Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

21 ноября 2017 г.

Оглавление

12 Виртуальные методы **3**

12.1 Вызовы методов при работе с классами-наследниками 3

12.2 Виртуальные методы 4

12.3 Множественное наследование 5

Лекция 12

Виртуальные методы

12.1 Вызовы методов при работе с классами-наследниками

Простейшие классы – структуры. Напишем на них пример:

Код 12.1: Пример виртуальных функций

```
#include<iostream>

 struct A
 {
      char x;
      void bar() {
          std::cout << "A";
      }
};

struct B : A    // struct inheritance is always public
{
    int y;
    void bar() {
        std::cout << "B";
    }
};

struct C: A
{
    double z;
    void bar() {
        std::cout << "C";
    }
};
```

Заметьте, что у структур наследование всегда публичное.

Распределение памяти

Экземпляр класса A будет занимать память, соответствующую размеру одного **char**'а, B – уже одного **char** и одного **int** (для шарящих: пока пренебрежем выравниванием), C – **int** и **double**. При этом занимаемые размеры расположатся в таком порядке: $a < b < c$.

12.2 Виртуальные методы

Рассмотрим пример:

Код 12.2: Вызов наследника вместо родителя

```
#include "1.cpp"

void foo(A& p)
{
    p.bar();
}

int main() {
    A a;    // Create class objects
    B b;
    C c;
    foo(b); // ???
    foo(c); // Will this stuff work??
}
```

Какая функция будет вызвана в **foo()**? Ответ прост: всегда из класса **A**. Дело в том, что функция **foo** *компилируема*: при сборке проекта **p** будет 'увиден' компилятором как экземпляр класса **A** и вызов **p.bar()** будет привязан к области памяти **A**.

Как бороться с этим? Есть несколько способов:

1. Поле типа (не рекомендуется). Добавим специальный атрибут **int m_type** в базовый класс. В конструкторах производных классов заполним **m_type** своими ID типа (например, у **A** будет 0, у **B** будет 1). При этом будет происходить *коммутация* вызовов:

   p.bar() превратится вот в такое:

Код 12.3: Условия на поле типа

```
if(p.m_type == 1) { // is A
    p.bar();
} else if(p.m_type == 2) {  // is B
    static_cast<B&>(p).bar();
```

```
5       // Cast p to B&, after that call bar() -- it will be from B
6   } else {    // is C
7       static_cast<C&>(p).C::bar();
8       // Call bar() from C namespace, the same idea
9   }
```

Это кошмар. Нам надо знать все классы, которые наследуют базовый, да и к тому же вести их учет через ID. Есть альтернатива.

2. Сохраним указатель на функцию в классе. Она уйдет всем наследникам. В конструкторе каждого наследника мы будем переопределять ссылку на этот метод, вот так:

Код 12.4: Переопределение указателя на функцию

```
1   *p.p_bar();
```

Остаются проблемы. Вдруг дочерний класс будет использовать родительский метод? Вдруг таких методов будет много? Приведем еще один способ.

3. Можно хранить не одну ссылку на метод, а указатель на таблицы, содержащей указатели на нужные методы. Те же костыли, те же проблемы. Запутаться еще легче, ведь для каждого класса эта таблица будет своя.

Но есть и хороший способ. Все это уже встроено в язык. Все обрабатывается автоматически. Для этого нужно только одно – указать перед методом слово **virtual**. Для всех потомков будет создана таблица виртуальных методов (подобно той, что указана в п.3 выше), которая будет автоматически заполняться у потомков.

Итак, *виртуальные* методы – те методы, которые предполагаются быть переопределенными в дочерних классах.

12.3 Множественное наследование

Абстрактный класс

Класс, у которого есть хотя бы один абстрактный метод с нулевой реализацией, является **абстрактным**. Его экземпляры нельзя создавать, но при этом абстрактные классы обладают мощной штукой – множественное наследование.

Код 12.5: Абстрактный метод с нулевой реализацией

```
1   class Test {
2       virtual foo() = 0;  // Zero realization
3   }
```

Множественное наследование

Множественное наследование — это явление, когда один класс наследует несколько других. При этом класс, как это говорят, будет удовлетворять нескольким *интерфейсам*. У такой штуки есть масса проблем. Что, если оба класса имеют два одинаковых метода с одинаковой сигнатурой? А атрибуты? С ними могут возникнуть большие проблемы при пересечении. Но если все методы виртуальны, а атрибутов нет, то все славно: даже если классы содержат пересекающиеся методы, мы сможем выбрать для переопределения любой из них (компилятору будет все равно, ведь они все равно не содержат реализации). Именно поэтому рекомендуется делать множественное наследование только в абстрактных классах и наследовать при этом только абстрактные классы.

Применение абстрактных классов

Итак, в чем польза абстрактных методов и классов? Они позволяют грамотнее и логичнее организовывать структуру кода, объединяя схожие классы. Наглядный пример их использования мы видели в лабе 12 'Арифметические драконы', когда реализовывали класс **Dragon**, а впоследствии и класс **Enemy** – каждый из них не обозначал какой-либо реальный объект сам по себе, однако служил шаблоном для клепания других классов, уже вполне связанных с реальными объектами (например, **Dragon** служил образом для класса **GreenDragon**).