**实验6 类的继承与派生**

**一、实验目的**

（1）掌握继承和派生的定义，派生类的定义方法和继承方式。

（2）掌握派生类中的构造函数的使用。

（3）掌握多重继承的概念、定义方法、多重继承派生类构造函数的执行顺序。

（4）掌握虚基类的概念和定义方法。

（5）掌握同名覆盖的用法，理解作用域的概念

**二、实验内容**

**内容1：**

#include<iostream>

#include<string>

using namespace std;

//先定义一个基类

class person

{

    public:

    person(){cout<<"基类构造函数"<<endl;}

    person(string n,string s,int a)

    {

        name=n;

        sex=s;

        age=a;

        cout<<"基类构造函数"<<endl;

    }

    ~person(){cout<<"基类析构函数"<<endl;}

    protected:

    string name,sex;

    int age;

};

//定义教师类

class Teacher: virtual public person

{

    public:

    Teacher(){cout<<"教师类构造函数"<<endl;}

    Teacher(string n,string s,int a,int n1,string t,int w):person(n,s,a),

            w\_number(n1),title(t),wages(w){cout<<"教师类构造函数"<<endl;}

    ~Teacher(){cout<<"教师类析构函数"<<endl;}

    protected:

    int w\_number;

    private://将教师职称和工资定义为private成员

    int wages;

    string title;

};

//定义学生类

class Student: virtual public person

{

    public:

    Student(){cout<<"学生类构造函数"<<endl;}

    Student(string n,string s,int a,int n2,string c,string m,double sc):person(n,s,a),

            s\_number(n2),clas(c),major(m),score(sc){cout<<"学生类构造函数"<<endl;}

    ~Student(){cout<<"学生类析构函数"<<endl;};

    protected:

    int s\_number;

    string clas,major;

    double score;

};

//定义在职研究生类

class G\_student:public Teacher,public Student

{

    public:

    G\_student(){cout<<"在职研究生类构造函数"<<endl;}

    G\_student(string n,string s,int a,int n1,string t,int w,int n2,string c,string m,double sc,string o):

              person(n,s,a),Teacher(n,s,a,n1,t,w),Student(n,s,a,n2,c,m,sc),of(o){cout<<"在职研究生类构造函数"<<endl;}

    ~G\_student(){cout<<"在职研究生类析构函数"<<endl;}

    //下面流运算符重载

    friend std::istream& operator >>(std::istream&,G\_student&);

    friend std::ostream& operator <<(std::ostream&,G\_student&);

    protected:

    string of;

};

//定义在读教师类

class S\_teacher:public Teacher,public Student

{

    public:

    S\_teacher(string n,string s,int a,int n1,string t,int w,int n2,string c,string m,double sc):

             person(n,s,a),Teacher(n,s,a,n1,t,w),Student(n,s,a,n2,c,m,sc){cout<<"在读教师类构造函数"<<endl;}

    ~S\_teacher(){cout<<"在读教师类析构函数"<<endl;}

};

//下面对成员及运算符重载函数定义

istream& operator >>(std::istream& input,G\_student& g)

{

    cout<<"姓名：";

    input>>g.name;

    cout<<"性别：";

    input>>g.sex;

    cout<<"年龄：";

    input>>g.age;

    cout<<"工号：";

    input>>g.w\_number;

    cout<<"学号：";

    input>>g.s\_number;

    cout<<"班级：";

    input>>g.clas;

    cout<<"专业：";

    input>>g.major;

    cout<<"入学成绩：";

    input>>g.score;

    return input;

}

ostream& operator <<(std::ostream& output,G\_student& g)

{

    output<<"("<<"姓名："<<g.name<<"  性别："<<g.sex<<"  年龄："<<g.age

    <<"  工号："<<g.w\_number<<"  学号："<<g.s\_number<<"  班级："<<g.clas

    <<"  专业："<<g.major<<"  入学成绩："<<g.score<<")";

    return output;

}

main()

{

    G\_student G;

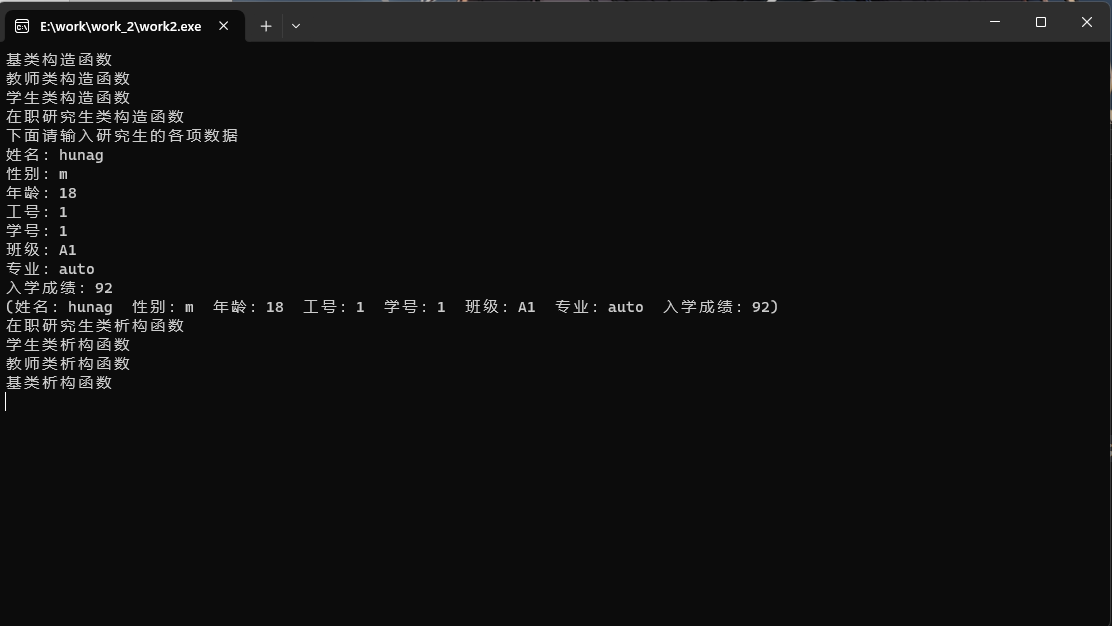
    cout<<"下面请输入研究生的各项数据"<<endl;

    cin>>G;

    cout<<G<<endl;

    return 0;

}



**分析：**

1. 虚基类的作用：

用于解决多重继承中的重名问题，即二义性问题，提供基类限定符和同名覆盖机制之外的另一种方式。

1. 构造函数调用顺序：
2. 先调用基类构造函数再调用派生类构造函数
3. 先调用虚基类构造函数再调用非基类构造函数
4. 若同层中包括多个虚基类，则按声明顺序调用

构造函数调用顺序，与构造函数的调用顺序完全相反。

**内容2：**

#include <iostream>

#include <string>

using namespace std;

namespace A

{

    int g\_i = 0;

}

namespace B

{

    int g\_i = 1;

}

class Parent

{

public:

    int mi;

    Parent()

    {

        cout<<"Parent中mi的地址："<<&mi<<endl;// 补充语句使其输出成员mi的地址

    }

};

class Child : public Parent

{

public:

    int mi;

    Child()

    {

        cout<<"Child中mi的地址："<<&mi<<endl;             //补充语句使其输出成员mi的地址

    }

};

int main()

{

    Child c;

    c.mi =10;//给对象c中新增成员mi赋值为10

    c.Parent::mi=100;// 给对象c中继承的成员mi赋值为100

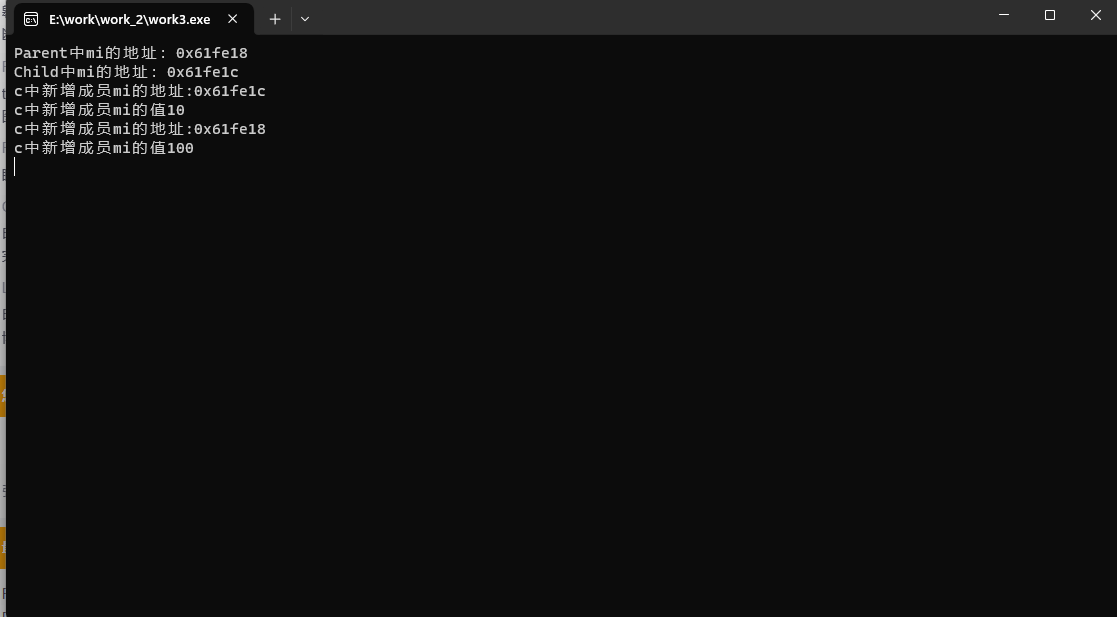
    cout<<"c中新增成员mi的地址:"<<&c.mi<<endl;//输出对象c中新增成员mi的地址

    cout<<"c中新增成员mi的值"<<c.mi<<endl;// 输出对象c中新增成员mi的

    cout<<"c中新增成员mi的地址:"<<&c.Parent::mi<<endl;// 输出对象c中继承的成员mi的地址

    cout<<"c中新增成员mi的值"<<c.Parent::mi<<endl; //输出对象c中继承的成员mi的值

    return 0;

}

3）

#include <iostream>

#include <string>

using namespace std;

namespace A

{

    int g\_i = 0;

}

namespace B

{

    int g\_i = 1;

}

class Parent

{

public:

    int mi;

    Parent()

    {

        cout<<"Parent中mi的地址："<<&mi<<endl;// 补充语句使其输出成员mi的地址

    }

    void add(int n)

    {

        mi=n+mi;

    }

    void add(int m,int n)

    {

        mi=n+m+mi;

    }

};

class Child : public Parent

{

public:

    int mi;

    Child()

    {

        cout<<"Child中mi的地址："<<&mi<<endl;//补充语句使其输出成员mi的地址

    }

    void add(int x,int y,int z)

    {

        mi=x+y+z+mi;

    }

};

int main()

{

    Child c;

    c.mi =10;//给对象c中新增成员mi赋值为10

    c.Parent::mi=100;// 给对象c中继承的成员mi赋值为100

    cout<<"c中新增成员mi的地址:"<<&c.mi<<endl;//输出对象c中新增成员mi的地址

    cout<<"c中新增成员mi的值"<<c.mi<<endl;// 输出对象c中新增成员mi的

    cout<<"c中新增成员mi的地址:"<<&c.Parent::mi<<endl;// 输出对象c中继承的成员mi的地址

    cout<<"c中新增成员mi的值"<<c.Parent::mi<<endl; //输出对象c中继承的成员mi的值

    c.Parent::add(1);

    cout<<"调用add函数1后mi的值(基类mi加上一个数“1”)："<<c.Parent::mi<<endl;

    c.Parent::add(1,5);

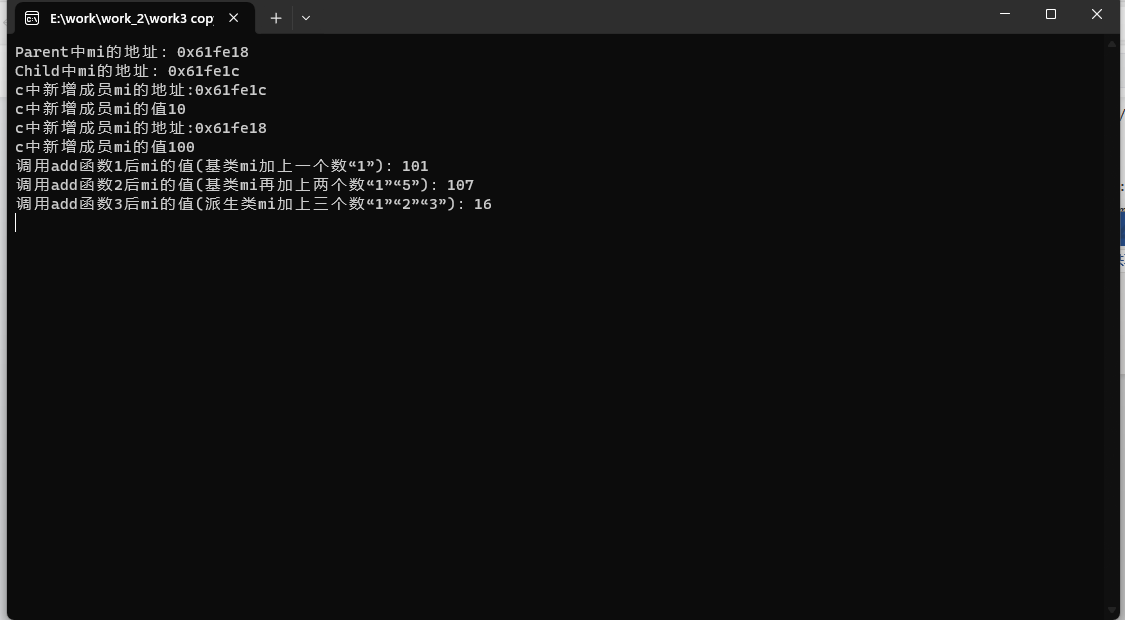
    cout<<"调用add函数2后mi的值(基类mi再加上两个数“1”“5”)："<<c.Parent::mi<<endl;

    c.add(1,2,3);

    cout<<"调用add函数3后mi的值(派生类mi加上三个数“1”“2”“3”)："<<c.mi<<endl;

    return 0;

}

****

**分析：**

1. 没有错误。应为两个同名变量的作用域不同。父类成员被子类覆盖，但是它依然存在子类中，可以加以域作用符区分，如此子类与父类就可以定义同名变量了。
2. 问题：如何区分派生类从基类中继承的同名数据成员？ 解决：使用域作用符加以区分。
3. 子类无法重载父类中的函数，只能进行重定义，定义上的重载必须再同个类中进行。