Создадим своё ООП.

Задача оценивается в 1 балл.

Задача

Имитировать наследование и виртуальные методы на языке С.

Ваша прграмма должна содержать 2 структуры. Одна структура - *базовая*. То есть не имеет “предка”. Другая структура - *наследник* базовой. (Это должно быть как-то помечено в программе).

У базовой структуры есть несколько “методов”. Структура-наследник переопределяет некоторые “методы” базовой структуры. (Определение методов тоже нужно как-то пометить)

В **main()** создайте 2 “объекта” (структуры) разных типов: базовый и наследник. Продемонстрируйте, что произойдёт при вызове разных методов этих объектов:

- Метод, определённый в базовом “классе”, но не в классе-наследнике должен быть найден и вызван для наследника.
- Метод, определённый в базовом “классе”, но переопределенный в наследнике, должен учитывать тип объекта, у которого метод вызван.
- Метод, не определённый для данного класса, может компилироваться, но бросать исключение. Или не компилироваться - на Ваше усмотрение.

Требования к реализации

1) Вы можете использовать компилятор C++;
2) Не разрешается использовать ключевое слово **virtual**;
3) Не разрешается использовать готовые системы макросов, например из Qt;
4) Реализация методов должна идентифицировать вызов метода, например, выводите на консоль имя метода;
5) Достаточно поддержать методы, не имеющие аргументов, не возращающие значение (**void method();**).

Рекомендации

Для расширения языка рекомендуется использовать макросы.
Рекомендуется повторить систему таблиц виртуальных функций, которую генерирует C++.

Пример программы

```
// б а з о в ы й  к л а с с
VIRTUAL_CLASS( Base )
      int a;
END( Base )
// м е т о д ы
DECLARE_METHOD( Base, Both )
DECLARE_METHOD( Base, OnlyBase )

// к л а с с - н а с л е д н и к
VIRTUAL_CLASS_DERIVED( Derived, Base )
      int b;
END_DERIVE( Derived, Base )
// м е т о д ы
DECLARE_METHOD( Derived, Both )
DECLARE_METHOD( Derived, OnlyDerived )


void main()
{
      Base base = /* к а к - т о  с о з д а л и  б а з о в ы й  к л а с с */
      base.a = 0; // р а б о т а е м  к а к  с о  с т р у к т у р о й
      Derived derived = /* ... к а к - т о  с о з д а л и  н а с л е д н и к */

      // п о л и м о р ф и з м
      Base* reallyDerived = reinterpret_cast<Base*>(&derived);

      VIRTUAL_CALL(&base, Both); // п е ч а т а е т  “Base::Both a = 0”
      VIRTUAL_CALL(reallyDerived, Both); // п е ч а т а е т  “Derived::Both b
= 1”
      VIRTUAL_CALL(reallyDerived, OnlyBase);  // п е ч а т а е т
“Base::OnlyBase”
      VIRTUAL_CALL(reallyDerived, OnlyDerived);
}
```