**实 验 报 告**

**课程名称：** 面向对象技术(C++)

**实验项目：** 继承与派生

**实验仪器：** 计算机

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **项目** | **报告格式**  **Report format** | **代码质量**  **Code quality** | **注释质量**  **Comment quality** | **逻辑或思想描述**  **Necessitate logical description** | **独创性**  **Originality** | **合计**  **Total** |
| **百分比(%)**  **percentage** | **10** | **25** | **25** | **25** | **15** | **100** |
| 得分（score） |  |  |  |  |  |  |

**系 别：** 计算机系

**专 业：** 计算机大类

**班级姓名：** 计类1805刘科宏

**学 号：** 2018011252

**日 期：** 4月23日

**成 绩：**

**同组成员：**

**指导教师：** 丁 濛

# 实验目的(Objects)

1. 了解继承与派生的概念，学习派生类的声明和定义方法，掌握派生类和基类构造函数和析构函数的调用顺序；

Understand what is “Class Inheritance” and learn how to declare and define a derived class; Understand the calling order of the constructors and destroyers of a derived class and its base class;

1. 验证protected修饰的成员函数或成员变量的访问控制；

Verify the access control properties of members under keyword “protected”.

1. 学习和掌握如何访问派生类与父类重名的变量或函数；

Learn how to access parent’s members from derived class, which have the same name with ones defined in derived class.

1. 学习和掌握派生类和基类的相互转化以及转化的实质；

Learn how to convert a derived class to its base class, and understand the essentials of the conversion.

1. 了解多重派生与多基类派生的概念，了解虚继承的概念；

Understand the concepts of multiple inheritance and virtual inheritance.

# 实验内容(Contents)

1. 以Person类为基类，声明派生类 National\_Person，该类有一个保护型成员变量 std::string Nationality；定义该派生类的构造函数；定义该类的公共接口，void setNationality(const std::string &)实现指定国籍的功能，void printNationality()实现打印国籍的功能，void printProperty()实现打印所有基类和派生类属性的功能;

Define a class *National\_Person* which inherits from class Person defined in Experiment 1. This *National\_Person* class has a protected member *std::string Nationality*.

Define its constructors.

Define a public member function *void* *setNationality(const std::string &)* to set *Nationality.*

Define a public member function *void printNationality()* to display *Nationality* and *void printProperty()* to display all information it has, including those from the base class.

1. 以Person类为基类，声明派生类 Student，该类有三个保护型成员变量 std::string schoolName, std::string studentID, int grade；定义该派生类的构造函数；定义该类的公共接口，分别实现指定/打印上述三个属性的功能，void printProperty()实现打印所有基类和派生类属性的功能;

Define a class *Student* which inherits from class Person defined in Experiment 1. This *Student* class has protected members: *std::string schoolName, std::string studentID, int grade*.

Define its constructors.

Define a public member function to set and display its three members *schoolName, studentID and grade.*

Define a public member function *void printProperty()* to display all information it has, including those from the base class.

1. 编写main函数，以自己的信息分别实例化上述派生类；利用基类的setName函数修改基类属性；验证对于基类的private类型变量，派生类能否访问；验证对于基类的protected类型变量，派生类是否可以访问；声明一个基类的对象，用上述派生类对其进行初始化，改变该基类对象的属性值，查看用来初始化该基类的派生类对象是否有变换；声明一个基类指针，指向上述派生类，验证该指针能访问的成员变量及函数的归属性；

Write a main function to test the above derived classes. Initialize those objects with your own information.

Use *setName* method in base-class to change *name.*

Verify whether you can access *private/protected* members of base-class in derived-class.

Define an object of class Person, and initialize this object using an existing object of either Student or National\_Person; Change its value, and see whether this change will apply to those existing objects of derived-class.

Define a pointer with type *Person\**, and let it point to the each exsiting objects, then check which class do members you can access using this pointer belong to.

1. 以National\_Person为基类，声明派生类 Chinese，自行设计并定义Chinese类的成员变量和成员函数；

Define a class Chinese (or named by your country) inheriting from *National\_Person;* Declare and define member variables and functions which you think are reasonable.

1. 利用已有的类，设计一个表示 中国学生 的派生类 Chinses\_Student。编写main函数，声明一个Chinses\_Student的对象，观察构造函数的调用过程；对于该对象，可否通过基类的setName函数改变姓名？如果可以，该如何调用，打印修改前和修改后的对象属性。

Design a new class to represent “a student with a nationality” making use of existing classes (Person, National\_Person, Student, Chinese).

Write a main function and define an object of the new designed class. Observe and verify the calling procedure of each related constructor. For such object, could you change the value of property *name* through *setName* method defined in base-class? If so, show your code, and print all information of that object before and after the call.

1. 用虚继承的方式 重做第5题。编写main函数，声明一个Chinses\_Student的对象，观察构造函数的调用过程；对于该对象，可否通过基类的setName函数改变姓名？如果可以，该如何调用，打印修改前和修改后的对象属性。

Redo 5 using virtual inheritance. Also write a main function and define an object of the new designed class. Observe and verify the calling procedure of each related constructor. For such object, could you change the value of property *name* through *setName* method defined in base-class? If so, show your code, and print all information of that object before and after the call.

# 实验内容 (Your steps or codes, and Results)

1.

#include <iostream>

#include <string>

#include <stdio.h>

using namespace std;

class Person{//Person基类,包含名字，性别，生日年月日

protected://以下成员为保护修饰，作用是可以被其派生类直接使用

string name;

char gender;

int year;

int month;

int day;

public:

Person() : name() ,gender('\0') ,year(1900),month(1),day(1){}

Person(const string& n,const char a,const int y,const int m,const int d) :name(n) ,gender(a) ,year(y),month(m),day(d)

{

printf("Person(const string&, char, int, int, int)\n");

}

Person(const Person& other):name(other.name) ,gender(other.gender) ,year(other.year),month(other.month),day(other.day)

{

printf("Person(const Person&)\n");

}

~Person()

{

printf("~Person()\n");

}

virtual void printProperty()//虚函数，因为在基类与派生类都含有这个同名函数，方便以后用指针指向时，可以使用相应类的函数

{

cout << "Name:" << name << '\n';

printf("Gender:%c\n",gender);

printf("Birthday:%d-%02d-%02d\n",year,month,day);

}

};

class National\_Person : public Person{//继承Person类的派生类

private:

string nat;

public:

National\_Person():nat(){}

National\_Person(const string& n, const string& na, char g, int y, int m, int d) : Person( n, g, y, m, d),nat(na)

{

cout << "National\_Person(const string&, const string&, char, int, int, int)\n";

}

National\_Person(const string& na) : nat(na){}

~National\_Person()

{

printf("~National\_Person()\n");

}

void printNationality()//输出国籍的函数

{

cout << "Nationality:" << nat << '\n' ;

}

void printProperty()//输出此人所有信息

{

Person::printProperty();

cout << "Nationality:" << nat << '\n' ;

}

};

int main(){

National\_Person lkh("LEO","CHN",'M', 1999, 10, 6);

lkh.printNationality();

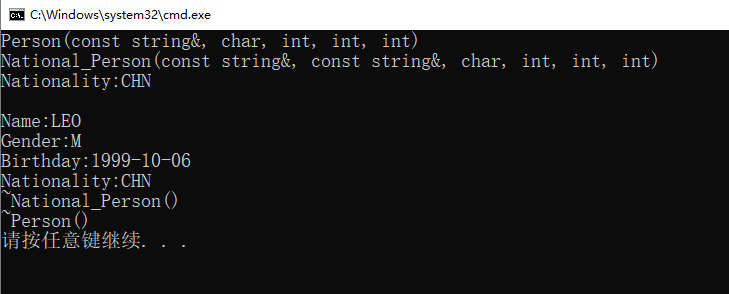
cout << '\n';

lkh.printProperty();

return 0 ;

}

执行结果：



图表 1：实验一执行结果

2.

#include <iostream>

#include <string>

#include <stdio.h>

using namespace std;

class Person{//Person基类,包含名字，性别，生日年月日

protected://以下成员为保护修饰，作用是可以被其派生类直接使用

string name;

char gender;

int year;

int month;

int day;

public:

Person() : name() ,gender('\0') ,year(1900),month(1),day(1){}

Person(const string& n,const char a,const int y,const int m,const int d) :name(n) ,gender(a) ,year(y),month(m),day(d)

{

printf("Person(const string&, char, int, int, int)\n");

}

Person(const Person& other):name(other.name) ,gender(other.gender) ,year(other.year),month(other.month),day(other.day)

{

printf("Person(const Person&)\n");

}

~Person()

{

printf("~Person()\n");

}

virtual void printProperty()//虚函数，因为在基类与派生类都含有这个同名函数，方便以后用指针指向时，可以使用相应类的函数

{

cout << "Name:" << name << '\n';

printf("Gender:%c\n",gender);

printf("Birthday:%d-%02d-%02d\n",year,month,day);

}

};

class Student : virtual public Person{

private:

string schoolName;

string studentID;

int grade;

public:

Student() : schoolName(),studentID(),grade(0){

cout << "Student()\n";

}

Student(const string& name, char g,int y,int m,int d,const string& sname,const string& sID,int \_grade) :

Person(name,g,y,m,d),schoolName(sname),studentID(sID),grade(\_grade){//Student是person的派生类，因此对于有参的构造函数，需要使用传入的值对Person类进行有参的构造，剩下的对Student自己的成员初始化

cout << "Student(const string&, char, int, int, int, const string&, const string&, int)\n";

}

~Student()

{

cout << "~Student()\n";

}

void setSchoolName(const string& N)//设置学校名的函数

{

schoolName = N;

}

void setStudentID(const string& ID)//设置学生ID的函数

{

studentID = ID;

}

void setGrade(int a)//设置成绩的函数

{

grade = a;

}

void printSchoolName()//输出学校名的函数

{

cout << schoolName << '\n';

}

void printStudentID()//输出学生的ID的函数

{

cout << studentID << '\n';

}

void printGrade()//输出成绩的函数

{

cout << grade << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "School Name:" << schoolName << '\n';

cout << "Student ID:" << studentID << '\n';

cout << "Grade:" << grade << '\n';

}

};

int main(){

Student lkh("LEO",'M', 1999, 10, 6,"BISTU","2018011252",100);

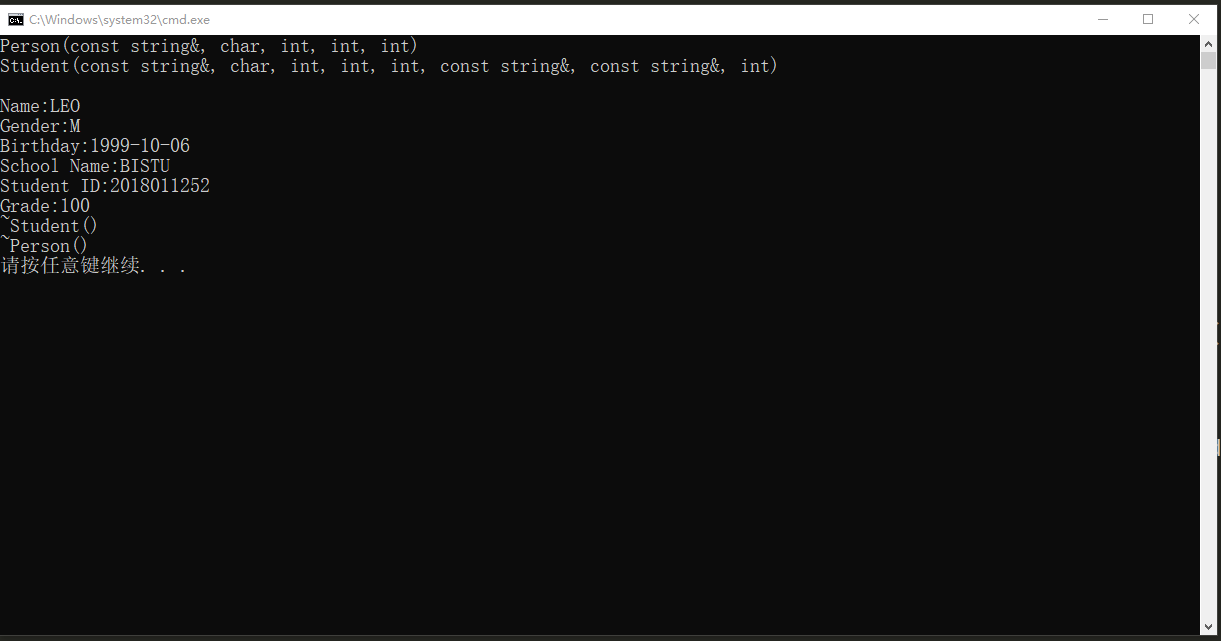
cout << '\n';

lkh.printProperty();

return 0 ;

}

执行结果



图表 2：实验二执行结果

3.

利用上一个实验的代码来实验，去掉了部分注释

#include <iostream>

#include <string>

#include <stdio.h>

using namespace std;

class Person{//Person基类,包含名字，性别，生日年月日

protected:

string name;

char gender;

int year;

int month;

int day;

public:

Person() : name() ,gender('\0') ,year(1900),month(1),day(1){}

Person(const string& n,const char a,const int y,const int m,const int d) :name(n) ,gender(a) ,year(y),month(m),day(d)

{

printf("Person(const string&, char, int, int, int)\n");

}

Person(const Person& other):name(other.name) ,gender(other.gender) ,year(other.year),month(other.month),day(other.day)

{

printf("Person(const Person&)\n");

}

~Person()

{

printf("~Person()\n");

}

void setName(const string& n)

{

name = n;

}

virtual void printProperty()//虚函数，因为在基类与派生类都含有这个同名函数，方便以后用指针指向时，可以使用相应类的函数

{

cout << "Name:" << name << '\n';

printf("Gender:%c\n",gender);

printf("Birthday:%d-%02d-%02d\n",year,month,day);

}

};

class Student : virtual public Person{

private:

string schoolName;

string studentID;

int grade;

public:

Student() : schoolName(),studentID(),grade(0){

cout << "Student()\n";

}

Student(const string& name, char g,int y,int m,int d,const string& sname,const string& sID,int \_grade) :

Person(name,g,y,m,d),schoolName(sname),studentID(sID),grade(\_grade){//Student是person的派生类，因此对于有参的构造函数，需要使用传入的值对Person类进行有参的构造，剩下的对Student自己的成员初始化

cout << "Student(const string&, char, int, int, int, const string&, const string&, int)\n";

}

~Student()

{

cout << "~Student()\n";

}

void foo()

{

year = 0;

}

void setSchoolName(const string& N)//设置学校名的函数

{

schoolName = N;

}

void setStudentID(const string& ID)//设置学生ID的函数

{

studentID = ID;

}

void setGrade(int a)//设置成绩的函数

{

grade = a;

}

void printSchoolName()//输出学校名的函数

{

cout << schoolName << '\n';

}

void printStudentID()//输出学生的ID的函数

{

cout << studentID << '\n';

}

void printGrade()//输出成绩的函数

{

cout << grade << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "School Name:" << schoolName << '\n';

cout << "Student ID:" << studentID << '\n';

cout << "Grade:" << grade << '\n';

}

};

int main(){

Student lkh("LEO",'M', 1999, 10, 6,"BISTU","2018011252",100);

cout << '\n';

lkh.printProperty();

lkh.setName("Leocaodou");

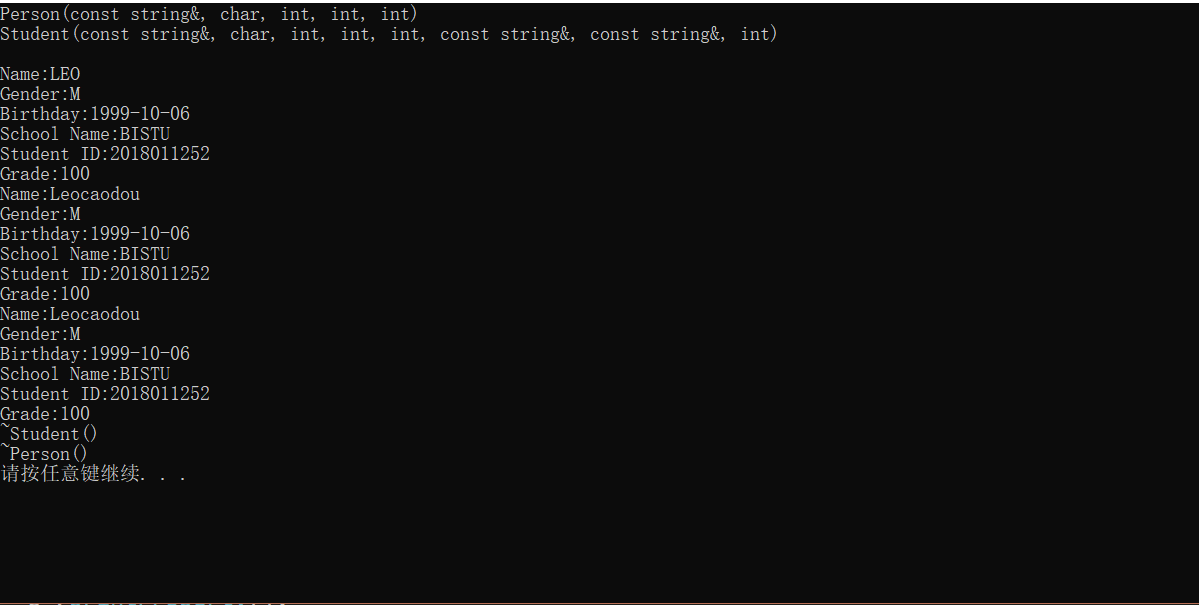
lkh.printProperty();

Person \*p = &lkh;//定义一个基类指针，指向了一个派生类对象

p->printProperty();

return 0 ;

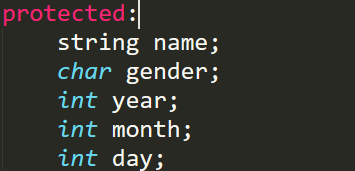
}

执行结果：

图表 3：实验三执行结果（1）

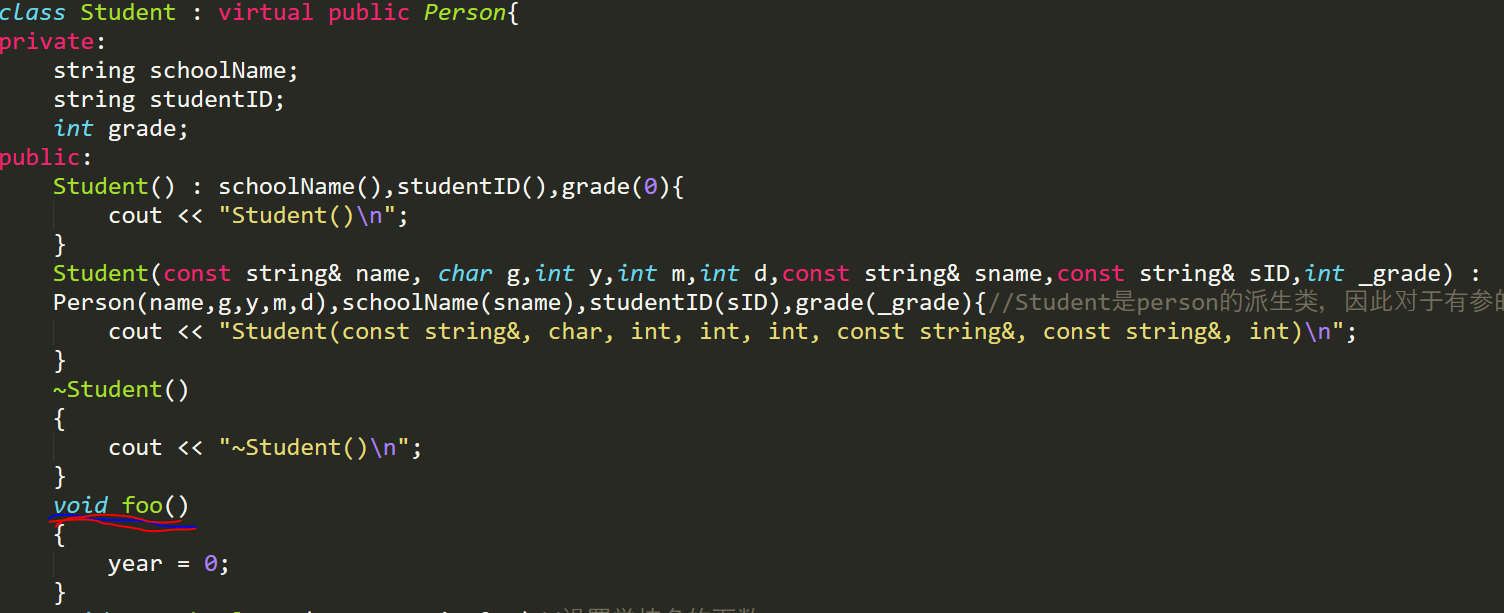
指针输出的函数包含了学校名称与学生ID，说明调用了派生类的printProperty()函数，这是由于加入了virtual修饰了此函数的结果。

当基类的成员为protected修饰时：



图表 4：实验三实验图（1）

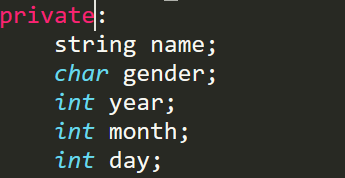
在派生类中加入一个foo()函数可以改变基类成员：



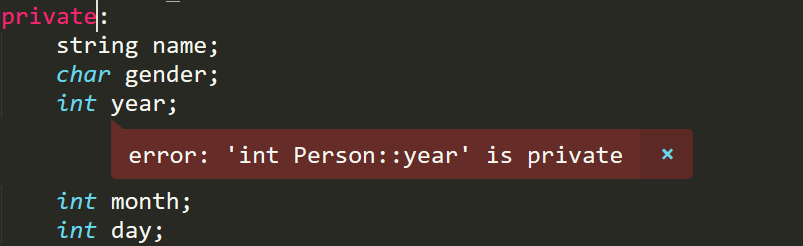
图表 5:实验三实验图（2）

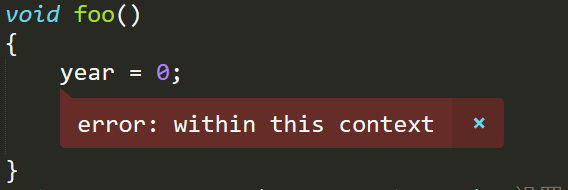
实验运行没有任何报错

若改为private修饰基类成员：



实验运行报错：





说明了被private修饰的基类成员不能被派生类使用，而protected修饰的基类成员可以被派生类使用。

4.

#include <iostream>

#include <string>

#include <stdio.h>

using namespace std;

class Person{

private:

string name;

char gender;

int year;

int month;

int day;

public:

Person() : name() ,gender('\0') ,year(1900),month(1),day(1){}

Person(const string& n,const char a,const int y,const int m,const int d) :name(n) ,gender(a) ,year(y),month(m),day(d)

{

printf("Person(const string&, char, int, int, int)\n");

}

Person(const Person& other):name(other.name) ,gender(other.gender) ,year(other.year),month(other.month),day(other.day)

{

printf("Person(const Person&)\n");

}

~Person()

{

printf("~Person()\n");

}

virtual void printProperty()

{

cout << "Name:" << name << '\n';

printf("Gender:%c\n",gender);

printf("Birthday:%d-%02d-%02d\n",year,month,day);

}

};

class National\_Person : public Person{

private:

string Nationality;

public:

National\_Person() : Nationality() {

cout << "National\_Person()\n";

}

National\_Person(const string& n, const string& na, char g, int y, int m, int d) : Person( n, g, y, m, d),Nationality(na)

{

cout << "National\_Person(const string&, const string&, char, int, int, int)\n";

}

National\_Person(const string& na) : Nationality(na){}

~National\_Person()

{

printf("~National\_Person()\n");

}

void setNationality(const string& nat)

{

Nationality = nat;

}

void printNationality()

{

Person::printProperty();

cout << "Nationality:" << Nationality << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "Nationality:" << Nationality << '\n';

}

};

class Chinese : public National\_Person{//创建Chinese类，派生与National\_Person

public://无新增加的私有成员

public:

Chinese() : National\_Person("CHN"){}//无参构造Chinese时，把国籍改为“CHN”

Chinese(const string& n, char g, int y, int m, int d) :National\_Person(n,"CHN",g,y,m,d){//有参构造时把，用传入的参数与国籍“CHN”来构造National\_Person

printf("Chinese(const string&, char, int, int, int)\n");

}

~Chinese()

{

cout << "~Chinese()\n";

}

void printProperty()

{

National\_Person::printProperty();

}

};

int main() {

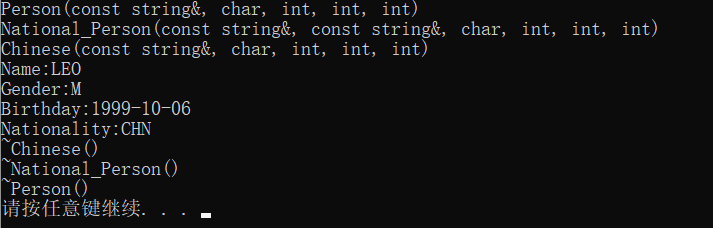
Chinese chnlkh("LEO", 'M', 1999, 10, 6);

chnlkh.printProperty();

return 0;

}

执行结果：



图表 6：实验四执行结果

5.

#include <iostream>

#include <string>

#include <stdio.h>

using namespace std;

class Person{

private:

string name;

char gender;

int year;

int month;

int day;

public:

Person() : name() ,gender('\0') ,year(1900),month(1),day(1){

cout << "Person()\n";

}

Person(const string& n,const char a,const int y,const int m,const int d) :name(n) ,gender(a) ,year(y),month(m),day(d)

{

printf("Person(const string&, char, int, int, int)\n");

}

Person(const Person& other):name(other.name) ,gender(other.gender) ,year(other.year),month(other.month),day(other.day)

{

printf("Person(const Person&)\n");

}

~Person()

{

printf("~Person()\n");

}

void setName(const string& n)//在基类中设置改名的函数

{

name = n;

}

virtual void printProperty()

{

cout << "Name:" << name << '\n';

printf("Gender:%c\n",gender);

printf("Birthday:%d-%02d-%02d\n",year,month,day);

}

};

class National\_Person : public Person{

protected:

string Nationality;

public:

National\_Person() : Nationality() {

cout << "National\_Person()\n";

}

National\_Person(const string& n, const string& na, char g, int y, int m, int d) : Person( n, g, y, m, d),Nationality(na)

{

cout << "National\_Person(const string&, const string&, char, int, int, int)\n";

}

National\_Person(const string& na) : Nationality(na){}

~National\_Person()

{

printf("~National\_Person()\n");

}

void setNationality(const string& nat)

{

Nationality = nat;

}

void printNationality()

{

cout << "Nationality:" << Nationality << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "Nationality:" << Nationality << '\n';

}

};

class Chinese : public National\_Person{

public:

public:

Chinese() : National\_Person("CHN"){}

Chinese(const string& n, char g, int y, int m, int d) :National\_Person(n,"CHN",g,y,m,d){

printf("Chinese(const string&, char, int, int, int)\n");

}

~Chinese()

{

cout << "~Chinese()\n";

}

void printProperty()

{

National\_Person::printProperty();

}

};

class Student : public Person{

private:

string schoolName;

string studentID;

int grade;

public:

Student() : schoolName(),studentID(),grade(0){

cout << "Student()\n";

}

Student(const string& name, char g,int y,int m,int d,const string& sname,const string& sID,int \_grade) :

Person(name,g,y,m,d),schoolName(sname),studentID(sID),grade(\_grade){

cout << "Student(const string&, char, int, int, int, const string&, const string&, int)\n";

}

~Student()

{

cout << "~Student()\n";

}

void setSchoolName(const string& N)

{

schoolName = N;

}

void setStudentID(const string& ID)

{

studentID = ID;

}

void setGrade(int a)

{

grade = a;

}

void printSchoolName()

{

cout << schoolName << '\n';

}

void printStudentID()

{

cout << studentID << '\n';

}

void printGrade()

{

cout << grade << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "School Name:" << schoolName << '\n';

cout << "Student ID:" << studentID << '\n';

cout << "Grade:" << grade << '\n';

}

};

class Chinese\_Student : public Student,public Chinese{

private:

public:

Chinese\_Student() {

cout << "Chinese\_Student()\n";

}

Chinese\_Student(const string& name, char g,int y,int m,int d,const string& sname,const string& sID,int \_grade) :

Student( name, g, y, m, d, sname, sID, \_grade),Chinese(name, g, y, m, d){//由于基类的Person和National\_Person为虚继承，不会构造出Person类，所以需要在这里再次构造

cout << "Chinese\_Student(const string&, char, int, int, int, const string&, const string&, int)\n";

}

~Chinese\_Student()

{

cout << "~Chinese\_Student()\n";

}

void printProperty(){

Student::printProperty();//输出函数调用先的是Student的输出函数，所以这里输出的Person为Student构造出来的，所以下面主函数中的setName函数就可以只需要调用Student的就成功

cout << "Nationality:" << Nationality << '\n';

}

};

int main() {

Chinese\_Student chnStulkh("LEO", 'M', 1999, 10, 6, "BISTU", "2018011252", 100);

chnStulkh.printProperty();

chnStulkh.Student::setName("Leocaodou");//直接调用基类Student类中的setName函数

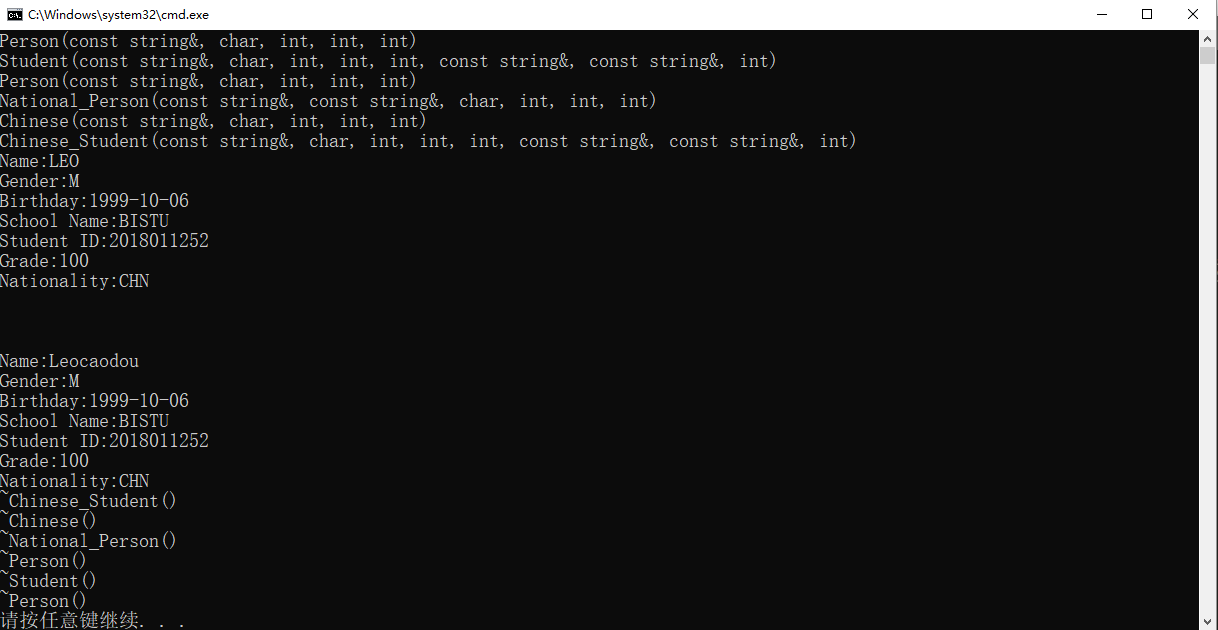
cout << "\n\n\n";

chnStulkh.printProperty();

return 0;

}

执行结果：



图表 7：实验五执行结果

如果这道题这么做的话，还有Natinal\_Person的name没有改变，会有出错的风险。

6.

#include <iostream>

#include <string>

#include <stdio.h>

using namespace std;

class Person{

private:

string name;

char gender;

int year;

int month;

int day;

public:

Person() : name() ,gender('\0') ,year(1900),month(1),day(1){

cout << "Person()\n";

}

Person(const string& n,const char a,const int y,const int m,const int d) :name(n) ,gender(a) ,year(y),month(m),day(d)

{

printf("Person(const string&, char, int, int, int)\n");

}

Person(const Person& other):name(other.name) ,gender(other.gender) ,year(other.year),month(other.month),day(other.day)

{

printf("Person(const Person&)\n");

}

~Person()

{

printf("~Person()\n");

}

void setName(const string& n)//在基类中设置改名的函数

{

name = n;

}

virtual void printProperty()

{

cout << "Name:" << name << '\n';

printf("Gender:%c\n",gender);

printf("Birthday:%d-%02d-%02d\n",year,month,day);

}

};

class National\_Person : virtual public Person{//虚继承Person类，为了防止在构造Chinese\_Student时构造出两个Person，无法达到修改名字的目的

protected:

string Nationality;

public:

National\_Person() : Nationality() {

cout << "National\_Person()\n";

}

National\_Person(const string& n, const string& na, char g, int y, int m, int d) : Person( n, g, y, m, d),Nationality(na)

{

cout << "National\_Person(const string&, const string&, char, int, int, int)\n";

}

National\_Person(const string& na) : Nationality(na){}

~National\_Person()

{

printf("~National\_Person()\n");

}

void setNationality(const string& nat)

{

Nationality = nat;

}

void printNationality()

{

cout << "Nationality:" << Nationality << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "Nationality:" << Nationality << '\n';

}

};

class Chinese : public National\_Person{//又National\_Person到Chinese\_Student继承是单线的，所以不会出现重复构造的情况，所以这里不需要加virtual

public:

public:

Chinese() : National\_Person("CHN"){}

Chinese(const string& n, char g, int y, int m, int d) :National\_Person(n,"CHN",g,y,m,d){

printf("Chinese(const string&, char, int, int, int)\n");

}

~Chinese()

{

cout << "~Chinese()\n";

}

void printProperty()

{

National\_Person::printProperty();

}

};

class Student : virtual public Person{//虚继承，理由同National\_Person

private:

string schoolName;

string studentID;

int grade;

public:

Student() : schoolName(),studentID(),grade(0){

cout << "Student()\n";

}

Student(const string& name, char g,int y,int m,int d,const string& sname,const string& sID,int \_grade) :

Person(name,g,y,m,d),schoolName(sname),studentID(sID),grade(\_grade){

cout << "Student(const string&, char, int, int, int, const string&, const string&, int)\n";

}

~Student()

{

cout << "~Student()\n";

}

void setSchoolName(const string& N)

{

schoolName = N;

}

void setStudentID(const string& ID)

{

studentID = ID;

}

void setGrade(int a)

{

grade = a;

}

void printSchoolName()

{

cout << schoolName << '\n';

}

void printStudentID()

{

cout << studentID << '\n';

}

void printGrade()

{

cout << grade << '\n';

}

void printProperty()

{

Person::printProperty();

cout << "School Name:" << schoolName << '\n';

cout << "Student ID:" << studentID << '\n';

cout << "Grade:" << grade << '\n';

}

};

class Chinese\_Student : public Student,public Chinese{

private:

public:

Chinese\_Student() {

cout << "Chinese\_Student()\n";

}

Chinese\_Student(const string& name, char g,int y,int m,int d,const string& sname,const string& sID,int \_grade) :

Person(name,g,y,m,d),Student( name, g, y, m, d, sname, sID, \_grade),Chinese(name, g, y, m, d){//由于基类的Person和National\_Person为虚继承，不会构造出Person类，所以需要在这里再次构造

cout << "Chinese\_Student(const string&, char, int, int, int, const string&, const string&, int)\n";

}

~Chinese\_Student()

{

cout << "~Chinese\_Student()\n";

}

void printProperty(){

Student::printProperty();

cout << "Nationality:" << Nationality << '\n';

}

};

int main() {

Chinese\_Student chnStulkh("LEO", 'M', 1999, 10, 6, "BISTU", "2018011252", 100);

chnStulkh.printProperty();

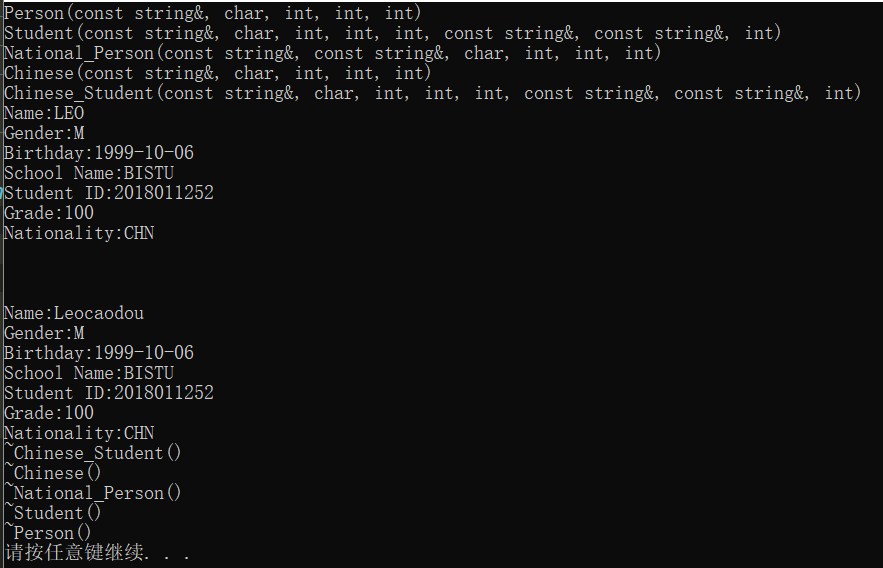
chnStulkh.setName("Leocaodou");//直接调用基类Person类中的setName函数

cout << "\n\n\n";

chnStulkh.printProperty();

return 0;

}

执行结果：

图表 8：实验六执行结果

若不使用虚继承的话，程序报错：

图表 9：实验六程序报错

原因是不是虚继承的话，将会构造出两个Person，电脑无法判断这里的setName函数究竟是哪一个Person的。

# 实验总结 (Conclusion)

这次实验当中学到了完全在c当中从未听说的内容，那就是类的继承，这其中的知识点错综复杂，需要花时间来慢慢消化与理解。

（1）首先继承分为了私有继承(private)，保护继承(protected)，还有共有继承(public),对于这三个，在面对不同的问题时，我们的选取会有所不同。

图表 10：总结(1)

（2）要学会使用虚拟函数，当基类与派生类都有用一个命名，作用相似的函数时，可以使用虚拟函数来避免不少问题，用基类指针来指向不同派生类时，这样就可以调用相应类当中的这个函数，有时可以节约很大的工程。

（3）有纯虚函数，用于方便使用派生类的同名函数，但有纯虚函数的类无法实体化。

（4）对于有分叉而最后又结合的继承关系时可以使用虚继承，把最开始分叉的派生类虚拟继承基类，这样做的话，不会使每一个派生类都会重复构造一个自己的基类，但这样需要最后的自主构造基类来为基类的成员赋值。

（5）继承关系有时可以通过画图来使内容更加清晰。