C++

Fall 2020

compscicenter.ru

Филипп Грабовой

gra.filipp@gmail.com
https://t.me/phil_grab

Лекция VII

Classes: Inheritance I

Некоторые специальные методы класса

Поговорим о трёх:

- Copy ctor
- Assignment op
- Dtor

Implicitly-defined

- по умолчанию генерируются компилятором
  - упрощенно: внутри member-wise copy/destroy
- можно декларировать явно через `default`, `delete`

```
struct T {
    T& operator(const T&) = delete;

private:
    T(const T&) = default;
};
```

Mixing with user's

- если класс реализует обертку над ресурсами
  - RAII — Resource Acquisition Is Initialization
  - `std::string`, `std::fstream`, `std::lock_guard`
- и/или реализованные методы *нетривиальны*

Mixing with user's

- скорее всего default-версии других методов **не подойдут**
  - т.к. ими будут нарушаться инварианты
  - например, тривиальное копирование указателя, адреса на дин память
- ⇒ переопределяя один из методов, не забудь про остальные два

Alignment & Padding, Reprs

Alignment

- процессор вычитывает данные по размерам, кратным машинным словам
  - ⇒ 32 vs 64 bits
- рассмотрим два расположения int32_t в памяти (`[..]` — 1 байт (char), int: `[i0-i3]`)

int32_t in memory

```
  -> reading
[i0][i1][i2][i3]
[..][..][..][..]

[..][i0][i1][i2]
[i3][..][..][..]
```

- В каком случае чтение эффективнее? (32 bit system)

Alignment in c++

- компилятор по умолчанию делает эффективнее
  - выравнивает данные на стеке, структуры и т.д.
  - платформо- и компиляторо-зависимо
  - значимо и на built-in типах
  - пример: на ARM допустимы только выровненные по 4 int'ы
- можно управлять: `alignof`, `alignas`, `#pragma pack(…)`

Alignment of CharShort

```
struct CharShort {
    char a;
    short b;
}; // sizeof -> ?
```

Arrays of CharShort

- `CharShort array[42];`:

```
[a0][..][b0][b1]  // array[0]
[a0][..][b0][b1]  // array[1]
...
```

Alignment of structs

```
struct CharIntShort {
    char a;
    int b;
    short c;
}; // sizeof -> ?

struct IntCharShort {
    int a;
    char b;
    short c;
}; // sizeof -> ?
```

Padding for structs

```
struct CharIntShort {
    char a;
    char _padding1[3];
    int b;
    short c;
    char _padding2[2];
}; // sizeof -> 12

struct IntCharShort {
    int a;
    char b;
    char _padding[1];
    short c;
}; // sizeof -> 8
```

Object and Value Representations

- Object Representation: `sizeof(T)` последовательных объектов типа `unsigned char`
- Value Representation: биты, хранящие значение объекта

```
[a0][..][b0][b1]  // obj repr: [a0 - b1]
[a0][..][b0][b1]  // value repr: [a0, b0, b1]
```

Inheritance

Наследование позволяет:

- расширять уже существующие классы
- работать с объектами разных типов однородно (через базовый класс)
- …

Syntax

```
class|struct derived-class-name:
   { access-specifier [virtual] base-class-name, ... }
{ member-specification }

access-specifier — public, protected, private
    влияет на доступ к открытым членам класса base-class-name в наследнике
```

Пример

```
struct GameObject{ Point position; };

class Car: private GameObject {
    Point vel;
    double orien;
    double omega;
};

struct Prize: GameObject {
    int value;
};
```

Object Repr

```
GameObject:
+----------+
| position |
+----------+


Prize:
+----------+-------+
| position | value |
+----------+-------+
|GameObject|
```

Преобразования Base ← Derived

Определены автоматически:

```
Prize p{ Point{...}, 100 };
GameObject &go = p;
GameObject *goPtr = &p;
```

Следовательно, родитель копируем от объекта-наследника:

```
Prize p{ Point{42, 24}, 100 };
GameObject go = p; // GameObject{Point{42, 24}}
```

Срезка: поля только базового класса, утрата других значений/инвариантов и т.д.

Особенности

- Базовый класс должен быть определен до наследования
- Из наследника нет доступа к private полям базового класса, есть к public и protected
  - protected наследование: public-поля Base доступны только в Derived [*]

Derived::Derived

- конструкторы не наследуются (и не бывают виртуальными)
- сконструировать Base-часть до Derived **необходимо**
  - явно или через констуктор по-умочанию
  - до выполнения списка инициализации полей Derived
- порядок конструирования: Base1, Base2 (в порядке объявления наследования), Derived
  - вызовы деструкторов — в обратном порядке

Derived::Derived

```
struct GameObject {
    // no default constructor
    GameObject(Point position) : position{position} {}

    Point position;
};

struct Prize: GameObject {
    Prize(Point pos, int val)
        : GameObject{pos}
        , val{val}
    {}

    Prize(Point p)
        : position(p) // error! not in init-list
        // error! no default ctor for base GameObject
    {}
```

Methods overriding

```
struct GameObject {
    Point position;
    /* ... */
    void CalcShift() { /* ... */ }
};

struct RoadSign: GameObject {
    /* ... */
    void CalcShift() {       // custom method, overrides
        if (HitByCar()) {
            InitShifting();
        }
        GameObject::CalcShift();
    }
};
```

Virtual methods motivation

```
struct GameObject {
    void CalcShift() { /* ... */ }
};

struct RoadSign: GameObject {
    void CalcShift() { /* ... */ }
};

std::vector<GameObject *go> objects;

int main() {
    GameObject *go = new RoadSign{...};
    objects.push_back(go);
    objects[0]->CalcShift();
}
```

Virtual methods syntax

```
virtual member-function [override] [final] [= 0;]

override — компилятор проверит, что функция с такой сигнатурой есть в предке
final — запрет переопределения в потомках
= 0; — pure virtual function (class → abstract class, нельзя создавать объекты)
```

Virtual methods example

```
struct GameObject: VisibleObject {
    virtual void CalcShift() { /* ... */ }
};

struct RoadSign: GameObject {
    virtual void CalcShift() override {
        /* ... */
        GameObject::CalcShift();
    }
};
```

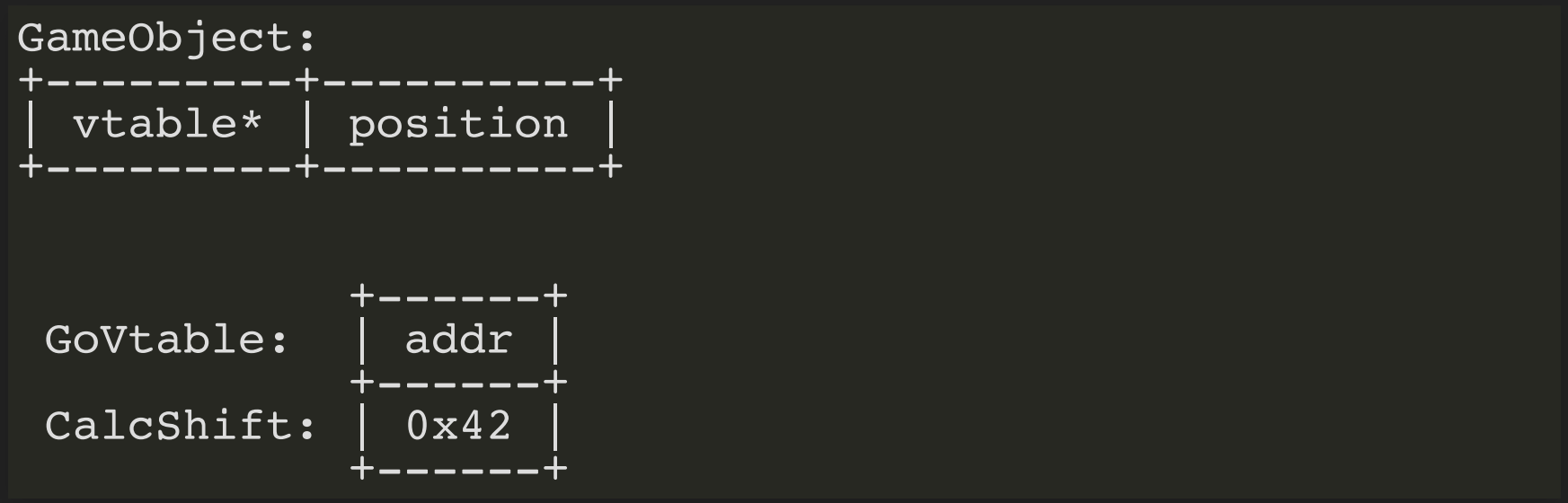
Abstract classes example

```
struct VisibleObject {
    virtual void Render() = 0;
}

// defined outside of class
void VisibleObject::Render() { /* ... */ }

struct RoadSign: VisibleObject {
    virtual void Render() final {
        /* ... */

        // called explicitly
        VisibleObject::Render();
    }
};
```

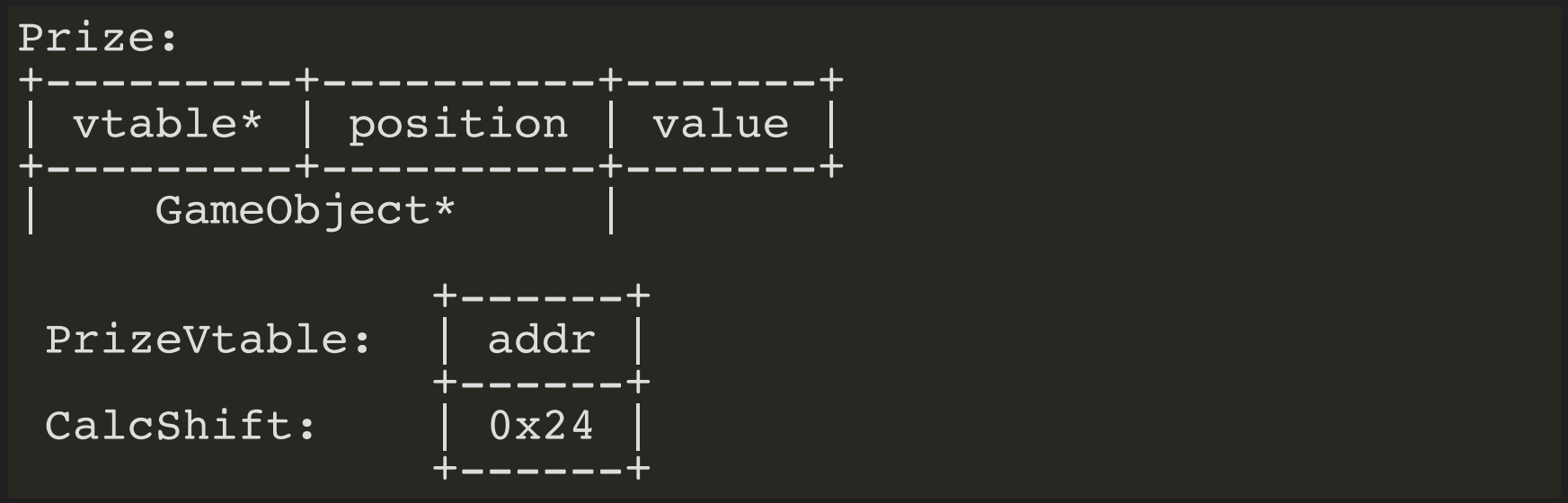
Virtual methods

- реализация — vtable
  - таблица виртуальных функций (в начале класса)
- важен виртуальный деструктор при наследовании
- виртуальные методы **не стоит** использовать в конструкторах и деструкторах

GameObject Repr



Prize Repr


Prize:
vtable*
position
value
GameObject*
PrizeVtable:
addr
CalcShift:
0x24