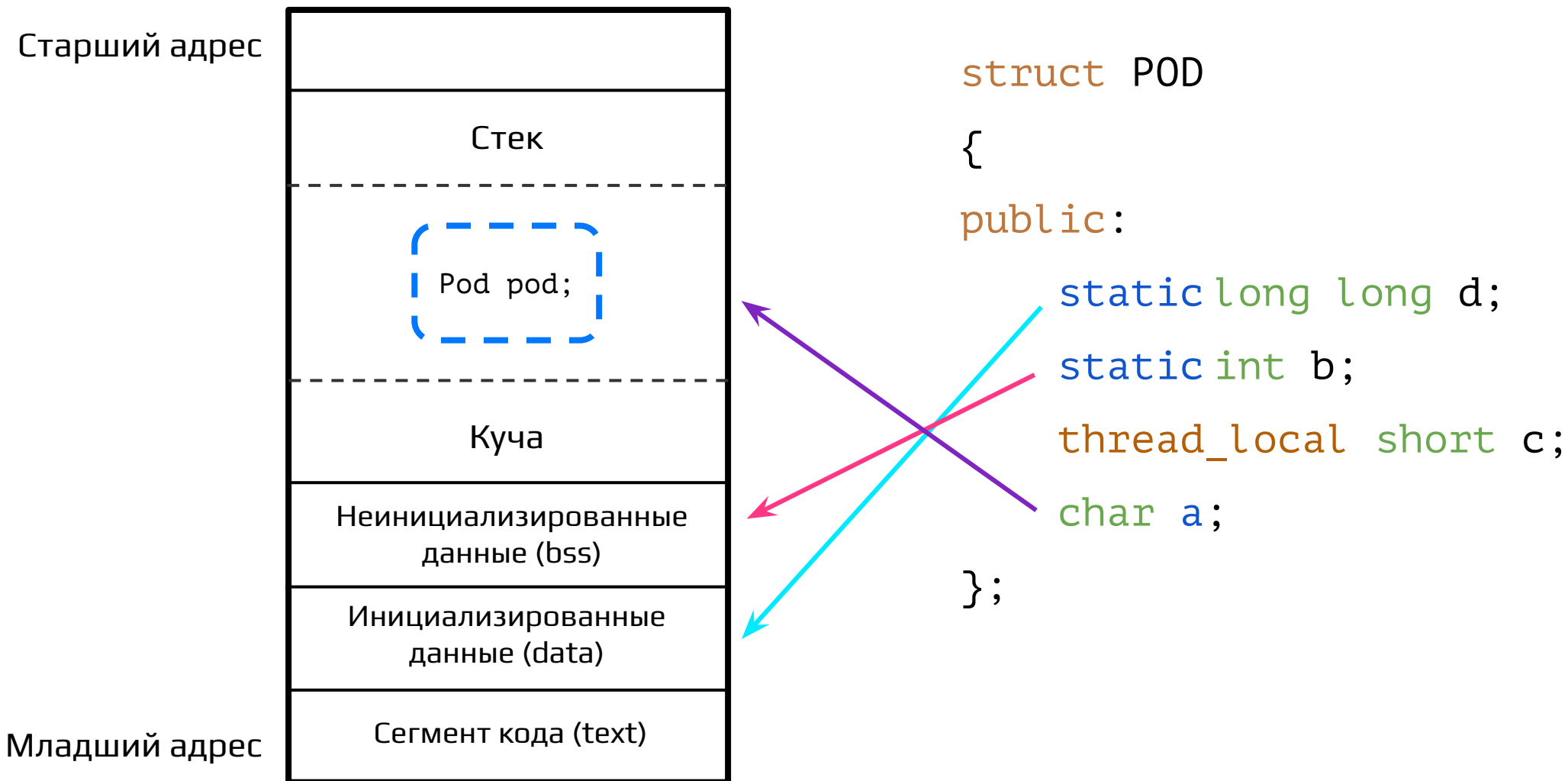
```
class NotPOD
{
    virtual void f()
    {
    }
};
```

```
class NotPOD
{
    int x;
};
```

# Простые типы (POD, Plain old data)

```
class POD
{
public:
    POD_ m1;
    int m2;
    static double m3;
private:
    void f() {}
};
```

# Data layout: static members

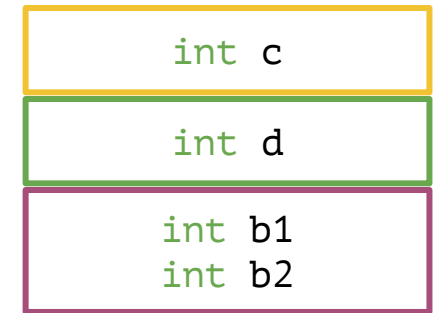


# Наследование

- Память классов родителей идёт в начале в последовательности указанной после ":";
- Приведение к базовому классу не требует действий;
- Приведение ко второму базовому классу, уже требует прибавить смещение

**Общее правило:** последовательность размещения наследуемых классов строится обходом дерева наследования DFS (Depth First, в глубину).

```
struct C
{
    int c;
};
struct D
{
    int d;
}
struct B : C, D
{
    int b1;
    int b2;
};
```



# Code time



- Разберём наследование;
- Выравнивание;
- Мотивация для виртуальных функций;

# Виртуальные функции



# Виртуальные функции

- Решают проблему, связанную с полем типа, предоставляя возможность программисту объявить в базовом классе функции, которые можно заместить в каждом производном классе.
- Производный класс, которые не нуждается в собственной версии виртуальной функции, не обязан её реализовывать;
- Функция из производного класса с тем же именем и с тем же набором типов аргументов, что и виртуальная функция в базовом классе, *замещает* (override) виртуальную функцию из базового класса;
- Тип, имеющий виртуальные функции, называется полиморфным типом.



1. Скотт Мейерс. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ. Правило 36. Никогда не переопределяйте наследуемые неvirtуальные функции.

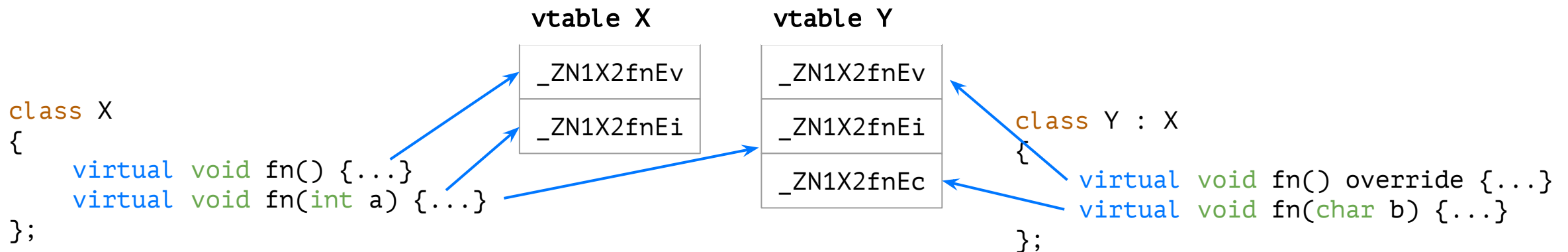


# Таблица виртуальных функций (1)

1. Если какая-либо функция класса объявлена как виртуальная, создается `vtable`, которая хранит адреса виртуальных функций этого класса;
2. Для всех таких классов компилятор добавляет скрытую переменную `vptr`, которая указывает на `vtable`;
3. Если виртуальная функция не переопределена в производном классе, `vtable` производного класса хранит адрес функции в родительском классе;

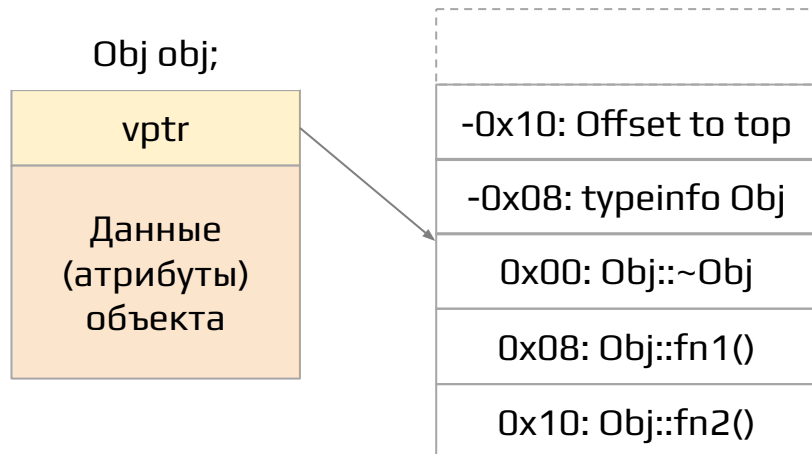
## Таблица виртуальных функций (2)

1. Если в унаследованном классе есть функция с такой же сигнатурой, то заменяется соответствующий слот в таблице этого класса;
2. Если такой сигнатуры нет - выделяем новый слот;
3. Вызов виртуальной функции идёт через указатель в этой таблице.



# vtable

1. В начале каждого объекта с виртуальными функциями есть указатель `vptr` на `vtable`
2. `vptr` указывает на элемент с индексом 2 (от начала);
3. `vptr[0]` содержит указатель на самую первую виртуальную функцию;
4. Виртуальные функции раскладываются в `vptr[n]` в порядке объявления;
5. `vptr[-1]` указывает на структуру `typeinfo` объекта;
6. `vptr[-2]` содержит смещение до начала объекта;
7. После `vptr` идут данные класса, в порядке объявления;
8. Неявно созданные деструкторы и т.п. располагаются в конце.



# Виртуальный деструктор

- Когда объект производного класса уничтожается через указатель на базовый класс с неvirtуальным деструктором, то результат не определен;
- Во время исполнения это обычно приводит к тому, что часть объекта, принадлежащая производному классу, никогда не будет уничтожена
- **Правило:** Объявляйте виртуальный деструктор! При удалении объектов производных классов будет происходить именно то, что нужно.



1. Скотт Мейерс. Эффективный использование C++. Правило 7. Объявляйте деструкторы виртуальными в полиморфном базовом классе.

# Абстрактные классы

- Класс с одной или несколькими чисто виртуальными функциями называется абстрактным классом;
- Абстрактный класс можно использовать только как интерфейс и в качестве базы для других классов;

```
class A
{
public:
    virtual void foo() = 0;
}
```

# Code time



- Дорабатываем пример для вывода на устройство;
- Напишем класс длинной арифметики с некоторыми операторами;

# Домашнее задание



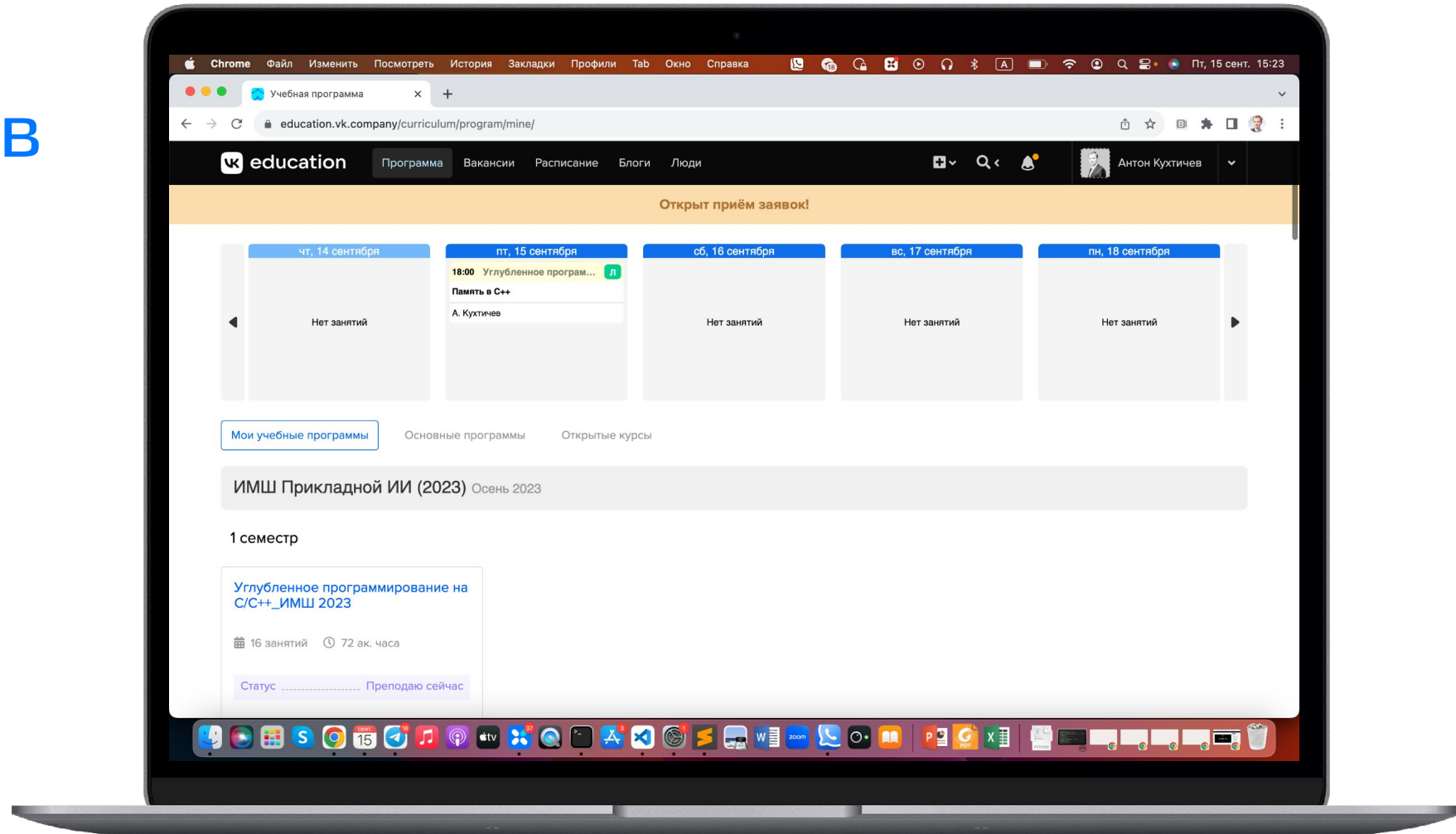
# Полезная литература в помощь

- 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!