本文内容：

1. 题目
2. 对题目的解释与分析
3. 本题的练习目的

【题目】

Given the following class Animal :

#include <iostream>

#include <string>

class **Animal** {

**protected:**

std::string name;

**public**:

Animal(const char\* nm ="A") : name("A") {

if(nm != NULL) name=nm;

}

Animal(const std::string& nm) : name(nm) { }

**virtual** **void** **iam** () { std::cout << "Animal" << name << \n"; }

**void hello** () { std::cout << "Animal::hello from " << name; }

void common() { std::cout << "Animal::common"; }

};

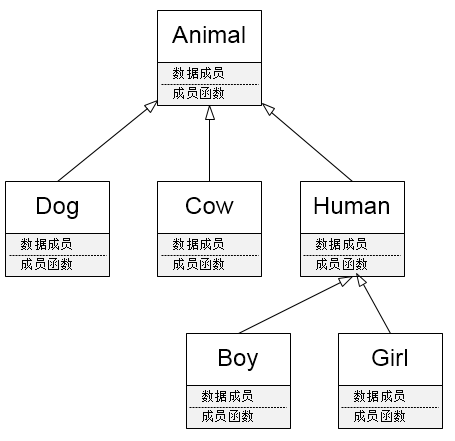
Please define two or more (directly or indirectly)derived classes from ***Animal***. For each derived class, according you wish, re-define ***iam***() to write out the name of the class and the object, and re-define ***hello***() to write out any different text with the name of the object.

Please create objects of these classes and call ***iam***(),***hello***() and common() for them. Assign pointers to objects of the derived classes to “***Animal \*”*** pointers and call ***iam***() , ***hello***() and common() through those pointers.

【解释与分析】为达到文末的练习目的，此处解释较多，耐心阅读。

题目给出了顶层基类**Animal**的完整定义，现要求你来定义 **Animal**的两个或更多（直接或间接）派生类（类名自定，如狗、猫，人等等）。你也可以在派生类中可以添加额外的数据成员。

右下类图是一个示意性的类层次结构设计。三角箭头代表类间的继承关系，它指向的是基类，另一个端是子类。

题目要求各个子类各自重新定义成员函数 iam() 和hello()，其中 iam() 输出各自所属的类型名和对象成员name的值，hello() 输出任意不同的文本。

提示1：题中的重新定义(re-define)有两方面含义： 对于虚函数（iam），在子类中“重置”它，即这两个函数在子类中的接口和 Animal 中一样，但重新给出实现；对于非虚函数（hello），在子类中也重新实现(保持函数头部不变)，但这不是重置。

提示2：题中的“according your wish”指某些子类重新定义两个函数中的一个或两个，而某些子类不重新定义这俩函数。

题目还要求测试你的类层次结构。在测试程序中，

1. 定义每个类的对象，并通过对象直接调用 iam() 、 hello() 和 common()；
2. 定义若干**Animal**\* 指针，有些指向**Animal**对象，有些指向派生类对象。然后通过这些指针去调用 iam() 、 hello() 和 common()；
3. 此外，建议你也通过定义 **Animal**& 引用名，有些引用**Animal**对象，有些引用派生类对象。然后通过这些引用名去调用 iam()、 hello() 和 common()。

程序执行时，请认真观察、对比以上各方式产生的输出结果，从而判定被调用的函数是哪个类的。

\*\* 为加深理解“重置是可选择的，而不是强制的”，你在上述要求基础上，可定义 Animal的第三个派生类（但不重置iam），或定义若干***派生类的派生类***（如上图中的 Boy 和 Girl，它们可有选择地去重置 iam）；再通过任一基类指针去调用 iam()，通过程序的输出结果看看到底调用哪个类的iam()。

\*\* 你也可以定义一个全局函数，令其参数类型为 **Animal** & 。函数体中通过该引用调用 iam() 等函数。调用时的对应实参可以是基类对象，或派生类对象，以提升你对上述内容的感性认识。

\*\* 你也可以定义一个全局函数，令其参数类型为 **Animal** \* 。函数体中通过该指针调用 iam()等函数。调用时的对应实参可以是任意类型对象的指针：基类对象、或派生类对象、或派生类的派生类对象，以提升你对上述内容的感性认识。

这里给出部分供参考的示例性的测试函数：

void invoke\_base(**Animal** \* ptr){

cout << “Function ” << \_\_FUNCTION\_\_ << endl;

ptr-> iam(); ptr->hello(); ptr->common();

}

void invoke\_base(**Animal** & ref){

cout << “Function ” << \_\_FUNCTION\_\_ << endl;

ref. iam(); ref.hello(); ref.common();

}

void invoke\_Human(**Human** & ref){

cout << “Function ” << \_\_FUNCTION\_\_ << endl;

ref. iam(); ref.hello(); ref.common();

}

// 下面假设 subclass是你定义的某个派生类

void invoke\_subclass(subclass\* ptr){ ptr-> iam(); ptr->hello();ptr->common();}

void invoke\_subclass (subclass & ref){ ref. iam(); ref.hello();ref.common();}

下面给出简单的主函数示例：

int main()

{

**Animal** a, b(“Goofy”);

subclass s1(“Woof”);

a.iam(); a.hello(); a.common();

b.iam(); b.hello(); b.common();

s1.iam(); s1.hello(); s1.common();

invoke\_base ( &b ); invoke\_base ( &s1 ); // 通过对象指针 调用

invoke\_base ( b ); invoke\_base ( s1 ); // 通过对象引用 调用

invoke\_subclass ( &s1 ); // 通过对象指针 调用

invoke\_subclass (s1); // 通过对象引用 调用

}

【本题目的】

希望大家通过这个练习，掌握这些重要的面向对象程序设计技术：

1. 继承、虚函数的基本概念和作用。使用继承、虚函数的基本原则有：
   1. **同则合并**：相同的特征（包括数据成员、函数接口、函数实现）均抽取并定义到父类；
   2. **异则拆分**：有差异的特性（包括数据成员、函数实现）留在各子类中。
2. 使用**继承**的目的在于：
   1. 对全体对象（即数据、实体）进行**分类**，并将分类关系在程序中明确地表述出来；
   2. 父类规定其所有子类对象的共同特性，而子类仅需定义自己的特性；

如本题假设每个动物都有名字，所以用基类Animal的成员 name 来存储之，子类就不用再次定义了。

* 1. 子类可**重用**（基类）成员。

1. 使用**虚函数（及重置）**的目的在于：
   1. 父类规定了所有子类对象**都有**的、相同的对外接口；
   2. 子类在继承父类时，对于父类声明的某个操作，有两种选择：
      1. 直接继承：若父类的某个操作之实现（即语义）也适应于子类对象，则子类无需再次声明该操作。该操作可以是虚函数，也可以不是虚函数。 如题目中的 common()。
      2. 重置：若父类的某个操作无法给出定义、或其实现不适应于子类，则子类仅继承该操作的接口，同时给出自己的实现。

此时该操作在父类中应声明为虚函数，如题目中的 iam()。

【注意】题目也要求子类再次定义一下非虚函数hello()。该要求指明了一种对重置的“误用”。

* 1. 对于父类中声明的虚函数，无论子类是否重置，都意味着父类对象、子类对象均有这样的操作，且操作的接口是相同的。

1. 当通过“基类指针/引用”来访问派生类对象时，该对象**被当做**基类对象看待，因为它们具有基类所规定的特性（接口）；
2. 当通过“基类指针/引用”来调用基类中声明的虚函数时，实际被调用的函数是由该指针/引用所引用的对象之实际类型决定【这就是一种“多态”现象】。这样便于对一组对象进行一致的管理和操作。

以上各条，请自行对比、观察被调用的成员函数，进一步理解继承与重置的作用。

【本文完】