**暨南大学本科实验报告专用纸**

课程名称 C++程序设计 成绩评定

实验项目名称 多重继承与虚继承编程实验 指导教师 王勇杰

实验项目编号 14 实验项目类型 编程型 实验地点 C105

学生姓名 李炜鹏 学号 2017052544

学院 智能科学与工程学院 系 计算机 专业 信息安全

实验时间2019年12月10日 午～12月13日 午 温度 ℃湿度

1. **实验目的**
2. 认识并且理解多重继承和虚继承的概念
3. 熟悉C++继承功能的编程操作
4. 实验要求：阅读课本第18章18.3的内容， 试调试运行实验十四中的源代码
5. 实验报告要求：
   1. 实验目的
   2. 实验原理
   3. 注释必要的代码
   4. 实验评估
6. **实验原理**
7. 多重继承：是指从多个直接基类中产生派生类的能力，多重继承的派生类继承类所有父类的属性。
8. 多重继承下的类作用域：在只有一个基类的情况下，派生类的作用域嵌套在直接基类和间接基类的作用域中，查找过程沿着继承体系自底向上进行，指导找到所需的名字，派生类的名字将隐藏基类的同名成员。
9. 虚继承（虚派生）：在默认情况下，派生类中含有继承链上梅格雷对应的字部分，如果每个类在派生过程中出现了多次，则派生类中将包含该类的多个子对象。于是，C++引入了虚继承的概念，徐概念的目的是令某个类作出声明，承诺愿意共享它的基类。其中，共享的基类子对象称为虚基类。虚继承只影响从指定了虚基类的派生类中进一步派生出来的类，它不会影响派生类本身。
10. 构造函数与虚继承：先构造虚基类部分，再按照初始化列表构造，最后构造最底层派生类
11. **主要仪器设备**

**仪器：**计算机

**实验环境：**Visual Studio 2017或Dev-C++

1. **源程序**

**Animal\_virtual\_baseVers.h**

#include <string>

#include <iostream>

class Endangered {

public:

enum Status { critical, environment, improving };

Endangered(Status stat = improving) : animal\_status(stat) { }

virtual ~Endangered()//虚析构函数(作为基类一般都需要定义虚析构函数)

{

std::cout << "Endangered::~Endangered" << std::endl;

}

virtual std::ostream& print(std::ostream&) const//虚常成员函数print

{

return std::cout << "Endangered::print" << std::endl;

}

virtual void highlight() const//虚常成员函数highlight

{

std::cout << "Endangered::highlighted" << std::endl;

}

// . . .

private:

Status animal\_status;

};

class ZooAnimal;

extern std::ostream& operator<<(std::ostream&, const ZooAnimal&);//声明函数或全局变量的作用范围(外部的：外部可以引用)

class ZooAnimal {

public:

ZooAnimal() : exhibit\_stat(false) { }

ZooAnimal(std::string animal, bool exhibit, std::string family) : nm(animal), exhibit\_stat(exhibit), fam\_name(family) { }

virtual ~ZooAnimal()//虚析构函数(作为基类一般都需要定义虚析构函数)

{

std::cout << "ZooAnimal::~ZooAnimal" << std::endl;

}

virtual std::ostream& print(std::ostream&) const//虚常成员函数print

{

return std::cout << "ZooAnimal::print" << std::endl;

}

virtual int population() const//虚常成员函数population

{

std::cout << "ZooAnimal::population" << std::endl; return 0;

}

//常成员函数：返回数据成员

std::string name() const { return nm; }

std::string family\_name() const { return fam\_name; }

bool onExhibit() const { return exhibit\_stat; }

// . . .

protected:

std::string nm;

bool exhibit\_stat;

std::string fam\_name;

// . . .

private:

};

//关键字public和virtual的顺序并不重要

class Raccoon : public virtual ZooAnimal {//虚继承ZooAnimal类

public:

Raccoon() : pettable\_flag(false) { }

Raccoon(std::string name, bool onExhibit = true);

virtual std::ostream& print(std::ostream&) const//虚常成员函数print

{

return std::cout << "Raccoon::print" << std::endl;

}

bool pettable() const { return pettable\_flag; }//返回受保护数据成员pettable\_flag

void pettable(bool petval) { pettable\_flag = petval; }//修改受保护数据成员pettable\_flag

// . . .

protected:

bool pettable\_flag;

// . . .

};

class Bear : virtual public ZooAnimal {//虚继承ZooAnimal类

public:

Bear(std::string name, bool onExhibit = true);

protected:

Bear() : dance\_flag(two\_left\_feet) { }

public:

enum DanceType { two\_left\_feet, macarena, fandango };

virtual std::ostream &print(std::ostream&) const//虚常成员函数print

{

return std::cout << "Bear::print" << std::endl;

}

virtual std::string isA() const//虚常成员函数isA

{

std::cout << "Bear::isA" << std::endl;

return "Bear";

}

int mumble(int)

{

std::cout << "Bear::mumble" << std::endl; return 0;

}

void dance(DanceType) const

{

std::cout << "Bear::dance" << std::endl;

}

virtual ~Bear() { std::cout << "Bear::~Bear" << std::endl; }

private:

std::string name;

DanceType dance\_flag;

};

class Panda : public Bear, public Raccoon, public Endangered {//直接继承于Bear类、Reccoon类、Endangered类

public:

Panda() : sleeping\_flag(false) { }

Panda(std::string name, bool onExhibit = true);

virtual std::ostream& print(std::ostream&) const//虚常成员函数print

{

return std::cout << "Panda::print" << std::endl;

}

bool sleeping() const { return sleeping\_flag; }

void sleeping(bool newval) { sleeping\_flag = newval; }

// . . .

protected:

bool sleeping\_flag;

// . . .

};

//Bear类、Reccoon类、Panda类的构造函数的定义

Bear::Bear(std::string name, bool onExhibit) : ZooAnimal(name, onExhibit, "Bear") { }

Raccoon::Raccoon(std::string name, bool onExhibit) : ZooAnimal(name, onExhibit, "Raccoon"), pettable\_flag(false) { }

Panda::Panda(std::string name, bool onExhibit) : ZooAnimal(name, onExhibit, "Panda"), Bear(name, onExhibit), Raccoon(name, onExhibit), Endangered(Endangered::critical), sleeping\_flag(false) { }//先构造虚基类ZooAnimal部分，在按照初始化列表分别构造Bear、Reccoon、Endangered部分，最后构造自身（Pander）部分

**virt-inherit.cpp**

#include "Animal\_virtual\_baseVers.h"

#include <iostream>

using std::cout; using std::endl; using std::ostream;

void dance(const Bear&)

{

cout << "dance(const Bear&)" << endl;

}

void rummage(const Raccoon&)

{

cout << "rummage(const Raccoon&)" << endl;

}

ostream& operator<<(ostream&, const ZooAnimal&)

{

return cout << "ZooAnimal output operator" << endl;

}

int main()

{

Panda ying\_yang;

dance(ying\_yang); //正确：把一个Panda对象当成Bear传递

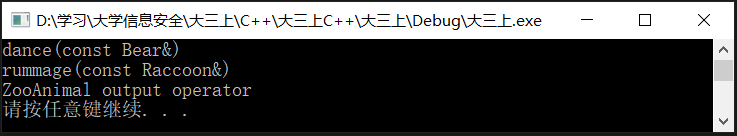
rummage(ying\_yang); //正确：把一个Panda对象当成Raccoon传递

cout << ying\_yang; //正确：把一个Panda对象当成ZooAnimal传递

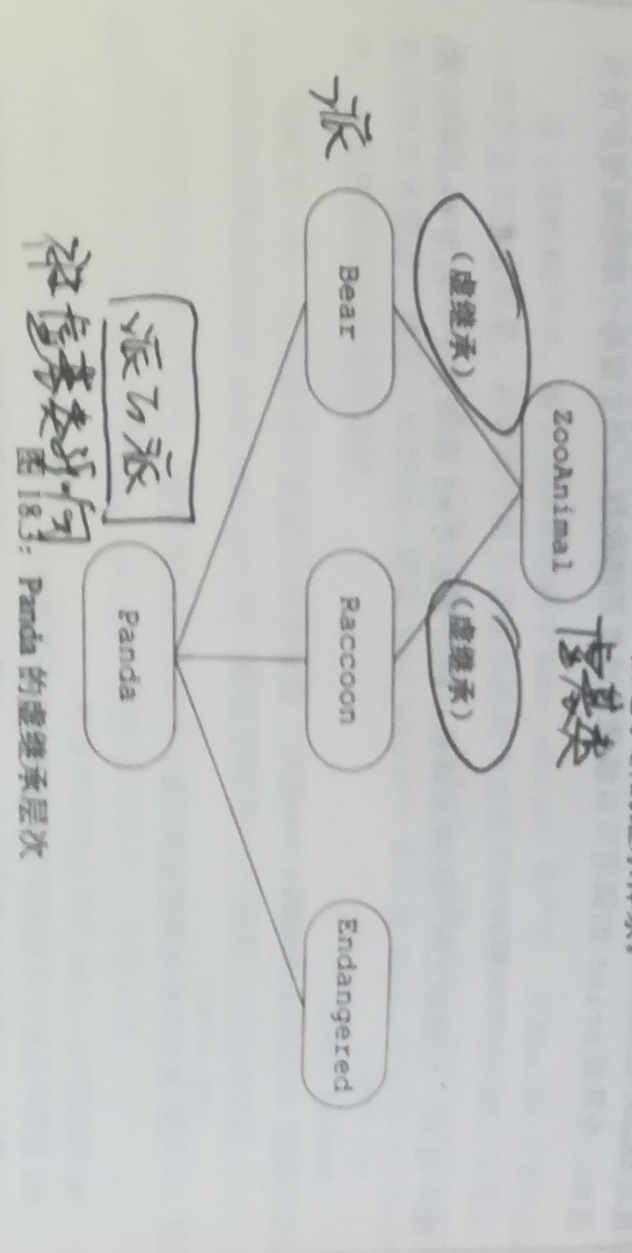
system("pause");

return 0;

}



本实验定义的类继承关系图如教材P718图18.3，如下：



其中定义Bear类和Raccoon类的时候声明为从ZooAnimal类中继承，继承之后（包括所有类型的继承），派生类可以以最安全的权限使用父类的数据成员或者成员函数，也可以重写从父类继承的成员函数并且覆盖它，从虚基类的派生类派生而来的子类只继承一个虚基类子对象（而不重复继承），在main函数中通过最底层子类对象引用其直接基类和虚基类的成员函数，运行结果符合预期。

1. **实验评估**

本实验主要涉及了C++面向对象编程的多重继承以及虚基类的概念，通过本实验，我理解了C++中多重继承的概念，虚基类的派生类中的构造函数和析构函数的实现顺序过程，子类对象引用（虚）基类成员函数时，访问的过程。完成本实验之后，本学期C++程序设计的课程内容已经基本完成，经过这一个学期的理论学习与自学过程中的实践，我对C++的编程思想已经有了深刻的理解，包括C++与C语言的区别，C++11新增的内容，我已经有了一个基本的框架：C++基本模板库（STL：string、vector、list、迭代器、标准输入输出流）、面向对象编程（类、构造函数、析构函数、友元、类的作用域、类继承、虚继承、抽象基类）、泛型编程（函数模板、类模板）、重载与引用、const、动态内存与智能指针（shared\_ptr、unique\_ptr、weak\_ptr）、文件I/O、命名空间、异常处理等等。由于C++的内容过于广泛，本学期的学习仅仅是打好了基础，在未来的日子里还需要多学习多思考，不断提升自己的能能力。