燕山大學
YANSHAN UNIVERSITY

C++面向对象程序设计

实验指导书

（6）类的继承与派生

燕山大学软件工程系

目　录

实验 06 类的继承与派生

1.1 时间安排 4 学时

本实验安排 4 个实验课时。

1.2 实验目的和要求

1、从深层次上理解继承与派生的关系

2、掌握不同继承方式下，从派生类/对象内部和外部对基类成员的访问控制权限。

3、掌握单继承和多继承的使用方法，尤其是派生类构造函数的声明方式。

4、掌握继承与派生下构造函数与析构函数的调用顺序。

5、理解“类型兼容”原则

6、学习利用虚基类解决二义性问题。

1.3 实验内容 I (调试、理解、体会、掌握)

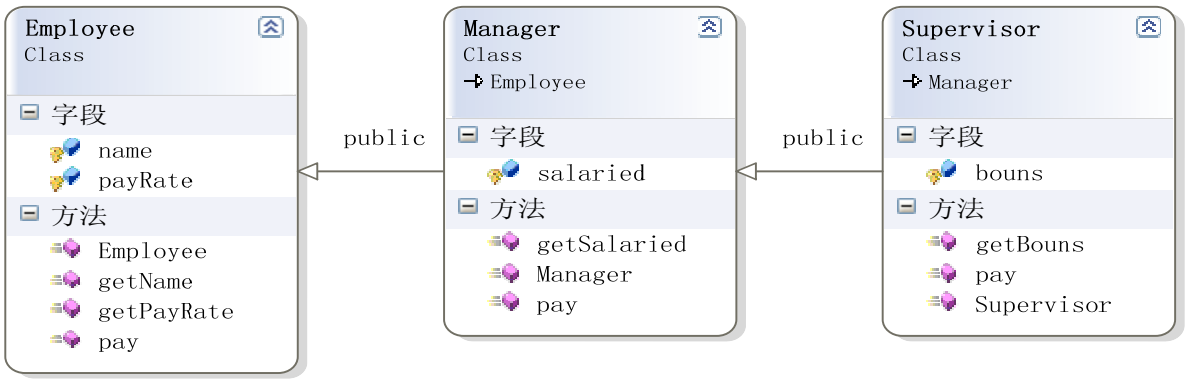
（1）阅读下列程序，仔细体会类的继承与派生机制、类型兼容原则等。

对于一个公司的雇员来说，无非三种：普通雇员、管理人员和主管。

这些雇员有共同的数据：名字、每小时的工资，也有一些共同的操作：数据成员初始化、读雇员的数据成员及计算雇员的工资。

但是，他们也有不同。例如，管理人员除有这些共同的特征外，有可能付固定薪水，主管除有管理人员的共同特征外，还有其它物质奖励等。

三种雇员中，管理人员可以看作普通雇员的一种，而主管又可以看作管理人员的一种。我们很容易想到使用类继承来实现这个问题：普通雇员作为基类，管理人员类从普通雇员类派生，而主管人员类又从管理人员类派生。




```
#include <iostream>
using namespace std;
/***************定义Employee类*******************/
class Employee
```

```
{
public:
    Employee(char *theName, float thePayRate);
    char *getName() const;
    float getPayRate() const;
    float pay(float hoursWorked) const;
protected:
    char *name; //雇员名称
    float payRate; //薪水等级
};
Employee::Employee(char *theName, float thePayRate)
{
    name = theName;
    payRate = thePayRate;
}
char *Employee::getName() const
{
    return name;
}
float Employee::getPayRate() const
{
    return payRate;
}
float Employee::pay(float hoursWorked) const
{
    return hoursWorked * payRate;
}
/*****  管理人员是普通雇员的一种  定义Manager类**********/
class Manager : public Employee
{
public:
    /*参数isSalaried代表付薪方式：true  付薪固定工资，false按小时付薪*/
    Manager(char *theName, float thePayRate, bool isSalaried);
    bool getSalaried() const;
    float pay(float hoursWorked) const;
protected:
    bool salaried;
};
Manager::Manager(char *theName,float thePayRate,bool isSalaried): Employee(theName, thePayRate)
{
    salaried = isSalaried;
}
bool Manager::getSalaried() const
{
```

```
    return salaried;
}
float Manager::pay(float hoursWorked) const
{
    if (salaried)
    return payRate;
    /*  否则*/
    return Employee::pay(hoursWorked);
}
/****主管人员是管理人员的一种　定义Supervisor类**********/
class Supervisor : public Manager
{
public:
    Supervisor(char *theName, float thePayRate, float theBouns):Manager (theName,
thePayRate,true) ,bouns(theBouns) {}
    float getBouns() const { return bouns; }
    float pay(float hoursWorked) const {return payRate+bouns;}
protected:
    float bouns;
};
/*类型兼容规则：指在需要基类对象的任何地方，都可以使用公有派生类
的对象来替代。此时，可调用的均为基类成员。*/
/***************定义普通函数Display　引用*******************/
void Display(Employee & ptr,float pay)
{
    cout<<"name:"<<ptr.getName ()<<endl;
    cout<<"Pay:"<<ptr.pay( pay )<<endl;
}
/***************定义普通函数Display　指针*******************/
void Display(Employee * ptr,float pay)
{
    cout<<"name:"<<ptr->getName ()<<endl;
    cout<<"Pay:"<<ptr->pay( pay )<<endl;
}
/*(1)普通雇员类是所有类的基类，描述了雇员的一些基本信息；
(2)管理人员类从普通雇员类派生，管理人员的付薪方式与普通雇员可能同，
所以该类添加了一个成员变量标识这种差异(salaried)，并覆盖了基类的pay()函数。
(3)主管类从管理人员类派生，主管人员是管理人员的一种，他们不仅支付固定薪水，而且还有奖金。所
以在主管类种添加了bonus成员，保存他们的奖金数额，
并覆盖了管理人员类的pay()函数重新计算工资。*/
void main()
{
    Employee e("Jack",50.00);
    Manager m("Tom",6000,true);
```

```
    Supervisor * Sptr=new Supervisor("Tanya",8000.00,5000.00);   //动态申请空间
    cout<<"Name:"<<e.getName()<<endl;
    cout <<"Pay:"<<e.pay(60)<<endl; //设每月工作小时
    cout <<"Name:"<<m.getName()<<endl;
    cout <<"Pay:"<<m.pay(40)<<endl;
    cout <<"Name:"<<Sptr->getName()<<endl;
    cout <<"Pay:"<<Sptr->pay(40)<<endl; //参数这里不起作用
    cout<<"/*********类型兼容规则测试*******************/"<<endl;
    /*三个Display，实质上调用的均是Employee的pay函数，
    因此，在显示m、*Sptr的对象内容时，出现了与上面不符的结果。*/
  Display(e,60);
  Display(m,40);
  Display(Sptr,40);
}
```

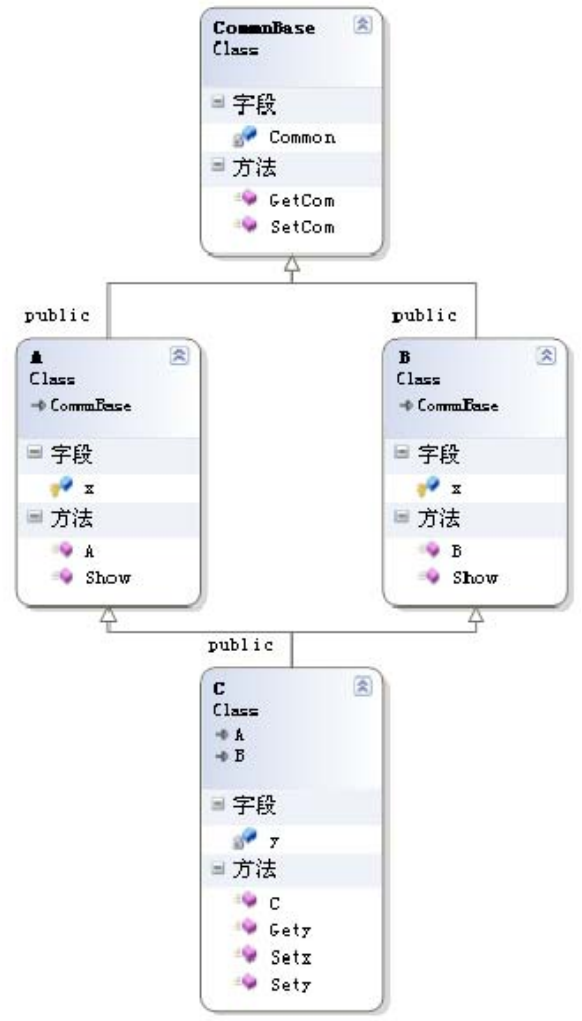
（2）调试并运行下列程序，分析、理解、体会多继承且含有内嵌对象时的构造函数及析构函数的调用顺序。

```
#include <iostream>
using namespace std;
class Base1 {  //基类Base1，构造函数有参数
public:
    Base1(int i) { cout << "Constructing Base1 " << i << endl; }
    ~Base1() { cout << "Destructing Base1" << endl; }
};
class Base2 {  //基类Base2，构造函数有参数
public:
    Base2(int j) { cout << "Constructing Base2 " << j << endl; }
    ~Base2() { cout << "Destructing Base2" << endl; }
};
class Base3 {  //基类Base3，构造函数无参数
public:
    Base3() { cout << "Constructing Base3 *" << endl; }
    ~Base3() { cout << "Destructing Base3" << endl; }
};
/*注意基类的继承顺序*/
class Derived: public Base2, public Base1, public Base3 {
//派生新类Derived，注意基类名的顺序
public:
    /*理解派生对象的构造函数与基类、内嵌对象的构造函数的调用顺序*/
    Derived(int a, int b, int c, int d): Base1(a), member2(d), member1(c), Base2(b)
  {cout << "Constructing Derived " << endl; }
    /*理解派生对象的析构函数与基类、内嵌对象的析构函数的调用顺序*/
   ~Derived() { cout << "Destructing Derived" << endl; }
private:
  /*注意派生类的私有成员的声明顺序*/
```

```
    Base1 member1;
    Base2 member2;
    Base3 member3;
};
int main() {
    Derived obj(1, 2, 3, 4);
    return 0;
        }
```

（3）利用 作用域分辨符 解决继承与派生时的访问二义性问题

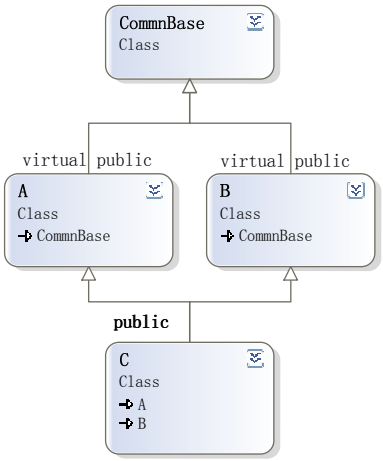
```
# include <iostream>
using namespace std;
class CommnBase
{
public:
    void SetCom( int com){Common=com;}
    int GetCom() const {return Common;}
private:
    int Common;
};
class A : public CommnBase
{
protected:
   int x ;
public:
   A (int a ) {x=a ; }
   void Show (void )   {cout<<"x="<<x<<'\n' ; }
} ;
class B    : public CommnBase
{
protected :
   int    x ;
public :
       B ( int a ){x=a ;}
void Show(void ) {cout<<"x="<<x<<'\n' ;}
};
class   C : public A,public B
{
  public :
    C (int a , int b ):A(a) ,B (b) { }
     //void Setx(int a){x=a;}     //A，冲突，二义性，编译时，不知是A 的x，还是B的x.
     void Setx(int a){A::x=a ;}   //改为：{A::x=a ;}
     void Sety(int b){y=b;}
     int Gety(void ) {return y ;}
```



```
private:
        int y ;
};
void main (void )
{
    C c1(10 ,100) ;
    //c1.Show ( ) ;      //冲突，二义性，编译时，不知是A 的show，还是B的show.
    c1.B::Show();       //改为；c1.B::Show();
 /******下述运行结果表明，A、B中均含有各自的CommnBase成员副本********/
    //c1.SetCom (100);//连接时出错无法解析的外部命令
    c1.A::SetCom (30);
    c1.B::SetCom (50);
   // cout<<c1.GetCom();  //编译时出错，GetCom访问对象不明确
    cout<<c1.A::GetCom()<<endl;
    cout<<c1.B::GetCom()<<endl;
}
```

（4）利用 虚基类 解决继承与派生时的访问二义性问题

按下列方式修改第（3）题源程序的相关行及主程序，运行并体会虚基类的作用。




```
class A : virtual public CommnBase
class B : virtual public CommnBase
void main (void )
{
    C c1(10 ,100) ;
    c1.B::Show();       //改为；c1.B::Show();
 /******下述运行结果可见，A、B中均含的CommnBase成员副本是同一个********/
    cout<<"*************虚基类效果测试***************"<<endl;
    cout<<"设置c1.A的Common值为"<<endl;
    c1.A::SetCom (30);
    cout<<"输出c1.B的Common值为："<<c1.B::GetCom()<<endl;
    cout<<"设置c1.B的Common值为"<<endl;
    c1.B::SetCom (50);
    cout<<"输出c1.A的Common值为："<<c1.A::GetCom()<<endl;
    cout<<"不用作用域分辨符，直接设置CommnBase的Common值为"<<endl;
    c1.SetCom (100);
    cout<<"直接输出CommnBase的Common值为："<<c1.GetCom()<<endl;
    cout<<"上述说明：A、B所含有的CommnBase成员副本是同一个"<<endl;
}
```

（5）不同继承方式下的访问控制权限表，要求理解并掌握。

	基类内部函数	基类对象	private继承方式		protected继承方式		public继承方式	
			派生类内部函数	派生类对象	派生类内部函数	派生类对象	派生类内部函数	派生类对象
基类prvate成员	可访问	不可访问	不可访问	不可访问	不可访问	不可访问	不可访问	不可访问
基类protected成员	可访问	不可访问	可访问，转为private	不可访问	可访问,转为protected	不可访问	可访问，保持protected	不可访问
基类public成员	可访问	可访问	可访问,转为private	不可访问	可访问,转为protected	不可访问	可访问,保持public	可访问

1.4 实验内容 II (自主完成)

注：将题目的构思过程、源码、运行结果（截图）、心得体会等内容按要求填写，详见实验报告模板。

1、编写 C++程序，以完成以下功能（具体的数据成员、函数成员，请自主定义）：

（1）声明一个基类 Shape（形状），其中包含一个方法来计算面积；

（2）从 Shape 派生两个类：矩形类（Rectangle）和圆形类（Circle）；

（3）从 Rectangle 类派生正方形类 Square；

（4）分别实现派生类构造函数、析构函数及其它功能的成员函数；

（5）创建各派生类的对象，观察构造函数、析构函数的调用次序；

（6）计算不同对象的面积。

2、将 1 中 Shape 基类计算面积的方法定义为虚函数，比较与【形状（一）】程序的差异，体验其优点。