Структуры и классы

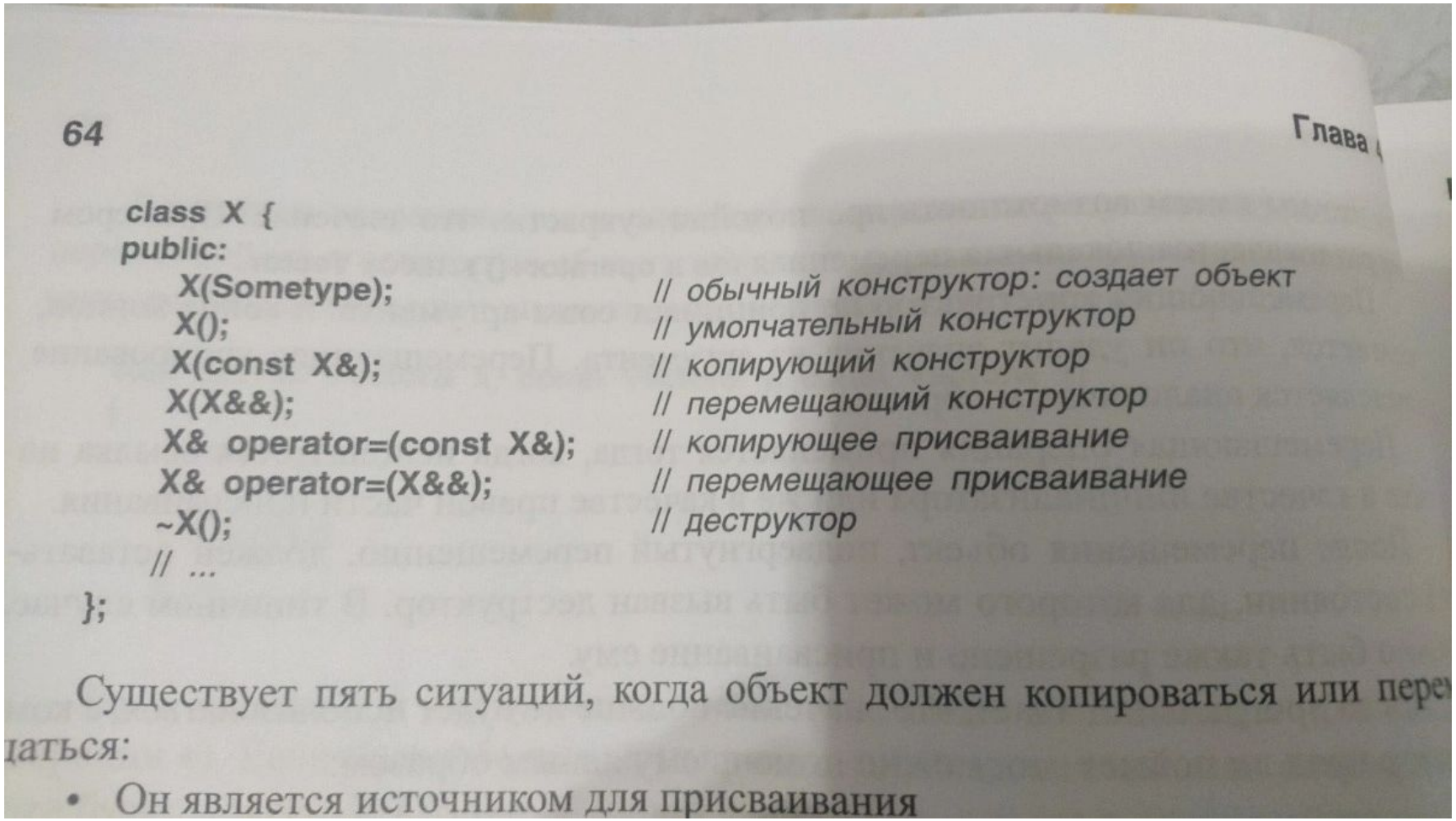
Антон Кухтичев

Содержание занятия

- Пользовательские типы
- Классы и структуры
- Модификаторы доступа
- RAII (Resource Acquire Is Initialization)
- Константные методы
- Наследование
- Перегрузка методов
- Виртуальные функции
- Операторы

Минутка юмора

Язык программирования C++. Стандарт C++11. Краткий курс



64 Глава

```
class X {
public:
    X(Sometype);              // обычный конструктор: создает объект
    X();                      // умолчательный конструктор
    X(const X&);              // копирующий конструктор
    X(X&&);                   // перемещающий конструктор
    X& operator=(const X&);   // копирующее присваивание
    X& operator=(X&&);        // перемещающее присваивание
    ~X();                     // деструктор
    // ...
};
```

Существует пять ситуаций, когда объект должен копироваться или пере
даться:

- Он является источником для присваивания

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. C++ 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. Происходит вызов деструктора;
2. Вызываются деструкторы для полей класса в обратном порядке их объявления в классе;
3. Уничтожаются базовые классы в порядке обратном списку наследования.

Списки инициализаторов членов

1. Все инициализации члена выполняются перед телом конструктора;
    a. Обеспечивает правильность всех элементов перде выполнение 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 неконстатный 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;

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); // !!!
```

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.

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() {}
};
```

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 виртуальные функции.

Таблица виртуальных функций

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;

Виртуальные 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.

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.

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++»

Спасибо за внимание!

Вопросы?