Углублённое программирование на C++

Структуры и классы. Часть II

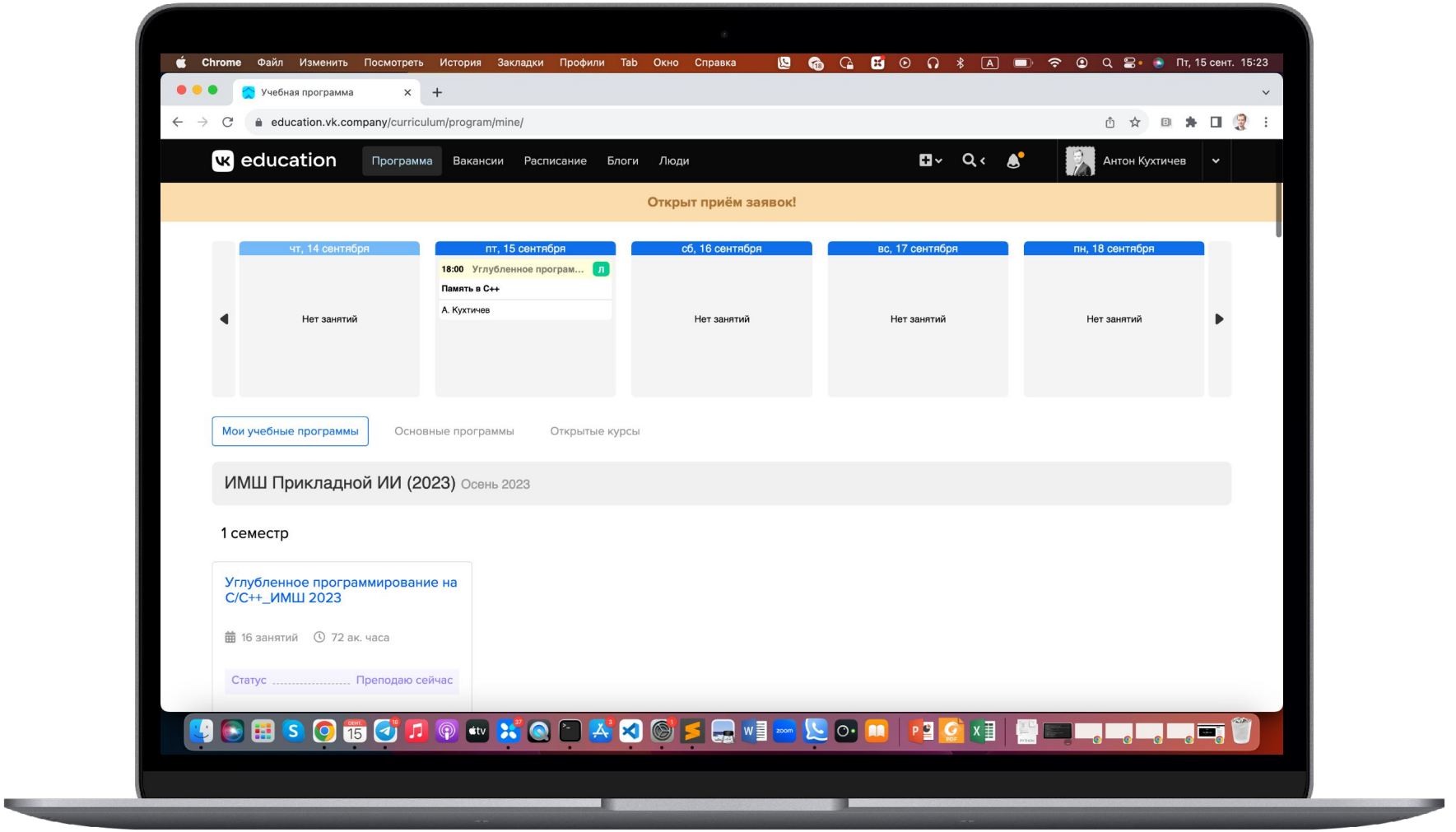
Кухтичев Антон

education

16 октября 2025 года

Напоминание отметиться на портале

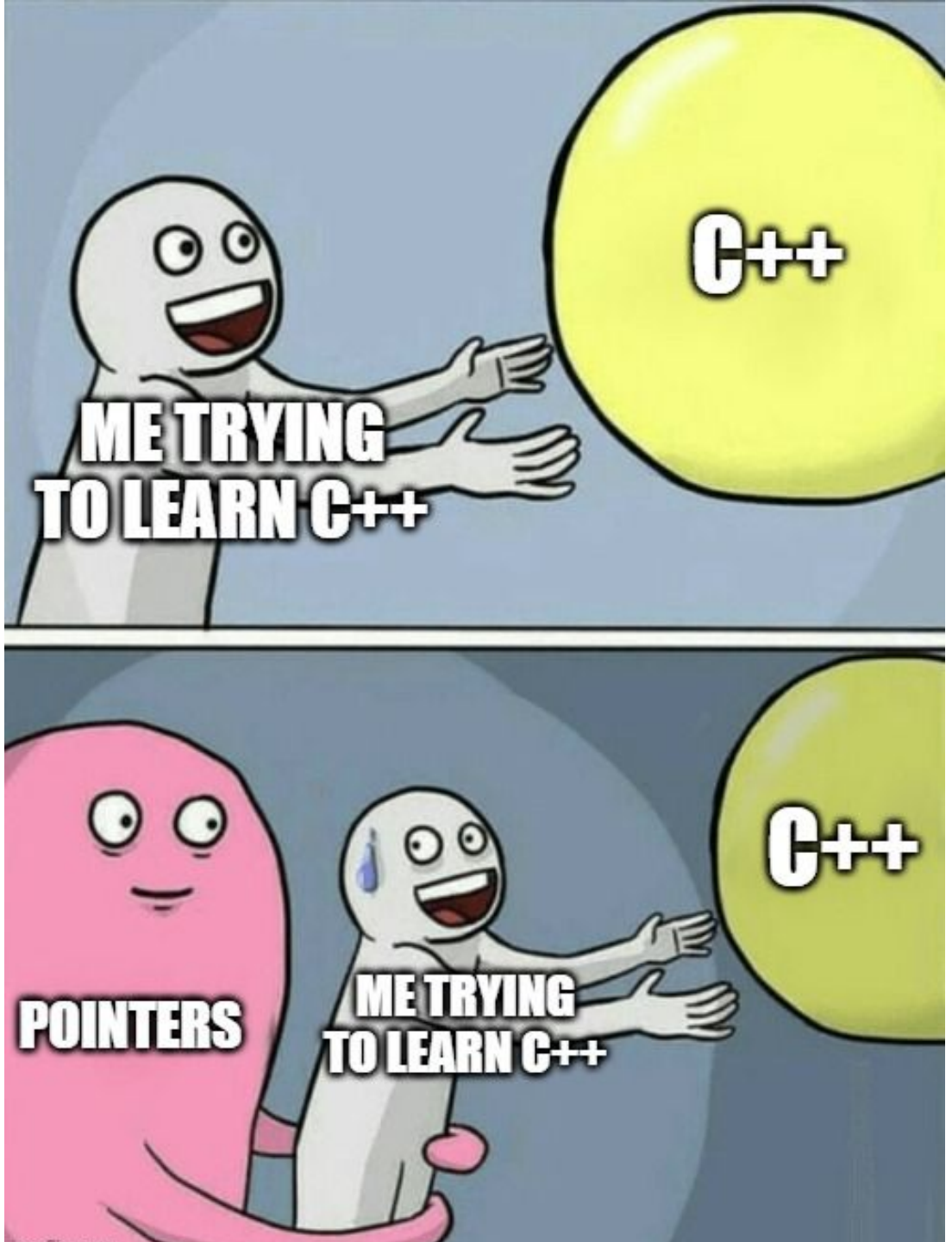
и оставить отзыв после лекции



Содержание занятия

- Наследование
- Перегрузка методов
- Виртуальные функции
- Операторы

Мем недели



Упражнение

```
class Example {
public:
    int* ptr;
    Example() : ptr(new int) {}
    ~Example() { delete ptr; }
};

Example* obj = new Example();
obj->~Example();
delete obj;
```

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); // !!!
```

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/
2. Евгений Ерохин — Back Deep to Basics: Наследование и виртуальность в C++ (в двух частях)
   Часть I: https://vk.com/video-77278886_456239842
   Часть II: https://vk.com/video-77278886_456239885

Простые типы (POD, Plain old data)$^{C++20}$

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

Простые типы (POD, Plain old data)

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

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

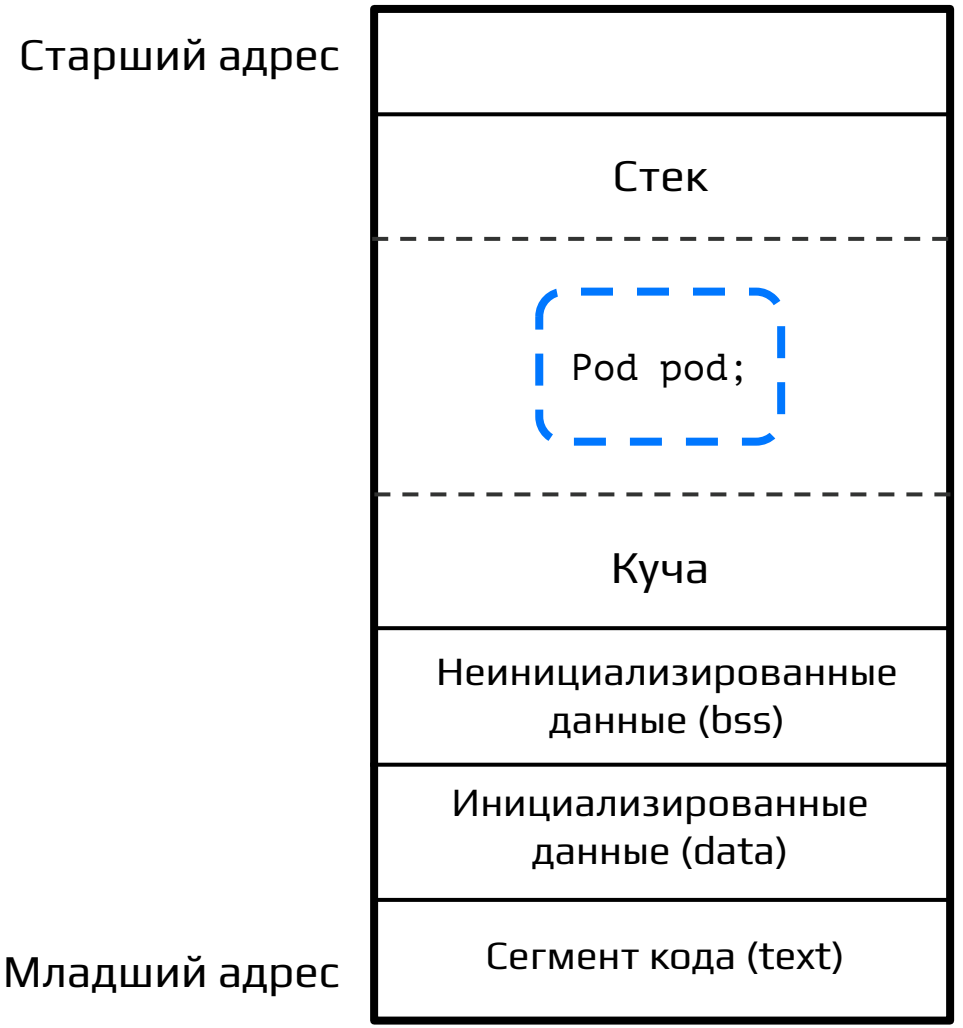
```
class NotPOD
    : public Base
{
};
```

```
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



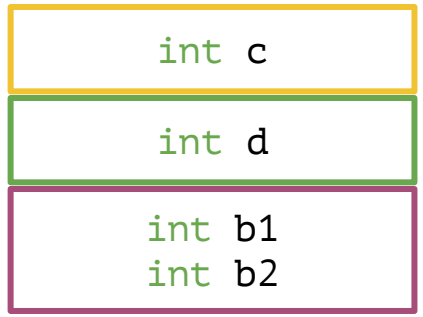

```
struct POD
{
public:
    static long long d;
    static int b;
    thread_local short c;
    char a;
};
```

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.

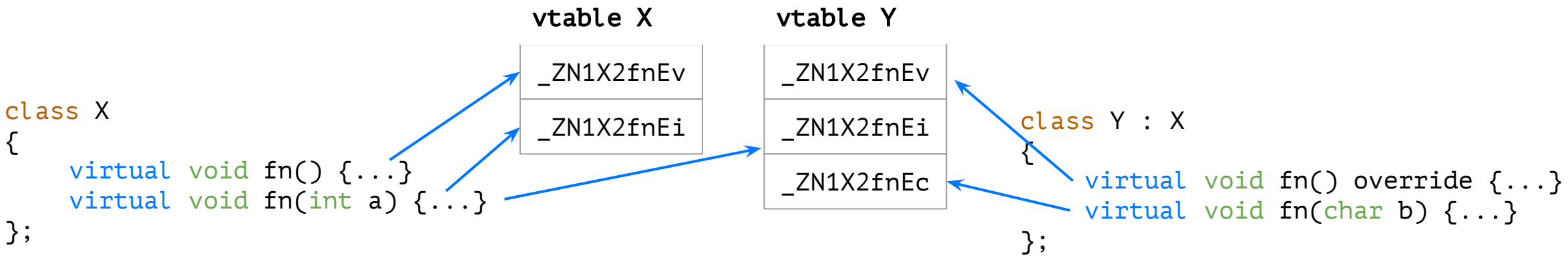
1. Скотт Мейерс. Эффективное использование C++. 55 верных советов улучшить структуру и код ваших программ. Правило 36. Никогда не переопределяйте наследуемые невиртуальные функции.

Таблица виртуальных функций (1)

1. Если какая-либо функция класса объявлена как виртуальная, создастся `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;

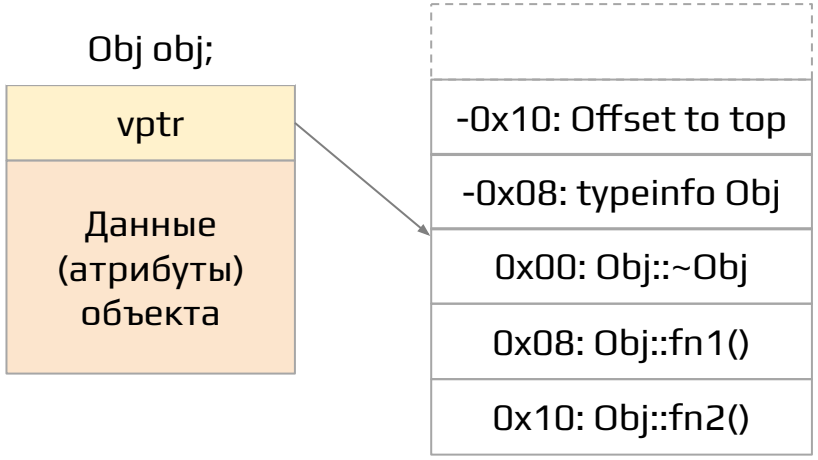
Таблица виртуальных функций (2)

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 невиртуальным 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.

Идентификацию типов времени выполнения (RTTI)

- Идентификацию типов времени выполнения обеспечивают два оператора:
  - `typeid`, возвращающий фактический тип заданного выражения,
  - `dynamic_cast` (о нём – в следующей лекции), безопасно преобразующий указатель или ссылку на базовый тип в указатель или ссылку на производный.
- Динамическое приведение следует использовать осторожно. При каждой возможности желательно создавать и использовать виртуальные функции, а не прибегать к непосредственному управлению типами,
- Применение оператора `typeid` к указателю (в отличие от объекта, на который указывает указатель) возвращает статический тип времени компиляции указателя

1. Стенли Б. Липпман, Жози Лажойе, Барбара Э. Язык программирования C++. Базовый курс. 19.2. Идентификация типов времени выполнения

typeid

- `typeid` проверяет точный тип объекта;
- Возвращает std::type_info;
- Работает с любыми типами (но для полиморфных возвращает динамический тип);

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;
}
```

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&) const` C++20
- Так же есть операторы `new`, `delete` и `,` (запятая)
  - `void* operator new (size_t)`
  - `void operator delete (void *)`

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

Оператор космический корабль <=> (1)

- Определяет, какое из соотношений A < B, A == B или A > B имеет место;
- Сгенерированный компилятором оператор трёхстороннего сравнения сравнивает указатели, а не объекты, на которые они ссылаются
- Три категории сравнения:
  - строгая (strong ordering),
  - слабая (weak ordering)
  - частичная упорядоченная (partial ordering)

1. Райнер Гримм. C++20 в деталях. 4.3 Оператор трёхстороннего сравнения

Оператор космический корабль <=> (2)

```
class Date {
public:
    int32_t year, month, day;
    auto operator<=>(const Date&) const = default;
};
```

Оператор космический корабль <=> (3)

```
struct WeakOrdered {
    std::weak_ordering operator<=>(const WeakOrdered&) const;
};

class C {
    WeakOrdered a;
    int32_t b;
    auto operator<=>(const C&) const = default;
};
```

Оператор космический корабль <=> (4)

- Правила вывода типа:
  - Если хотя бы одно поле возвращает partial_ordering, итоговый тип — `std::partial_ordering`,
  - Если есть поля с weak_ordering, но нет partial_ordering, итоговый тип — `std::weak_ordering`,
  - Если все поля возвращают strong_ordering, итоговый тип — `std::strong_ordering`.
- Если в классе есть поля без поддержки <=>, компилятор не сможет сгенерировать оператор по умолчанию.

Code time

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

Домашнее задание

Домашнее задание #3 (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 #3 (2)

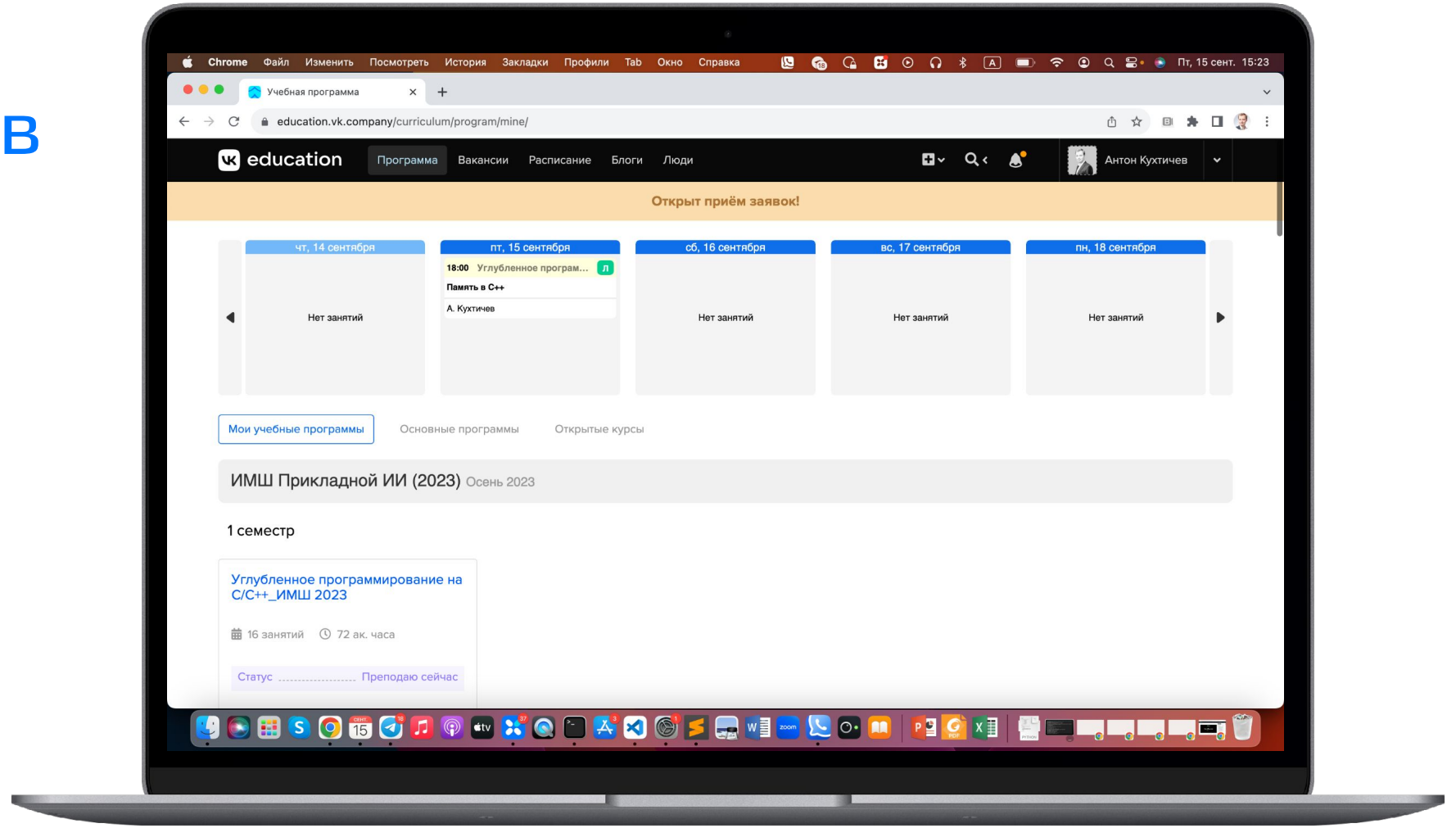
Чтобы реализовать семантику [][] понадобится прокси-класс. Оператор матрицы возвращает другой класс, в котором тоже используется оператор [] и уже этот класс возвращает значение.

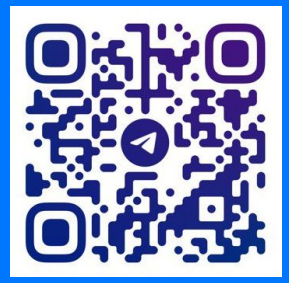
Полезная литература в помощь

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

Напоминание оставить отзыв

Это правда важно





Спасибо
за внимание!