

# 实验十三 继承与多态

## 一、 问题描述

### 1. 实验目的:

掌握派生类的若干基本概念和特性,并能够应用于程序编写。

掌握验证性实验的基本方法和过程(认知、实验、总结)。

### 2. 实验内容:

分别编写一段测试代码来回答任务书中的相关问题(每一个问题,用一个工程文件,同时需要记录相应的调试过程),具体问题请参考“实验任务 说明12.doc”;

调试的过程:(动态调试的相关截图,比如 设置断点、查看当前变量值等);

编译出来的可执行程序单独放在一个目录下(bin/exe/debug目录下,同时附上输入数据说明和输出结果)

## 二、 实验过程

### 一、名词解释

#### 1、虚函数、虚基类

答:(1)在一个类中用关键字virtual 说明的成员函数称为虚函数。定义了虚函数的类称为多态类。虚函数原型语句格式: virtual 类型 函数名(参数表);在基类中某个成员函数被声明为虚函数后,这个成员函数通常要在派生类中被重新定义。定义一个虚函数的目的是为了在程序运行时自动选择各派生类中的重定义版本,所以一个多态的基类一定要定义一个以上的派生类才有意义。

(2)当在多条继承路径上有一个公共的基类,在这些路径中的某几条汇合处,这个公共的基类就会产生多个实例(或多个副本),若只想保存这个基类的一个实例,可以将这个公共基类说明为虚基类。通过把基类继承声明为虚拟的,就只能继承基类的一份拷贝,从而消除歧义。用virtual限定符把基类继承说明为虚拟的。

#### 2、纯虚函数、抽象类

答:(1)纯虚函数是在基类中只有说明而没有实现定义的虚函数,它的任何派生类都必须定义自己的实现版本。纯虚函数定义形式: virtual 类型 函数名(参数表)=0;

(2)抽象类是表示一组具有某些共性的具体类的公共特征的类,表示更高层次的抽象概念。不能创建抽象类的对象。

## 二、选择题

1 虚函数必须是类的 ( B )

A) 成员函数    B) 友元函数    C)构造函数    D)析构函数

2 多态调用 (C)

A) 以任何方式调用一个虚函数                      B) 以任何方式调用一个纯虚函数

C) 借助于指向对象的基类指针或引用调用一个虚函数

D) 借助于成员访问运算符. 来调用一个虚函数

## 三、简答题

1、请描述多继承中出现的二义性问题及解决措施

答：(1) 二义性问题：派生类与基类中存在名字相同的数据或函数，使得函数无法判断与调用。(2) 解决方法：1、在引用数据成员或成员函数时指明其作用域。(类名::数据名) 2、同名覆盖(函数重载，基类函数必须有 **virtual** 关键字)：派生类新增的同名成员将会覆盖基类中的同名成员，不论有多少个基类，实现的途径是利用派生类的新增成员，即要有同名的，则在直接调用不加类名作用域时就会调用派生类的成员。3、虚基类(虚拟继承)：让继承间接共同基类时只保留一份成员。

2、请描述重复继承中出现的二义性问题及解决措施

答：多重继承：一个类派生出多个类，多个类派生出一个类。当在多条继承路径上有一个公共的基类,在这些路径中的某几条汇合处，这个公共的基类就会产生多个实例(或多个副本)，若只想保存这个基类的一个实例，可以将这个公共基类说明为虚基类。通过把基类继承声明为虚拟的，就只能继承基类的一份拷贝，从而消除歧义。用 **virtual** 限定符把基类继承说明为虚拟的。

3、请描述虚基类、虚函数的功能

答：(1) 当派生类从多个基类派生，而这些基类又有共同基类，则在访问此共同基类中的成员时，将产生冗余，并有可能因冗余带来不一致性(二义性)。使用虚基类来解决多继承时可能发生的对同一基类继承多次而产生的二义性问题。(2) 定义一个虚函数的目的是为了在程序运行时自动选择各派生类中的重定义版本，所以一个多态的基类一定要定义一个以上的派生类才有意义。

4、请描述虚基类机制下的构造函数的执行顺序

答：(1) 虚基类的构造函数在非虚基类之前调用； (2) 若同一层次中包含多个虚基类，这些虚基类的构造函数按它们说明的次序调用； (3) 若虚基类由非虚基类派生而来，则仍先调用基类构造函数，再调用派生类的构造函数。

## 5、请描述虚函数动态绑定是如何实现的

答：动态绑定是通过虚函数表实现的。对于含有虚函数的多态类，编译器为每个对象生成一个虚表指针。即在每个对象的内存映像中增加了一个\_vfptr 指针，它指向一个虚函数表vtable。在基类的虚函数表中列出基类所有虚函数的入口地址，在派生类的虚函数表中列出派生类的所有虚函数的入口地址。当基类指针指向派生类的时候，实际上既可以获得派生类的虚表指针，通过虚表指针，基类既可以去调用派生类中的成员函数。如此不同的派生类的虚函数不同，调用的时候实现的效果也就不同，也就符合了多态的定义，面对同一消息的不同应答。

## 四、代码题

### 1、给出下列文字的代码表示：

程序中定义了类A，类A1和类A2都是在类A的基础上以公有继承方式产生的派生类，而类B是在类A1和类A2的基础上经过多重继承方式产生的派生类。

```
#include<iostream>
using namespace std;
class A {
public:
    A() {
        a = 5; cout << "A=" << a << endl;
    }
protected:
    int a;
};
class A1 :virtual public A {
public:
    A1() { a += 10; cout << "A1=" << a << endl; }
};
class A2 :virtual public A {
public:
    A2() { a += 20; cout << "A2=" << a << endl; }
};
class B :public A1, public A2 {
public:
    B() { cout << "B a =" << a << endl; }
};
int main() {
    B obj;
    return 0;
}
```

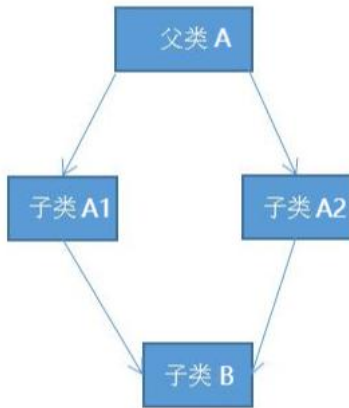
Microsoft Visual Studio 调试控制台

```
A=5
A1=15
A2=35
B a =35
```

思考：画图分析是否存在二义性问题？该如何解决？

答：存在二义性问题。派生类B从多个基类派生，而这些基类A1、A2又有共同基类A，则在访问此共同基类中的成员时，将产生冗余，并有可能因冗余带来不一致性（二

义性)。使用虚基类来解决多继承时可能发生的对同一基类继承多次而产生的二义性问题。即通过virtual定义虚基类从而避免二义性。



2、分别定义Teacher（教师类）和Cadre（干部类），采用多重继承方式由这两个类派生出新类Teacher\_Cadre（教师兼干部类）。要求：

- （1）在两个基类中都包含姓名、年龄、性别、地址、电话等数据成员；
- （2）在Teacher类中还包含数据成员，职称；在Cadre类中还包含数据成员，职务，在Teacher\_Cadre类中还包含数据成员，工资；
- （3）对两个基类中的姓名、年龄、性别、地址、电话等数据成员用相同的名字，在引用这些数据成员时，指定作用域；
- （4）在类体中声明成员函数，在类外定义成员函数（使用多文件编程！）
- （5）在派生类Teacher\_Cadre的成员函数show中调用Teacher类中的display函数，输出姓名、年龄、性别、职称、地址、电话，然后用cout语句输出职务和工资。

答：(1) 头文件

```

#include <iostream>
#include<string>
using namespace std;
//在类体中声明成员函数，在类外定义成员函数

class Teacher
{
public:
    Teacher() {}
    Teacher(string nam, int a, char s, string tit, string ad, string t);
    void display();
protected:
    //在两个基类中都包含姓名、年龄、性别、地址、电话等数据成员；
    //在Teacher类中还包含数据成员，职称；
    string name;
    int age;
    char sex;
    string title;
    string addr;
    string tel;
};

class Cadre
{
public:
    Cadre() {}
    Cadre(string nam, int a, char s, string p, string ad, string t);
    void display();
protected:
    //在两个基类中都包含姓名、年龄、性别、地址、电话等数据成员；
    //在Cadre类中还包含数据成员，职务，
    string name;
    int age;
    char sex;
    string post;
    string addr;
    string tel;
};

class Teacher_Cadre :public Teacher, public Cadre
{
private:
    //在Teacher_Cadre类中还包含数据成员，工资
    int wages;
public:
    Teacher_Cadre() {}
    Teacher_Cadre(string nam, int ag, char se, string tit, string pos, string add, string te, int wag);
    void Show();
};

```

## (2) cpp文件

```

#include "4.2.h"
Teacher::Teacher(string nam, int a, char s, string tit, string ad, string t) :
    name(nam), age(a), sex(s), title(tit), addr(ad), tel(t) {
    cout << "调用Teacher构造函数" << endl;
}

void Teacher::display()
{
    cout << "姓名: " << name << endl;
    cout << "年龄: " << age << endl;
    cout << "性别: " << sex << endl;
    cout << "职称: " << title << endl;
    cout << "地址: " << addr << endl;
    cout << "电话: " << tel << endl;
}

Cadre::Cadre(string nam, int a, char s, string p, string ad, string t) :
    name(nam), age(a), sex(s), post(p), addr(ad), tel(t) {
    cout << "调用Cadre的构造函数" << endl;
}

void Cadre::display()
{
    cout << "姓名: " << name << endl;
    cout << "年龄: " << age << endl;
    cout << "性别: " << sex << endl;
    cout << "职务: " << post << endl;
    cout << "地址: " << addr << endl;
    cout << "电话: " << tel << endl;
}


Teacher_Cadre::Teacher_Cadre(string nam, int ag, char se, string tit, string pos, string add, string te, int wag) :
    Teacher(nam, ag, se, tit, add, te),
    Cadre(nam, ag, se, pos, add, te),
    wages(wag) {
    cout << "调用Teacher_Cadre的构造函数" << endl;
}

void Teacher_Cadre::Show() {
    //在派生类Teacher_Cadre的成员函数show中调用Teacher类中的display函数
    //输出姓名、年龄、性别、职称、地址、电话，然后用cout语句输出职务和工资。
    //对两个基类中的姓名、年龄、性别、地址、电话等数据成员用相同的名字
    //在引用这些数据成员时，指定作用域;
    Teacher::display();
    cout << "职务: " << Cadre::post << endl;
    cout << "工资: " << wages << endl;
}

int main()
{
    Teacher_Cadre te_ca("Wangli", 50, 'f', "prof.", "president", "135 Beijing Road, Shanghai", "(021)61234567", 1534.5);
    te_ca.Show();
    return 0;
}

```

### (3) 实验结果

 Microsoft Visual Studio 调试控制台

```

调用Teacher构造函数
调用Cadre的构造函数
调用Teacher_Cadre的构造函数
姓名: Wangli
年龄: 50
性别: f
职称: prof.
地址: 135 Beijing Road, Shanghai
电话: (021)61234567
职务: president
工资: 1534

```

### 3、编写程序，计算圆形、三角形、正方形和长方形四种图形的面积

提示，定义抽象类shape，在其中说明一个纯虚函数area()作为接口。在派生类中定义具体的函数实现。

```
#include<iostream>
using namespace std;
//定义抽象类shape，在其中说明一个纯虚函数area()作为接口。在派生类中定义具体的函数实现。

class Shape // 抽象基类
{
public:
    virtual void printArea() = 0; // 纯虚函数
};

class Circle :public Shape // 定义 Circle 类
{
public:
    Circle(float r) :radius(r) {}; // 定义构造函数
    virtual void printArea() // 对虚函数再定义
    {
        cout << "Area of Circle: " << endl << 3.14159 * radius * radius << endl;
    }
private:
    float radius;
};

class Rectangle :public Shape // 定义 Rectangle 类
{
public:
    Rectangle(float w, float h) :width(w), height(h) {}; // 定义构造函数
    virtual void printArea() // 对虚函数再定义
    {
        cout << "Area of Rectangle: " << endl << width * height << endl;
    }
private:
    float width;
    float height;
};

class Square:public Shape // 定义 Square 类
{
public:
    Square(float w) :length(w) {}; // 定义构造函数
    virtual void printArea() // 对虚函数再定义
    {
        cout << "Area of Square: " << endl << length * length << endl;
    }
private:
    float length;
};
```

Microsoft Visual Studio 调试控制台

```
Area of Circle:
63.6172
Area of Rectangle:
12
Area of Triangle:
7.5
Area of Square:
9
```

4、设计一个抽象类vehicle，由它派生出类car和类truck。类car包含名称、颜色、载客数等3个数据成员，类truck包含名称、颜色、载重量3个数据成员。纯虚函数用以输出数据成员（对>>和<<使用运算符重载）。

备注：上述3和4两道题目与虚函数相关，代码编写是一方面，关键是要思考和总结，“一个接口，多个实现”的意义

```
#include <iostream>
#include <string>
using namespace std;

class Vehicle {
protected:
    string na;//名字
    string co;//颜色
};

class Car :public Vehicle {
protected:
    string na;
    string co;
    int pa;
public:
    Car() {
        na = "";
        co = "";
        pa = 0;
    }
    Car(string name, string color, int pas) {
        na = name;
        co = color;
        pa = pas;
    }

    //只能将重载“<<”和“>>”的函数作为友元函数或普通函数，而不能将它们定义为成员函数
    friend ostream& operator <<(ostream& strm, Car& car);
    friend istream& operator >>(istream& strm, Car& car);
};
```



```

- ostream& operator <<(ostream& strm, Car& car) {
    strm << "Car name:" << car.na << " Car color:" << car.co << " Car passager:" <<
        car.pa << endl;
    return strm;
}

- istream& operator >>(istream& strm, Car& car) {
    cout << "Car name:" << endl;
    strm >> car.na;
    cout << "Car color:" << endl;
    strm >> car.co;
    cout << "Car passager:" << endl;
    strm >> car.pa;
    return strm;
}

```

```

- class Truck :public Vehicle {
protected:
    string na;
    string co;
    double ca;

public:
    Truck(string name, string color, double cap) {
        na = name;
        co = color;
        ca = cap;
    }
    Truck() {
        na = "";
        co = "";
        ca = 0;
    }
    friend ostream& operator <<(ostream&, Truck&);
    friend istream& operator >>(istream& strm, Truck& truck);
};

```

//只能将重载 "<<" 和 ">>" 的函数作为友元函数或普通函数，而不能将它们定义为成员函数

```

- ostream& operator <<(ostream& strm, Truck& truck) {
    strm << "Truck name:" << truck.na << " Truck color:" << truck.co << " Truck capacity:" << truck.ca << endl;
    return strm;
}


- istream& operator >>(istream& strm, Truck& truck) {
    cout << "Truck name:" << endl;
    strm >> truck.na;
    cout << "Truck color:" << endl;
    strm >> truck.co;
    cout << "Truck capacity:" << endl;
    strm >> truck.ca;
    return strm;
}

```

```

int main() {
    cout << "请输入Car的数据: " << endl;
    Car car;
    cin >> car;
    cout << car << endl;
    cout << "请输入Truck的数据: " << endl;
    Truck truck;
    cin >> truck;
    cout << truck << endl;
    return 0;
}

```

 Microsoft Visual Studio 调试控制台

```

请输入Car的数据:
Car name:
Benz
Car color:
black
Car passager:
3
Car name: Benz Car color: black Car passager: 3

请输入Truck的数据:
Truck name:
Dongfeng
Truck color:
white
Truck capacity:
8.5
Truck name: Dongfeng Truck color: white Truck capacity: 8.5

```

## 五、程序分析题

1、结合程序的执行结果，分析：

```

#include <iostream>
using namespace std;

class root1 {
public:
    root1() { cout << "执行root1类的构造函数" << endl; }
};

class root2 {
public:
    root2() { cout << "执行root2类的构造函数" << endl; }
};

class mid : public root1, virtual public root2 {
public:
    mid() { cout << "执行mid类的构造函数" << endl; }
};

class top : public mid {
public:
    top() { cout << "执行top类的构造函数" << endl; }
};

void main() {
    top t;
}

```

(1) 描述：虚基类机制下的构造函数的执行顺序；

答：在main函数中实例化一个top类，top类继承自mid类，mid类继承自root1类和虚基类root2。当定义一个top类对象时，先按照顺序调用root2和root1的构造函数，再调用mid类的构造函数，最后调用自己的top类构造函数。

 Microsoft Visual Studio 调试控制台

```

执行root2类的构造函数
执行root1类的构造函数
执行mid类的构造函数
执行top类的构造函数

```

(2) 描述，若各个类都是默认构造函数，则最终派生类可以不写构造函数。

```

#include <iostream>
using namespace std;

class root1 {
public:
};

class root2 {
public:
};

class mid :public root1, virtual public root2 {
public:
};

class top:public mid {
public:
};

void main() {
    top t;
}

```

Microsoft Visual Studio 调试控制台

D:\XPfile\学习资料\年级分类\大二下\C++  
要在调试停止时自动关闭控制台，请启用“  
按任意键关闭此窗口。...”

当各个类都是默认构造函数，则最终派生类可以不写构造函数。即root1、root2、mid都使用默认构造函数，最终的派生类top可以不写构造函数。

## 2、同第一题比较，描述：

```

#include <iostream>
#include <string>
using namespace std;

class A {
public:
    A(string s) { cout << s << endl; }
};

class B :virtual public A {
public:
    B(string s1, string s2) :A(s1) { cout << s2 << endl; }
};

class C :virtual public A {
public:
    C(string s1, string s2) :A(s1) { cout << s2 << endl; }
};

class D :public B, public C {
public:
    D(string s1, string s2, string s3, string s4) :B(s1, s2), C(s3, s4), A(s1) { cout << s4 << endl; }
};

void main() {
    string strA = "class A"; string strB = "class B"; string strC = "class C"; string strD = "class D";
    D d(strA, strB, strC, strD);
}

```

(1) 若基类中有自定义构造函数，则最终派生类需要写构造函数。

答：若基类中有自定义构造函数，则最终派生类需要写构造函数。如果最终派生类不写构造函数则会出现“无法引用默认构造函数”的报错。而若各个类都是默认构造

函数，则最终派生类可以不写构造函数。

```
class D :public B, public C {
public:
    //D(string s1, string s2, string s3, string s4) :B(s1, s2), C(s3, s4), A(s1) { cout << s4 << endl; }
};

void main() {
    string strA = "class A"; string strB = "class B"; string strC = "class C"; string strD = "class D";
    //D d(strA, strB, strC, strD);
    D d;
}
```

无法引用 "D" 的默认构造函数 -- 它是已删除的函数

[联机搜索](#)

(2) 在有虚基类的时候，最终派生类的构造函数是否需要调用虚基类的构造函数？

答：在虚继承中，虚基类是由最终的派生类初始化的，换句话说，最终派生类的构造函数必须要调用虚基类的构造函数。

(3) 如果把virtual删除，结果发生什么变化？（构造函数被调用多次）

答：原先说明了公共基类A为虚基类时，可以只保存这个基类的一个实例，即只调用一次A的构造函数，当删除后，会出现不允许使用间接非虚拟基类。由于对A的访问不明确，出现的非法成员初始化。

Microsoft Visual Studio 调试控制台

```
class A
class B
class D
class D
```

```
class B :public A {
public:
    B(string s1, string s2) :A(s1) { cout << s2 << endl; }
};

class C :public A {
public:
    C(string s1, string s2) :A(s1) { cout << s2 << endl; }
};
```

E0293 不允许使用间接非虚拟基类

C2614 "D": 非法的成员初始化:"A"不是基或成员

C2385 对"A"的访问不明确

3、结合程序的运行结果,并修改程序使之符合多态性预期目的

```

#include "iostream" //这是一道较为典型的由静态绑定向动态绑定的迁移
using namespace std;

class Base {
public:
    void disp() { cout << "Base class" << endl; }
};

class Derive1 :public Base {
public:
    void disp() { cout << "Derive1 class" << endl; }
};

class Derive2 :public Base {
public:
    void disp() { cout << "Derive2 class" << endl; }
};

void main() {
    Base obj, *p;
    Derive1 obj1;
    Derive2 obj2;
    p = &obj;
    p->disp();
    p = &obj1;
    p->disp();
    p = &obj2;
    p->disp();
    obj1.disp();
    obj2.disp();
}

```

 Microsoft Visual Studio 调试控制台

```

Base class
Base class
Base class
Derive1 class
Derive2 class

```

答：首先为多态类的基类声明一个指针变量，然后让这个指针变量指向此多态类继承树中某一个类的对象。由于基类指针指向对象内存映像的首地址，它直接访问的是该对象的虚表指针，下一步由具体对象的虚表指针就可以访问到该对象所在的类中为虚函数定义的那一份代码。

```

class Base {
public:
    //虚函数定义
    virtual void disp() { cout << "Base class" << endl; }
};

class Derive1 :public Base {
public:
    //虚函数重定义
    void disp() { cout << "Derive1 class" << endl; }
};

class Derive2 :public Base {
public:
    //虚函数重定义
    void disp() { cout << "Derive2 class" << endl; }
};

```

Microsoft Visual Studio 调试控制台

```
Base class  
Derive1 class  
Derive2 class  
Derive1 class  
Derive2 class
```

4、请比较3，

```
#include "iostream"  
using namespace std;  
  
class Base {  
public:  
    virtual void disp() { cout << "Base class" << endl; }  
};  
class Derive1 :public Base {  
public:  
    void disp() { cout << "Derive1 class" << endl; }  
};  
class Derive2 :public Base {  
public:  
    void disp() { cout << "Derive2 class" << endl; }  
};  
void show(Base* p) {  
    p->disp();  
}  
void main() {  
    Base a;  
    Derive1 b;  
    Derive2 c;  
    show(&a);  
    show(&b);  
    show(&c);  
}
```

(1) 在程序的代码实现上，3和本题有什么异同点？

答：通过在基类对disp函数的定义前面加了virtual，实现了虚函数的定义。

(2) 结合程序的运行结果，分析：若没有声明基类A的成员函数show()为虚函数，则运行结果如何？

答：程序运行时，通过动态绑定，使得可以在main函数中分别调用Derive1和Derive2的show函数。当没有声明基类A的成员函数show()为虚函数，调用的都是基类A的show函数。

这是因为C++编译器在默认情况下，对函数成员的调用实施的是静态绑定。在函数覆盖时，在基类中调用函数，实际上调用的是基类中的函数，而不是子类中的函数。为了解决上述问题，可以设置虚函数。虚函数也是一个函数成员，唯一要求的是在子类中一定要覆盖它。对于虚函数，编译器完成的是动态绑定，即对函数的调用是在运行时确定的。声明虚函数的方法很简单，只需要将virtual 关键字放在函数类型之前

```
Microsoft Visual Studio 调试控制台
Base class
Derive1 class
Derive2 class

Microsoft Visual Studio 调试控制台
Base class
Base class
Base class
```

六、选做题：思考一下概念，并用程序给出具体例子（代码要做标注）

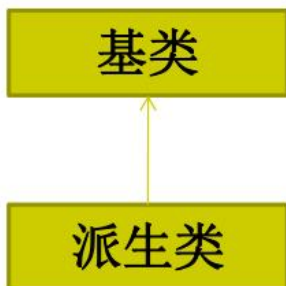
1、单继承，请给出图，例子重点描述

（1）派生类构造函数

（2）派生类中基类成员的访问属性

答：

单继承：派生类只从一个基类派生。





```

#include<iostream>
using namespace std;
//单继承
class A1 { // 定义基类A1
    int a1;
public:
    A1(int i) { a1 = i; cout << "constructor A1." << a1 << endl; }
    void print() { cout << a1 << endl; }
};
class B : public virtual A1 { // 定义派生类 B, 基类为A1和A2
    int b;
public:
    B(int i, int l) : A1(i) { // 派生类构造函数
        b = l;
        cout << "constructor B." << b << endl;
    }
    void print() {
        A1::print();
    }
};
void main() {
    B bb(1, 2);
    bb.print();
}

```

Microsoft Visual Studio 调试控制台

```

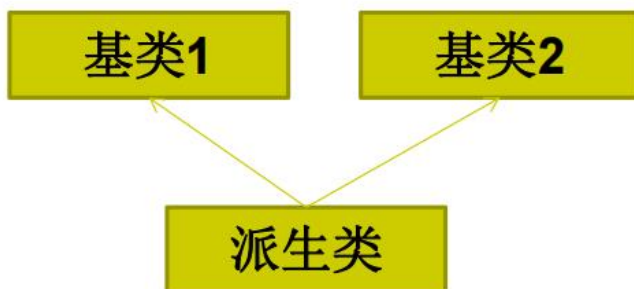
constructor A1. 1
constructor B. 2
1

```

2、多继承，请给出图和例子。思考，如何解决二义性问题？

答：

多继承：派生类从多个基类派生。通过虚基类解决二义性问题。



```

#include<iostream>
using namespace std;
//多继承：派生类从多个基类派生
class A1 { // 定义基类A1
public:
    int a1;
    A1(int i) { a1 = i; cout << "constructor A1." << a1 << endl; }
    void print() { cout << a1 << endl; }
};
class A2 { // 定义基类A2
public:
    int a2;
    A2(int i) { a2 = i; cout << "constructor A2." << a2 << endl; }
    void print() { cout << a2 << endl; }
};
class B :public virtual A2, public virtual A1 { // 定义派生类 B，基类为A1和A2
public:
    int b;
    B(int i, int j, int k, int l) : A1(i), A2(j) { //派生类构造函数
        b = 1;
        cout << "constructor B." << b << endl;
    }
    void print() {
        A1::print();
        A2::print();
    }
};
void main() {
    B bb(1, 2, 3, 4);
    bb.print();
}

```

Microsoft Visual Studio 调试控制台

```

constructor A2. 2
constructor A1. 1
constructor B. 4
1
2

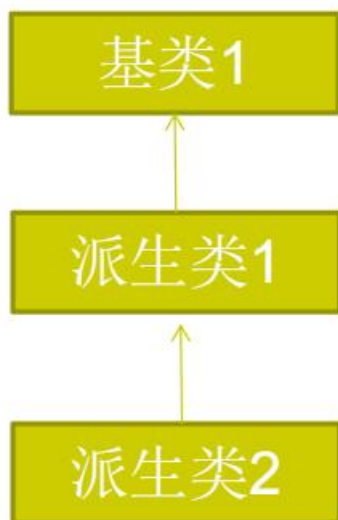
```

3、多层派生，请给出图，例子重点描述

- (1) 各个层次的类的构造函数的描述
- (2) 各个层次中父类成员的访问属性

答：

多层派生：派生类又作为基类，继续派生新的类



```

#include <iostream>
#include<string>
using namespace std;
//多层派生
class Student//声明基类
{
public://公用部分
    Student(int n, string nam) { //基类构造函数
        num = n;
        name = nam;
    }
    void display() { //输出基类数据成员
        cout << "num:" << num << endl;
        cout << "name:" << name << endl;
    }
protected://保护部分
    int num;//基类有两个数据成员
    string name;
};
class Student1 : public Student//声明公用派生类Student1
{
public:
    Student1(int n, string nam, int a) :Student(n, nam) { //派生类构造函数
        age = a;
    } //在此处只对派生类新增的数据成员初始化
    void show() { //输出num, name和age
        display(); //输出num和name
        cout << "age: " << age << endl;
    }
private://派生类的私有数据
    int age; //增加一个数据成员
};

```

```

};
class Student2 :public Student1 //声明间接公用派生类Student2
{
public://下面是间接派生类构造函数
    Student2(int n, string nam, int a, int s) :Student1(n, nam, a) { score = s; }
    void show_all() { //输出全部数据成员
        show(); //输出num和name
        cout << "score:" << score << endl; //输出age
    }
private:
    int score; //增加一个数据成员
};
int main()
{
    Student2 stud(10010, "Li", 17, 89);
    stud.show_all(); //输出学生的全部数据
    return 0;
}

```

Microsoft Visual Studio 调试控制台

```

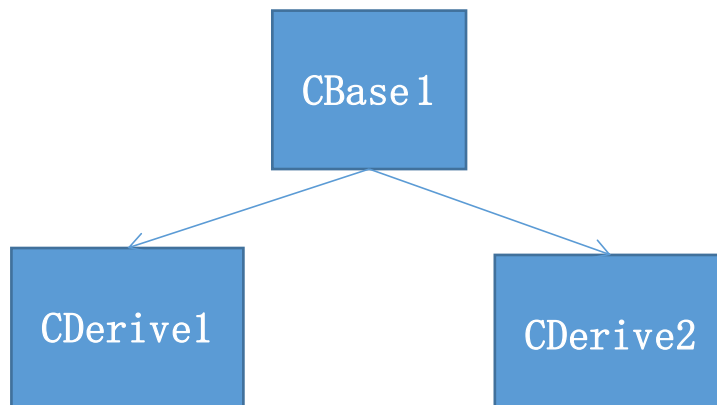
num:10010
name:Li
age: 17
score:89

```

4、多派生，请给出图和例子。

答：

多重派生：由一个基类派生出多个不同的派生类。



```

#include <iostream>
#include <string>
using namespace std;
//多重派生
class CBase1
{
public:
    CBase1() :a(1){
        cout << "base1 structure..." << endl;
    }
    ~CBase1(){
        cout << "base1 destructure ..." << endl;
    }
    void print(){
        cout << "a=" << a << endl;
    }
protected:
    int a;
};

class CDerivel :public CBase1
{
public:
    CDerivel():b(2) {
        cout << "base2 structure..." << endl;
    }
    ~CDerivel(){
        cout << "base2 destructure..." << endl;
    }
    void print(){
        CBase1::print();
        b2.print();
    }
};


```

```

        cout << "b=" << b << endl;
    }
private:
    CBase1 b2;
    int b;
};
class CDerive2 :public CBase1
{
public:
    CDerive2():c(3) {
        cout << "derive structure..." << endl;
    }
    ~CDerive2() {
        cout << "derive destructure..." << endl;
    }
    void print() {
        CBase1::print();
        b1.print();
        cout << "c=" << c << endl;
    }
private:
    CBase1 b1;
    int c;
};

int main() {
    CDerive1 d1;
    d1.print(); //调用函数时应加上括号
    CDerive2 d;
    d.print(); //调用函数时应加上括号
}

```

 Microsoft Visual Studio 调试控制台

```
base1 structure...  
base1 structure...  
base2 structure...  
a=1  
a=1  
b=2  
base1 structure...  
base1 structure...  
derive structure...  
a=1  
a=1  
c=3  
derive destructure...  
base1 destructure ...  
base1 destructure ...  
base2 destructure...  
base1 destructure ...  
base1 destructure ...
```

### 三、 附录

源程序文件项目清单：src/homework