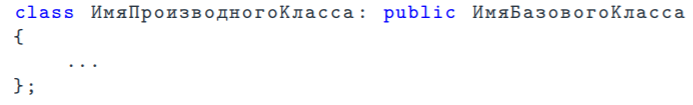
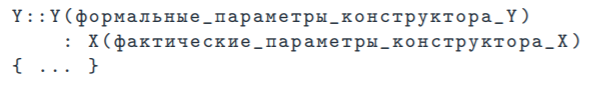
# Общая информация

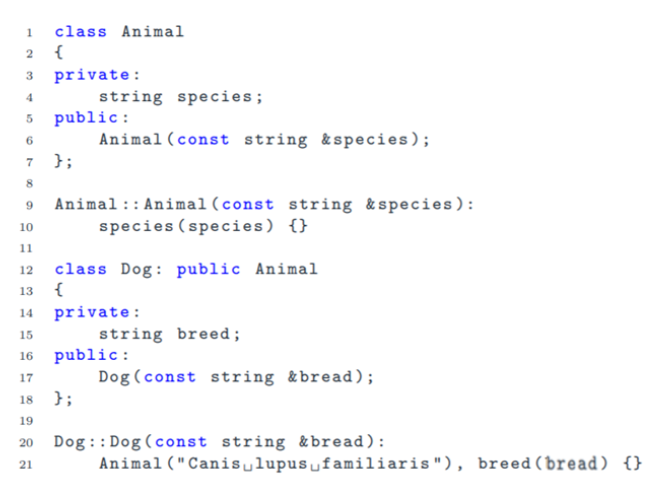


Если конструктор базового класса не имеет параметров (является конструктором по умолчанию), его вызов добавляется компилятором С++ в конструктор производного класса автоматически.

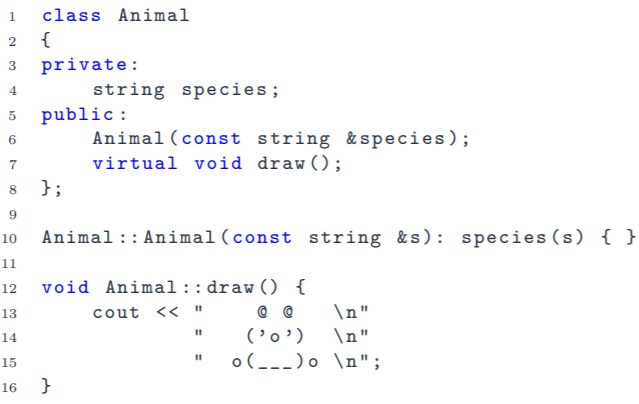
В противном случае необходимо явно вызывать конструктор базового класса в конструкторе производного класса. В языке C++ вызов конструктора базового класса X из конструктора производного класса Y выполняется аналогично вызовам конструкторов полей:

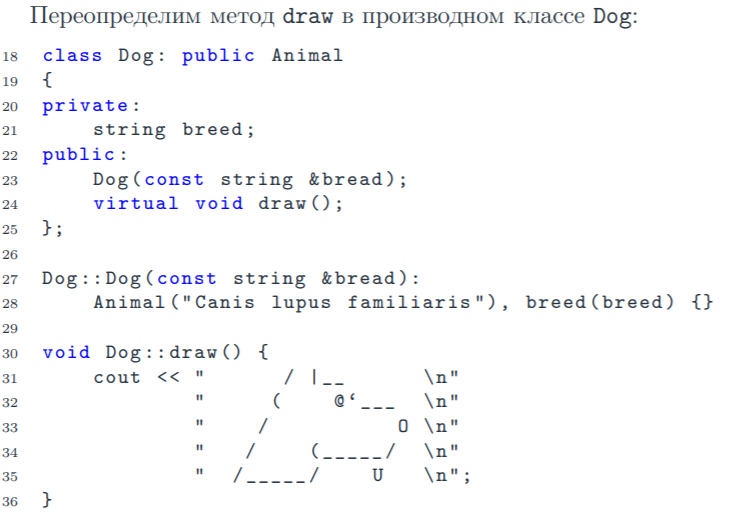


Пример

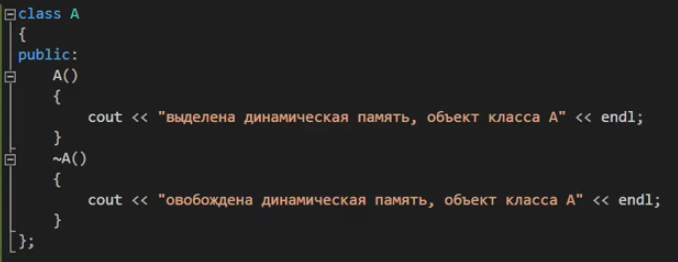


В C++, как и в языке Java, для переопределения метода, унаследованного от базового класса, достаточно объявить и определить этот метод в производном классе. В качестве примера рассмотрим базовый класс Animal:

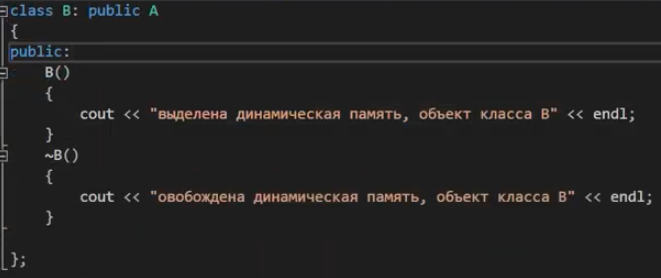




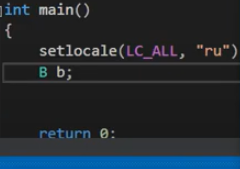
Рассмотрим **переопределение деструктора в C++ и разберем**, зачем оно вообще нужно. Допустим у нас есть класс А



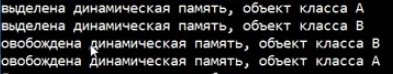
И класс B, который мы наследуем от А



Сначала просто создадим объект класса B в main() и посмотрим что будет.

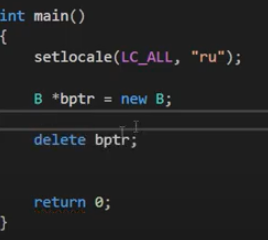


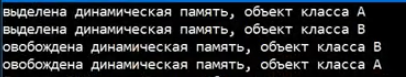
Результат:



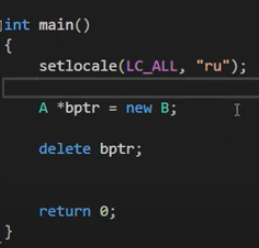
Комментировать особо нечего, принцип ABBA работает даже тут(и вы говорите, что Головин – не гений?)

Теперь выделим память используя указатели





Аналогично. И вроде все хорошо, но, как говорил Василий Иваныч, есть нюанс!



А че, так можно было? А почему нет? Указатель на базовый класс может хранить ссылки на своих потомков. Такие вещи иногда применяются. Например Animal animal = new Dog();

Т. е. мы выделили память под животных, а потом уточняем, что нам нужна собака, и используем конструктор из класса “Dog” (Ир, это совпадение, честное слово, просто такой пример нашел)

Так вот, посмотрим, что будет



Деструктор класса B не вызвался! Значит, произошла утечка памяти. Чтобы этого избежать, надо сделать деструктор базового класса виртуальным.

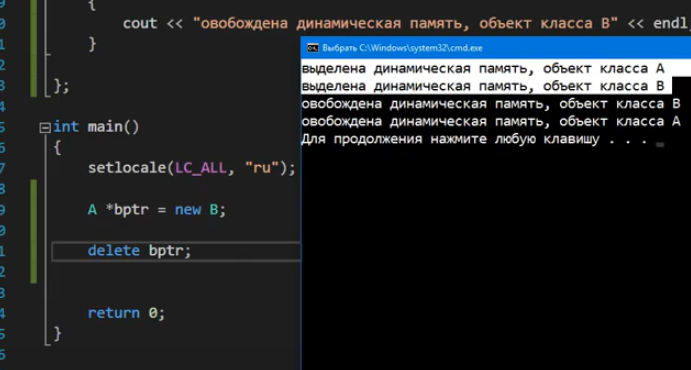


И еще сделать вот так



В целом тут прописывать override не обязательно (все и так будет работать), но все-таки это нужно для понимания (мы же все-таки переопределяем деструктор)

Итог:



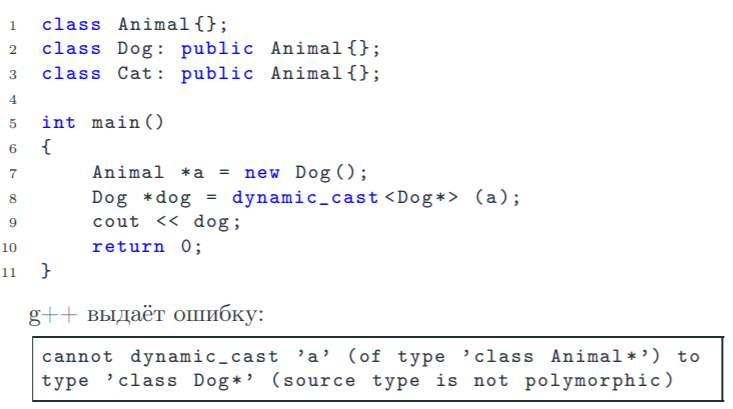
# Динамическое приведение типов

Операция динамического приведения объекта obj к типу T проверяет, является ли тип T одним из типов объекта obj, и возвращает obj, если является. В противном случае операция либо возвращает нулевой указатель, либо порождает исключение. В C++ операция динамического приведения типа может применяться к указателям и ссылкам на объекты и записывается как

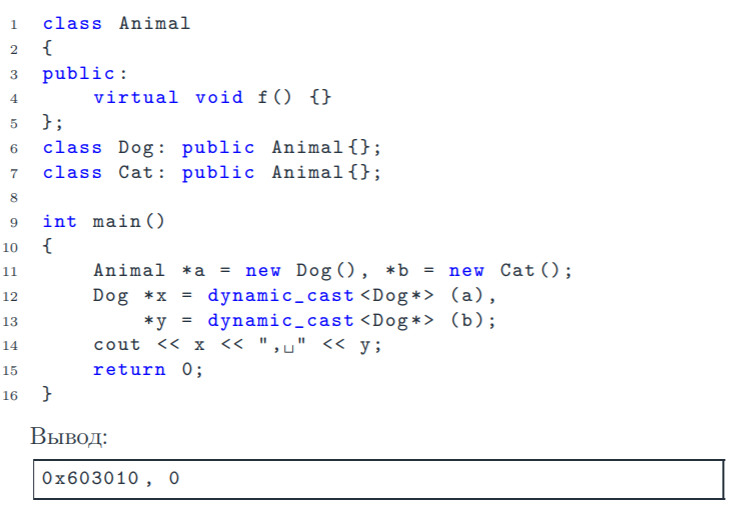


Так как нулевых ссылок не существует, то при неудачном приведении порождается исключение std::bad\_cast.

Пример:



Т. к. в классе Dog нет виртуальных методов, то для экономии памяти объект этого класса не содержит информации о типе, тем самым делая невозможным динамическое приведение типов.



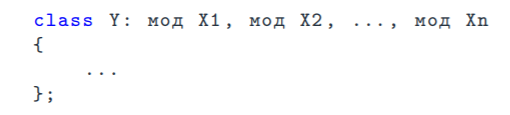
# Типы наследования в С++



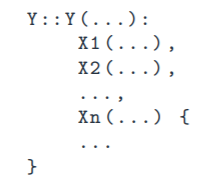
Думаю, комментарии излишни

# Наследование от нескольких базовых классов

C++ поддерживает множественное наследование. Если класс Y – производный от нескольких базовых классов X1, X2, ..., Xn, то объявление класса Y выглядит на C++ как

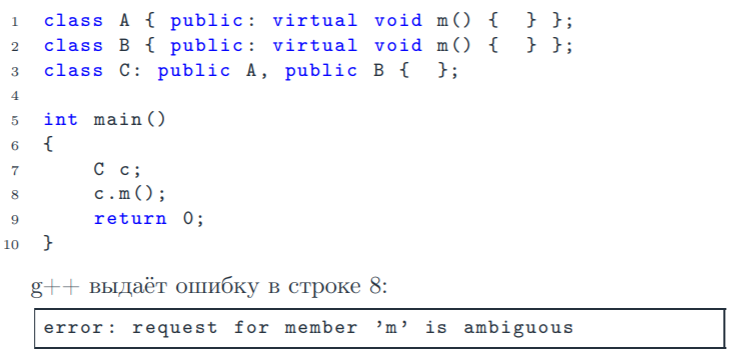


Здесь «мод» – это public, protected или private. При этом в конструкторе класса Y должны выполняться вызовы конструкторов всех базовых классов, кроме, возможно, тех из них, которые имеют конструкторы по умолчанию:

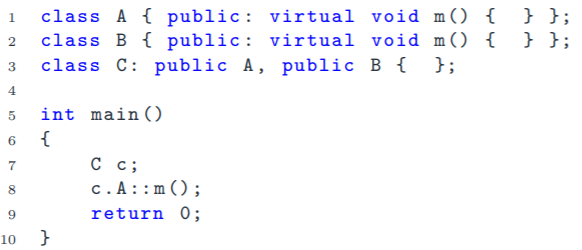


# Противоречия в именах наследуемых членов класса

При множественном наследовании в производном классе могут возникать два или более методов с совпадающими именами и сигнатурами. То же самое справедливо и для полей. Пример:

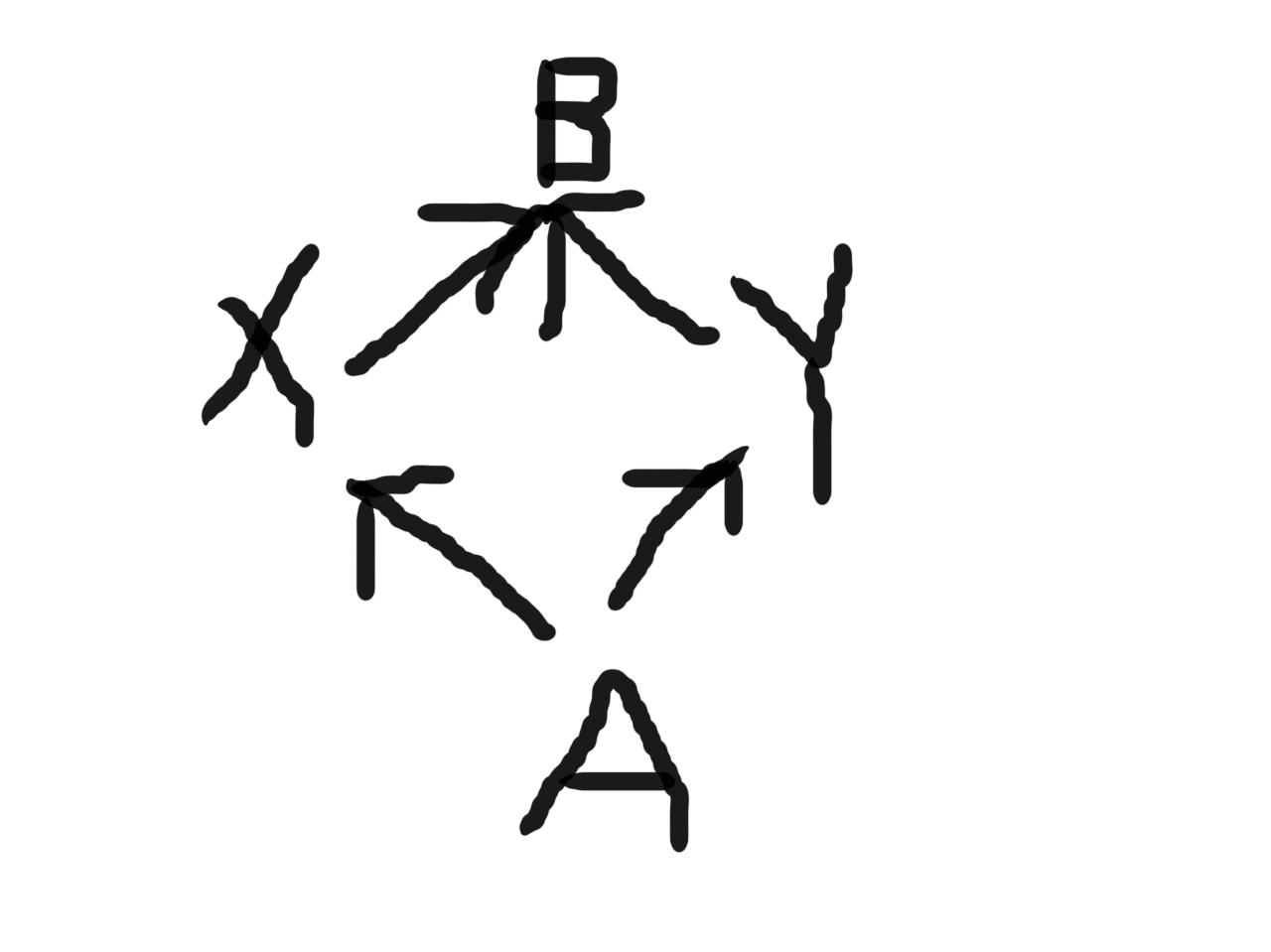


Для разрешения противоречий в именах наследуемых членов класса необходимо использовать квалифицированные имена. Пример:



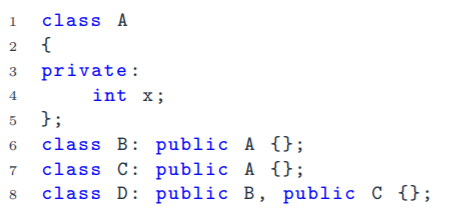
# Иерархия наследования и классы противоречия

Иерархия наследования – это ориентированный ациклический граф, множеством узлов которого является множество классов программы. При этом если класс Y является непосредственным базовым классом для класса X, то из узла X исходит дуга, входящая в Y. Мы будем говорить, что класс B является классом противоречия, если существует такой класс A, что в иерархии наследования можно провести не менее двух непересекающихся путей из A в B. Иерархия наследования называется противоречивой, если в ней существуют классы противоречия.



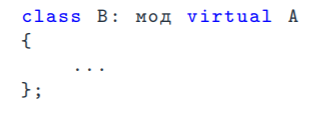
-класс противоречия, иллюстрация

Основной проблемой противоречивых иерархий является возможность многократного включения полей базового класса в производный класс. Пример:



Поле x будет содержаться в объекте класса D дважды.

Избежать многократного включения базового класса в производный класс позволяет виртуальное наследование. Виртуальное наследование – это способ реализации наследования, гарантирующий, что базовый класс не будет включён ни в один из производных классов более чем в одном экземпляре. Если класс B виртуально наследует классу A, то говорят, что A – виртуальный базовый класс для класса B. Синтаксически это выражается как



Здесь «мод» – это public, protected или private.

