第4篇

面向对象的程序设计

第11章 继承与派生

第12章 多态性与虚函数

第13章 输入输出流

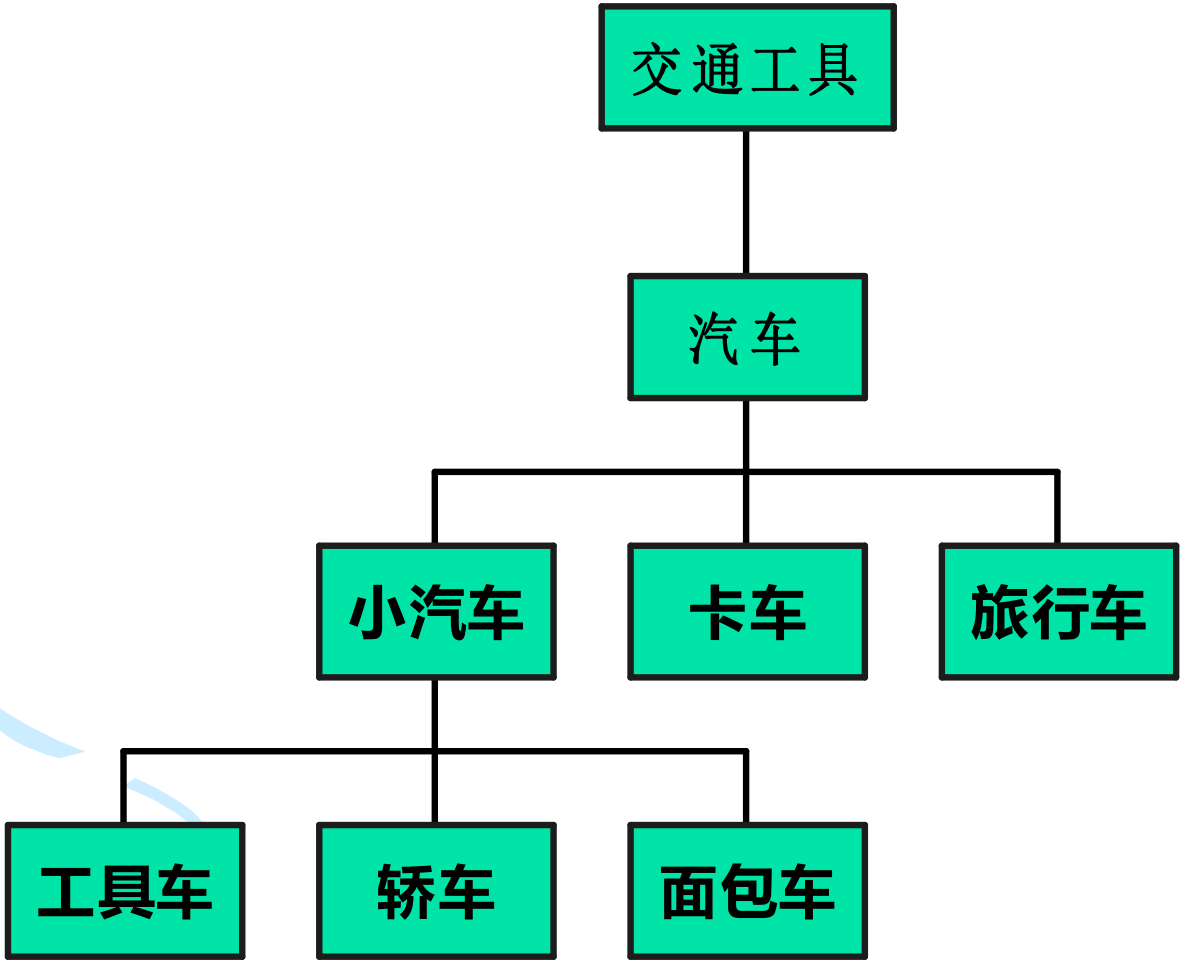
第14章 C++工具

第11章 继承与派生

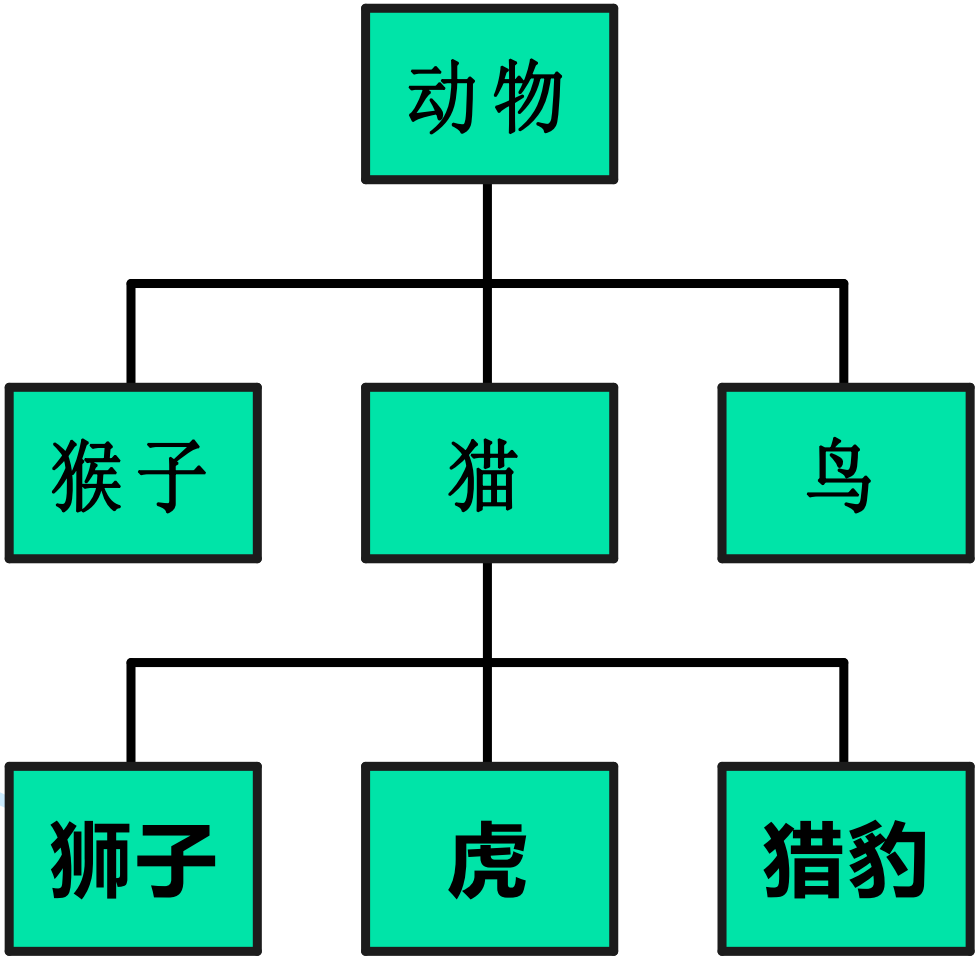
Inherit and derivation of class

- The process of constructing a new class based on specialties of an existing class is known as inherit.
- The process of constructing a new class by adding self-specialties based on an existing class is named as derivation.
- The existing class inherited by is called basic class or father class.
- The new class constructed by derivation is called derived class.

Example of inherit and derivation



Example of inherit and derivation



Purpose of inherit and derivation

- **Purpose of inherit：**

  Realizing code reused.

- **Purpose of derivation：**

  When a new problem appears and the original program can’t resolve it, we must reconstruct the original program.

11.2 派生类的声明方式

```
class Student1: public Student//声明基类是Student
{
public:
    void display_1( )                    //新增加的成员函数
    {
        cout<<"age: "<<age<<endl;
        cout<<"address: "<<addr<<endl;
    }
private:
    int age;                             //新增加的数据成员
    string addr;                         //新增加的数据成员
};
```

Mode of inherit

- The effect of different manner of inherit:
  1. Member of derived class controls and access the member of basic class.
  2. Object of derived class controls and access the member of basic class.
- Three types of inherit
  - public inherit
  - private inherit
  - protected inherit

声明派生类的一般形式为

class 派生类名:［继承方式］ 基类名

{

　　派生类新增加的成员

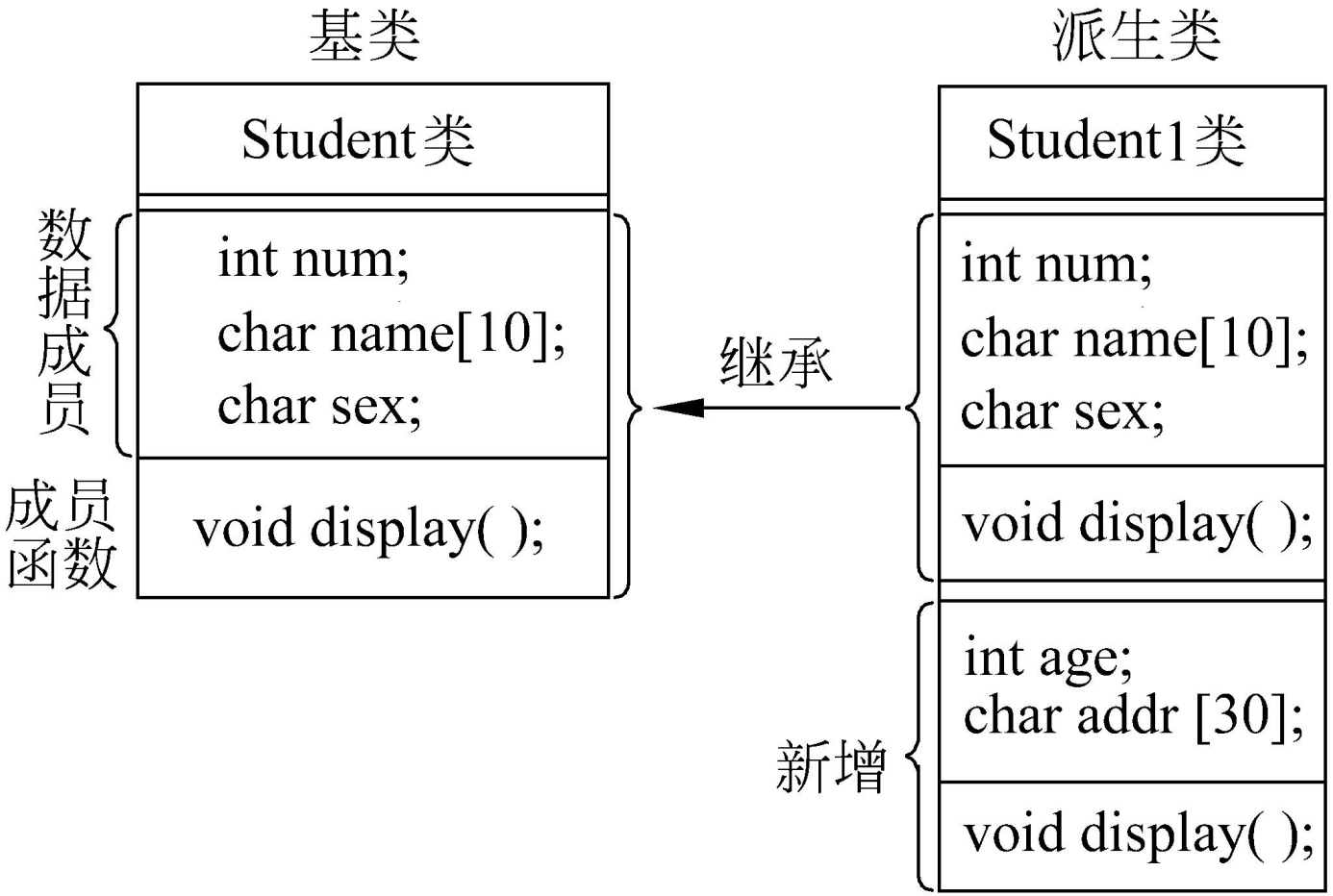
};

继承方式包括: **public**(公用的),**private**(私有的)和**protected**(受保护的)。

如果不写此项，则默认为**private**(私有的)。

11.3 派生类的构成

派生类中的成员包括:从基类继承来的成员；
声明派生类时增加的成员。



派生类的构造:

(1) 从基类接收成员。派生类把基类全部的成员(不包括构造函数和析构函数)接收过来。

(2) 调整从基类接收的成员。

(3) 声明派生类时增加的成员，体现了派生类对基类功能的扩展。

(4) 定义派生类的构造函数和析构函数，因为构造函数和析构函数是不能从基类继承的。

派生类是抽象基类的具体实现。

11.4 派生类成员的访问属性

(1) 基类的成员函数访问基类成员。

(2) 派生类的成员函数访问派生类增加的成员。

(3) 基类的成员函数访问派生类的成员。//不能

(4) 派生类的成员函数访问基类的成员。

(5) 在派生类外访问派生类的成员。

(6) 在派生类外访问基类的成员。

要考虑基类成员的访问属性和派生类的继承方式。

（1） 公用继承(public inheritance)

基类的公用成员和保护成员在派生类中保持原有属性，其私有成员仍为基类私有。

（2） 私有继承(private inheritance)

基类的公用成员和保护成员在派生类中成了私有成员，其私有成员仍为基类私有。

（3） 受保护的继承(protected inheritance)

基类的公用成员和保护成员在派生类中成了保护成员，其私有成员仍为基类私有。

11.4.1 公用继承 (public inheritance)

用公用继承方式建立的派生类称为公用派生类(public derived class)；其基类称为公用基类(public base class)。

采用公用继承方式时，基类的公用成员和保护成员在派生类中仍然保持原有属性；

而基类的私有成员在派生类中成为派生类中的不可访问的成员。

Public inherit

- Accessing attribute of public and protected member in basic class will keep invariable in derived class.
- Member functions of derived class can directly access the public and protected member of basic class, but can't access the private member of basic class.
- The objects of derived class are only able to access the public member of basic class.

例11.1 访问公有基类的成员

```
#include <iostream>
#include <string>
using namespace std;
Class Student                          //声明基类
{
public:                                //基类公用成员
   void get_value( )
   { cin>>num>>name>>sex;}
   void display( )
   {
      cout<<" num: "<<num<<endl;
      cout<<" name: "<<name<<endl;
      cout<<" sex: "<<sex<<endl;
   }
private :                                        //基类私有成员
   int num;
   string name;
   char sex;
};
```

```
class Student1: public Student //声明派生类Student1
{
public:
   void display_1( )
   {
      cout<<" num: "<<num<<endl;
      cout<<" name: "<<name<<endl;
      cout<<" sex: "<<sex<<endl;
      cout<<" age: "<<age<<endl;
      cout<<" address: "<<addr<<endl;
   }
private:
   int age;
   string addr;
};
```

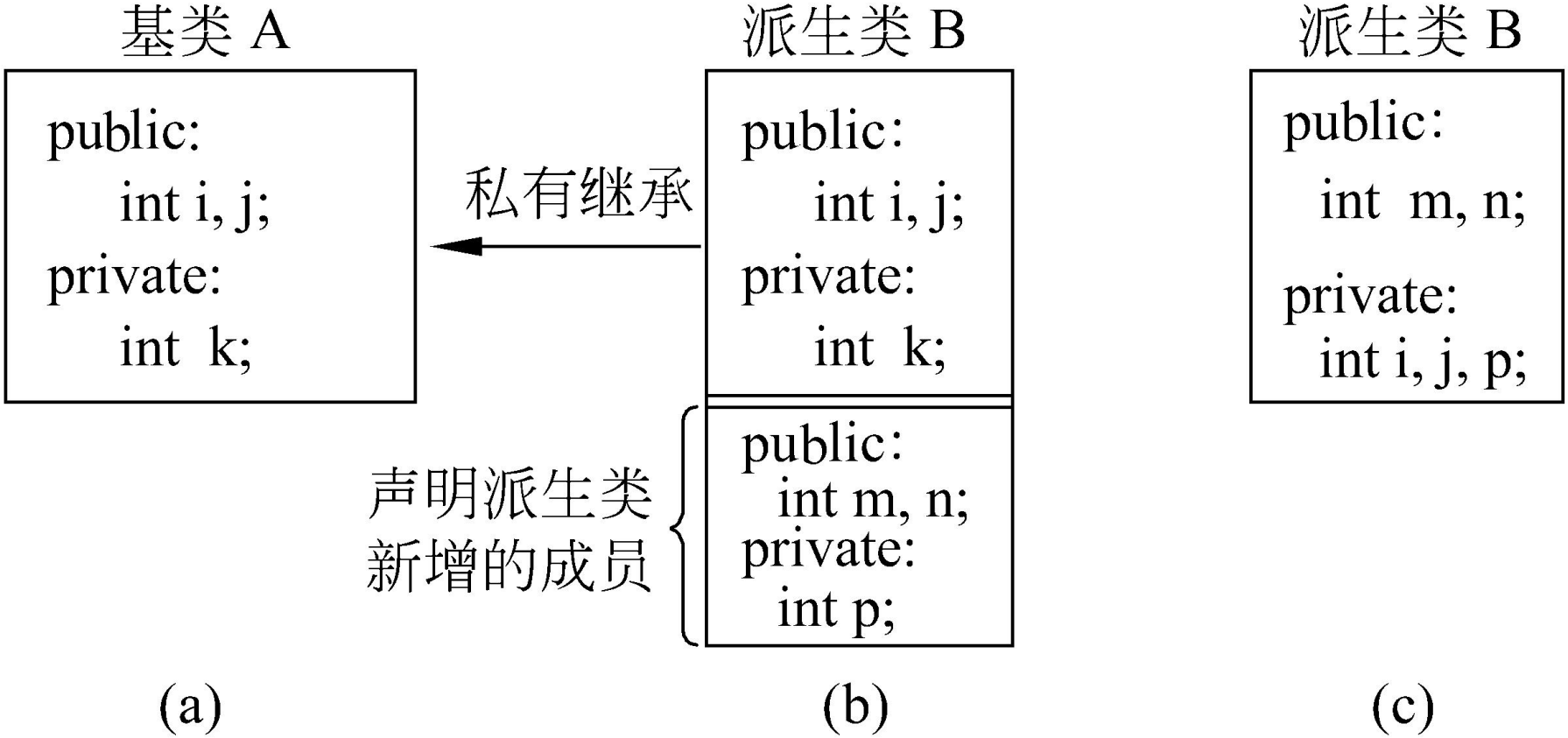
```
int main( )
{
    Student1  stud1;
    stud1.display();
    stud1.display_1( );
    stud1.age=18;
     stud1.num=10023;
    return 0;
}
```

11.4.2 私有继承

用私有继承方式建立的派生类称为私有派生类(private derived class)，其基类称为私有基类(private base class)。

私有基类的公用成员和保护成员在派生类中的访问属性相当于派生类中的私有成员。

私有基类的私有成员在派生类中成为不可访问的成员。


基类 A
public:
int i, j;
private:
int k;
私有继承
派生类 B
public:
int i, j;
private:
int k;
声明派生类
新增的成员
public:
int m, n;
private:
int p;
派生类 B
public:
int m, n;
private:
int i, j, p;
(a)
(b)
(c)

私有基类的私有成员只能被基类的成员函数引用，在基类外不能访问他们，因此它们在派生类中是隐蔽的，不可访问的。

对于不需要再往下继承的类的功能可以用私有继承方式把它隐蔽起来，这样，下一层的派生类无法访问它的任何成员。

一个成员在不同的派生层次中的访问属性与继承方式有关。

Private inherit

- public and protected member of basic class will be regarded as private member in derived class.
- Member functions of derived class can directly access the public and protected member of basic class, but can't access the private member of basic class.
- The objects of derived class can't access the any member of basic class.

例11.2访问私有基类的成员

```
class Student1: private Student     //声明派生类 Student1
{
public:
    void display_1( )
    {
        cout<<"age: "<<age<<endl;
        cout<<"address: "<<addr<<endl;
    }
private:
    int age;
    string addr;
};
```

```
int main( )
{
    Student1  stud1;
    stud1.display();          //错误
    stud1.display_1( );
    stud1.age=18;             //错误
    return 0;
}
```

结论

(1) 不能通过派生类对象（类外）引用从私有基类继承过来的任何成员。

(2) 派生类的成员函数不能访问私有基类的私有成员，但可以访问私有基类的公用成员。

可以通过派生类的成员函数调用私有基类的公用成员函数，从而引用私有基类的私有成员。

例11.2改 访问私有基类成员

```
void display_1( )
{ display();
  cout<<"age: "<<age<<endl;
  cout<<"address: "<<addr<<endl;}
int main( )
{ Student1 stud1;
   stud1.display_1( );
   return 0;
}
```

私有派生类限制太多，使用不便，一般不使用。

11.4.3 保护成员和保护继承

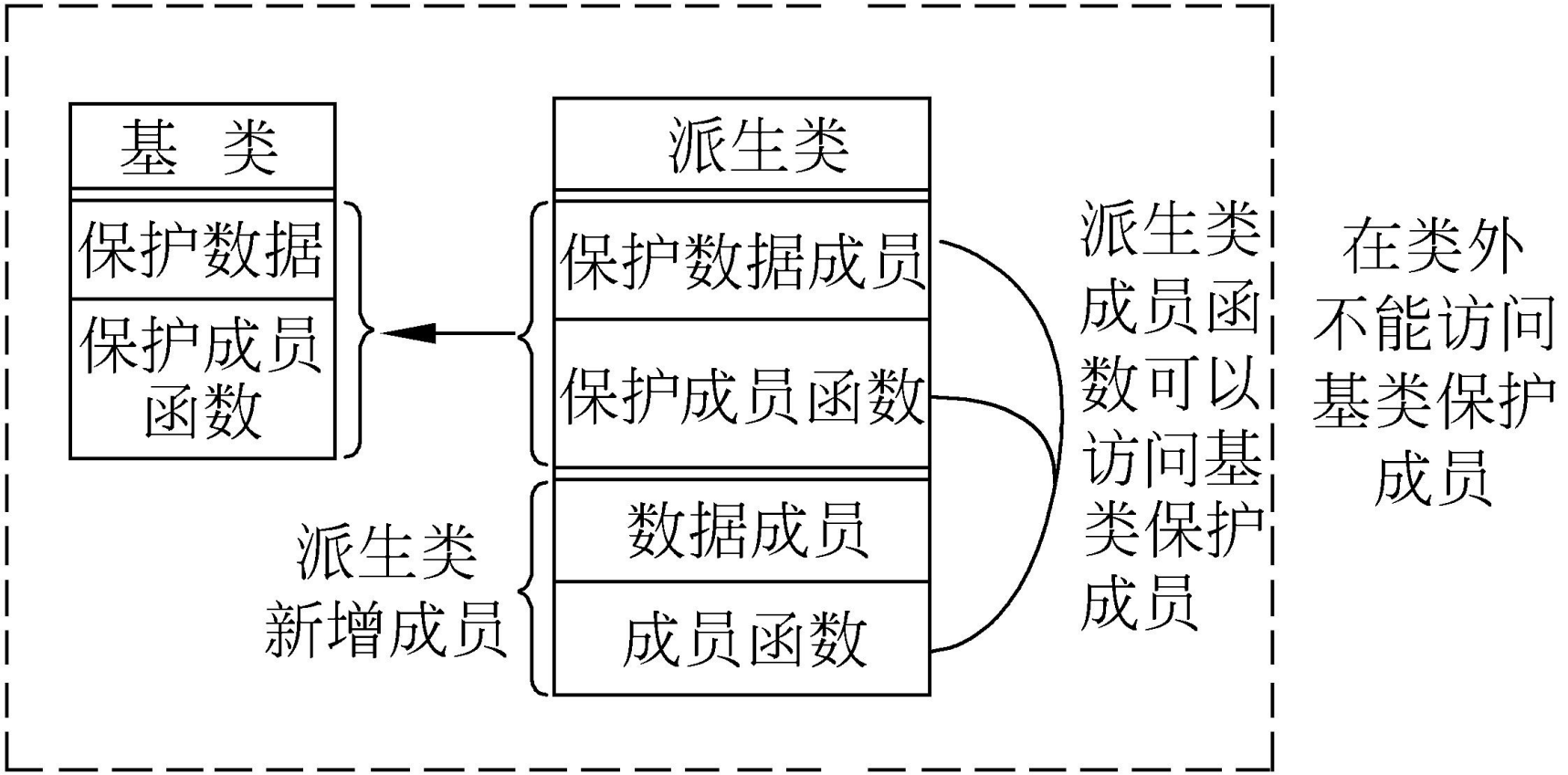
用保护继承方式建立的派生类称为保护派生类(protected derived class)，其基类称为受保护的基类(protected base class)，简称保护基类。

保护基类的公用成员和保护成员在派生类中都成了保护成员；

其私有成员仍为基类私有。

11.4.3 保护成员和保护继承

由protected声明的成员称为“保护成员”。
保护成员可以被派生类的成员函数引用。



用保护继承方式建立的派生类称为保护派生类(protected derived class)，其基类称为受保护的基类(protected base class)，简称保护基类。

保护基类的公用成员和保护成员在派生类中都成了保护成员；

其私有成员仍为基类私有。

私有继承和保护继承在直接派生类中的作用实际上是相同的，在类外不能访问任何成员，但如果继续派生，在新的派生类中，两种继承方式的作用就不同了。

Protected inherit

- public and protected member of basic class will be regarded as private member in derived class.
- Member functions of derived class can directly access the public and protected member of basic class, but can't access the private member of basic class.
- The objects of derived class can't access the any member of basic class.

例11.3 在派生类中引用保护成员

```
#include <iostream>
#include <string>
using namespace std;
class Student//声明基类
{
public:                          //基类公用成员
    void display( );
protected :                        //基类保护成员
    int num;
    string name;
    char sex;
};
```

```
void Student::display( )          //定义基类成员函数
{
   cout<<"num: "<<num<<endl;
   cout<<"name: "<<name<<endl;
   cout<<"sex: "<<sex<<endl;
}

class Student1: protected Student
{
public:
   void display1( );              //派生类公用成员函数
private:
   int age;                       //派生类私有数据成员
   string addr;                   //派生类私有数据成员
};
```

```
void Student1::display1( )
{
    cout<<"num: "<<num<<endl;
    cout<<"name: "<<name<<endl;
    cout<<"sex: "<<sex<<endl;
    cout<<"age: "<<age<<endl;
    cout<<"address: "<<addr<<endl;
}

int main( )
{
    Student1 stud1;
    stud1.display1( );
    stud1.num=10023;                    //错误
    return 0;
}
```

在派生类的成员函数中引用基类的保护成员是合法的。保护成员和私有成员不同之处，在于把保护成员的访问范围扩展到派生类中。

私有继承和保护继承方式在使用时需要十分小心，很容易搞错，一般不常用。

结论

- (1) 在派生类中，成员有4种不同的访问属性:
  - ① 公用的，派生类内和派生类外都可以访问。
  - ② 受保护的，派生类内可以访问，派生类外不能访问，其下一层的派生类可以访问。
  - ③ 私有的，派生类内可以访问，派生类外不能访问。
  - ④ 不可访问的，派生类内和派生类外都不能访问。
- (2) 类的成员在不同作用域中有不同的访问属性。

例11.4 多级派生的访问属性

```
class A
{
public:
    int i;
protected:
    void f2( );
    int j;
private:
    int k;
};
```

```
class B: public A
{
public:
    void f3( );
protected:
    void f4( );
private:
    int m;
};
```

```
class C: protected B
{
public:
    void f5( );
private:
    int n;
};
```

各成员在不同类中的访问属性如下：

	i	f2	j	k	f3	f4	m	f5	n
基类A	公用	保护	保护	私有					
公用派生类B	公用	保护	保护	不可访问	公用	保护	私有		
保护派生类C	保护	保护	保护	不可访问	保护	保护	不可访问	公用	私有

- 多级派生时采用公用继承，那么直到最后一级派生类都能访问基类的公用成员和保护成员。
- 如果采用私有继承，经过若干次派生之后，基类的所有的成员已经变成不可访问的了。
- 如果采用保护继承，经过多次派生后，很难记住哪些成员可以访问，哪些成员不能访问。
- 在实际中，常用的是公用继承。

11.5 派生类的构造函数和析构函数

派生类接收的基类成员不包括构造函数和析构函数。

在设计派生类的构造函数时，不仅要考虑派生类所增加的数据成员的初始化，还应考虑接收的基类的数据成员的初始化。

解决这个问题的思路是：在执行派生类的构造函数时，调用基类的构造函数。

11.5.1 简单的派生类的构造函数

简单的派生类只有一个基类，而且只有一级派生(只有直接派生类，没有间接派生类)，在派生类的数据成员中不包含基类的对象(即子对象)。

例11.5 简单的派生类的构造函数

```
#include <iostream>
#include<string>
using namespace std;
class Student                          //声明基类Student
{
public:
    Student(int n,string nam,char s)          //基类构造函数
    {
        num=n;
        name=nam;
        sex=s;
    }
    ~Student( ){ }
protected:
    int num;
    string name;
    char sex ;
};
```

```
class Student1: public Student
{
public:
   Student1(int n,string nam,char s,int a,string ad)
 :Student(n,nam,s)
   { age=a;    addr=ad;}
   void show( )
   {
      cout<<"num: "<<num<<endl;
      cout<<"name: "<<name<<endl;
      cout<<"sex: "<<sex<<endl;
      cout<<"age: "<<age<<endl;
      cout<<"address: "<<addr<<endl<<endl;
}
```

```
    ~Student1( ){ }                    //派生类析构函数
private:                               //派生类的私有部分
    int age;
    string addr;
};

int main( )
{
    Student1 stud1(10010,"Wang-li",'f',19,"115 Beijing
Road,Shanghai");
    Student1 stud2(10011,"Zhang-fun",'m',21,"213 Shanghai
Road,Beijing");
    stud1.show( );
    stud2.show( );
    return 0;
}
```

派生类构造函数首行的写法:

Student1(int n，string nam，char s，int a，string ad):Student(n，nam，s)

其一般形式为

派生类构造函数名（总参数表列）:基类构造函数名（参数表列）

{派生类中新增数据成员初始化语句}

派生类构造函数在类外面定义，在类中函数的声明:

Student1(int n，string nam，char s，int a，string ad);

派生类构造函数定义:

Student1:: Student1(int n，string nam,char s,int a，string ad)

:Student(n，nam，s)//调用基类构造函数

{…派生类数据成员初始化 }

在类中对派生类构造函数作声明时，不包括基类构造函数名及其参数表列。在定义函数时才将它列出。

构造函数初始化表

可以用初始化表同时对基类和派生类的数据成员进行初始化。

```
Student1(int n, string nam,char s,int a, string ad)
:Student(n,nam,s)
    {…派生类数据成员初始化 }

Student1(int n, string nam,char s,int a, string ad)
:Student(n,nam,s),age(a),addr(ad){}
```

11.5.2 有子对象的派生类的构造函数

类的数据成员中还可以包含类对象，称为子对象(subobject)，即对象中的对象。

定义派生类构造函数的一般形式为

派生类构造函数名（总参数表列）：基类构造函数名（参数表列），子对象名(参数表列)

{派生类中新增数成员据成员初始化语句}

即增加对子对象数据成员初始化。

例11.6 包含子对象的派生类的构造函数

```
#include <iostream>
#include <string>
using namespace std;
class Student
{
public:
    Student(int n, string nam )
    { num=n; name=nam;}
    void display( )
    { cout<<"num:"<<num<<endl<<"name:"<<name<<endl;}
protected:
    int num;
    string name;
};
```

```
class Student1: public Student
{
public:
   Student1(int n, string nam,int n1, string nam1,int a, string
 ad):Student(n,nam),monitor(n1,nam1)
   {
      age=a;
      addr=ad;
   }
   void show( )
   {
      cout<<"This student is:"<<endl;
      display();                          //输出num和name
      cout<<"age: "<<age<<endl;
      cout<<"address: "<<addr<<endl<<endl;
   }
```

```
    void show_monitor( )
    {
       cout<<endl<<"Class monitor is:"<<endl;
       monitor.display( );
    }
private:
    Student monitor;                    //定义子对象
    int age;
    string addr;
};
int main( )
{
    Student1 stud1(10010,"Wang-li",10001,"Li-sun",19,"115 Beijing
 Road,Shanghai");
    stud1.show( );
    stud1.show_monitor();
    return 0;
}
```

派生类构造函数的任务应该包括3个部分：

（1） 对基类数据成员初始化；

（2） 对子对象数据成员初始化；

（3） 对派生类数据成员初始化。

派生类构造函数的总参数表列中的参数，应当包括基类构造函数和子对象的参数表列中的参数。基类构造函数和子对象的次序可以是任意的。

11.5.3 多层派生时的构造函数

例11.7 多级派生情况下派生类的构造函数。

```
#include <iostream>
#include<string>
using namespace std;
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( )
   {
      cout<<"num:"<<num<<endl;
      cout<<"name:"<<name<<endl;
   }
```

```
protected:
    int num;
    string name;
};
class Student1: public Student
{
public:
    Student1(int n,char nam[10],int a):Student(n,nam)
    { age=a; }                          //新增数据成员初始化
    void show( )
    {
        display( );
        cout<<"age: "<<age<<endl;
    }
private:
    int age;
};
```

```
class Student2:public Student1   //声明间接公用派生类
{
public:
   Student2(int n, string nam,int a,int
 s):Student1(n,nam,a){score=s;}
   void show_all( )
   { show( ); cout<<"score:"<<score<<endl; }
private:
   int score;
};
int main( )
{
   Student2 stud(10010,"Li",17,89);
   stud.show_all( );
   return 0;
}
```

基类和两个派生类的构造函数的写法:

基类的构造函数首部:

Student(int n, string nam)

派生类Student1的构造函数首部:

Student1(int n, string nam],int a):Student(n,nam)

派生类Student2的构造函数首部:

Student2(int n, string nam,int a,int s) :Student1(n,nam,a)

11.5.4 派生类构造函数的特殊形式

在使用派生类构造函数时，有以下特殊的形式:

(1) 当不需要对派生类新增的成员进行任何初始化操作时，派生类构造函数的函数体可以为空，如

```
Student1(int n, strin nam,int n1, strin nam1)
:Student(n,nam),monitor(n1,nam1) { }
```

此派生类构造函数的作用只是为了将参数传递给基类构造函数和子对象，并在执行派生类构造函数时调用基类构造函数和子对象构造函数。

(2) 如果在基类中没有定义构造函数，或定义了没有参数的构造函数，那么在定义派生类构造函数时可不写基类构造函数。

如果在基类和子对象类型的声明中都没有定义带参数的构造函数，而且也不需对派生类自己的数据成员初始化，则可以不必显式地定义派生类构造函数。

在基类或子对象类型的声明中定义了带参数的构造函数，就必须显式地定义派生类构造函数，并在派生类构造函数中写出基类或子对象类型的构造函数及其参数表。

在基类中既定义无参的构造函数，又定义了有参的构造函数，则在定义派生类构造函数时，既可以包含基类构造函数及其参数，也可以不包含基类构造函数。

11.5.5 派生类的析构函数

派生类不能继承基类的析构函数，需要通过派生类的析构函数去调用基类的析构函数。

在派生类中可以根据需要定义自己的析构函数，用来对派生类中所增加的成员进行清理工作。

11.6 多重继承

一个派生类有两个或多个基类，派生类从两个或多个基类中继承所需的属性。这种行为称为多重继承(multiple inheritance)。

11.6.1 声明多重继承的方法

如果已声明了类A、类B和类C，可以声明多重继承的派生类D:

```
class D:public A,private B,protected C
{类D新增加的成员}
```

11.6.2 多重继承派生类的构造函数

多重继承派生类的构造函数形式与单继承时的构造函数形式基本相同，只是在初始表中包含多个基类构造函数。如

派生类构造函数名(总参数表列):

基类1构造函数(参数表列),

基类2构造函数(参数表列),

基类3构造函数(参数表列)

{派生类中新增数成员据成员初始化语句}

例11.8 多重继承

```
#include <iostream>
#include <string>
using namespace std;
class Teacher                                   //声明类Teacher(教师)
{
public:                                         //公用部分
   Teacher(string nam,int a, string
 t):name(nam),age(a),title(t){}
   void display( )                              //输出教师有关数据
   {
      cout<<"name:"<<name<<endl;
      cout<<"age"<<age<<endl;
      cout<<"title:"<<title<<endl;
   }
protected:                                      //保护部分
   string name; int age; string title;          //职称
};
```

```
class Student                          //定义类Student(学生)
{
public:
   Student(string nam,char s,float sco)          //构造函数
   {
      name1=nam; sex=s; score=sco;
   }
   void display1( )  //输出学生有关数据
   {
      cout<<"name:"<<name1<<endl;
      cout<<"sex:"<<sex<<endl;
      cout<<"score:"<<score<<endl;
   }
protected:              //保护部分
   string name1; char sex; float score;          //成绩
};
```

```
class Graduate:public Teacher,public Student   //多重继承
{
public:
   Graduate(string nam,int a,char s, string t,float sco,float w):
 Teacher(nam,a,t),Student(nam,s,sco),wage(w) { }
   void show( )      //输出研究生的有关数据
   {
      cout<<"name:"<<name<<endl;
      cout<<"age:"<<age<<endl;
      cout<<"sex:"<<sex<<endl;
      cout<<"score:"<<score<<endl;
      cout<<"title:"<<title<<endl;
      cout<<"wages:"<<wage<<endl;
   }
private:
   float wage;                    //工资
};
```

```
int main( )
{
    Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);
    grad1.show( );
    return 0;
}
```

在两个基类中分别用name和name1来代表姓名，其实这是同一个人的名字。

解决方法: 在两个基类中可以都使用同一个数据成员名name，而在show函数中引用数据成员时指明其作用域，如

```
cout<<"name:"<<Teacher::name<<endl;
```

这就是惟一的，不致引起二义性，能通过编译，正常运行。

11.6.3 多重继承引起的二义性问题

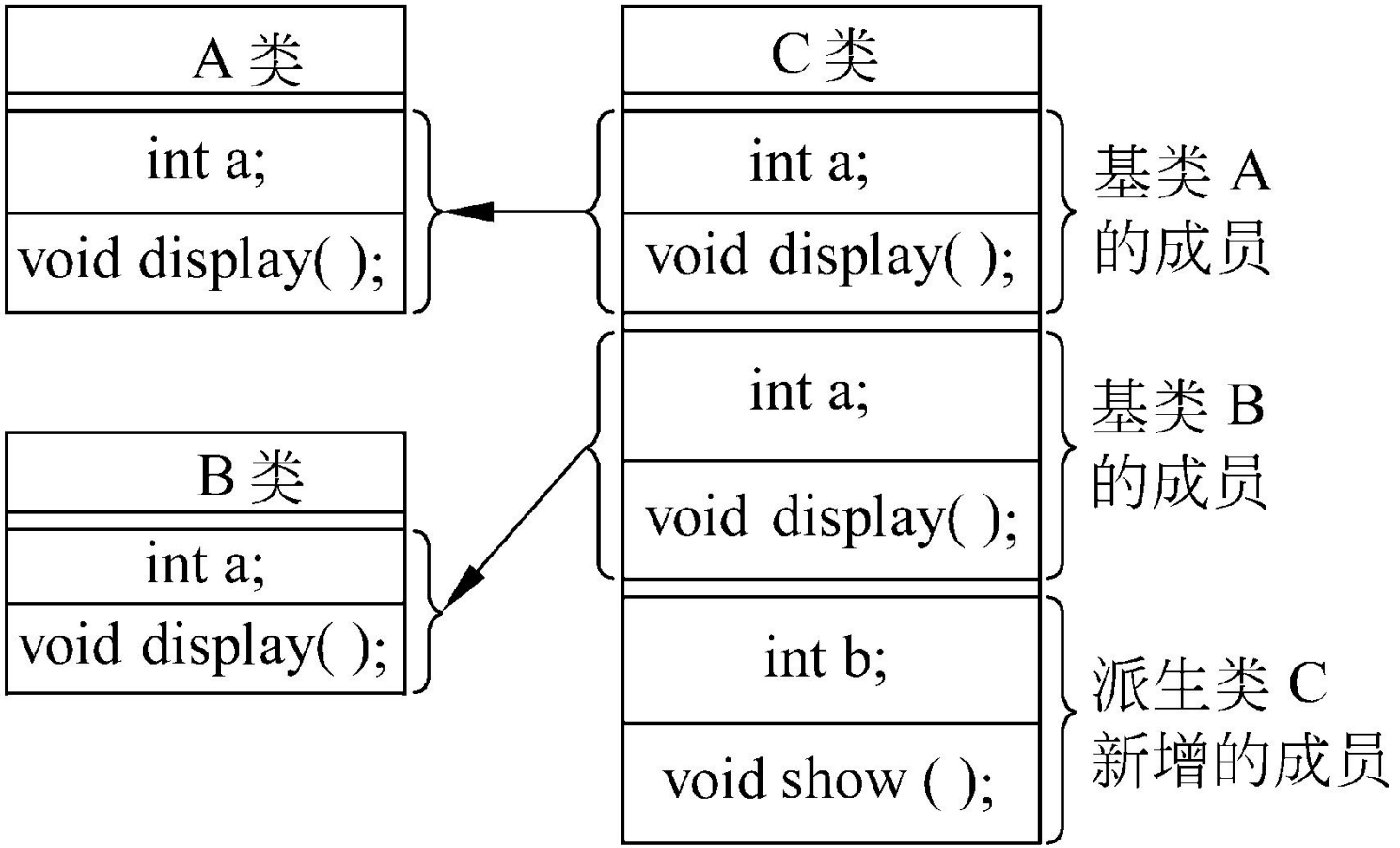
在多重继承时，从不同的基类中会继承一些重复的数据。这就是继承的成员同名而产生的二义性(ambiguous)问题。

如果类A和类B中都有成员函数display和数据成员a，类C是类A和类B的直接派生类。分别讨论下列3种情况:

(1) 两个基类有同名成员

```
class A
{
public:
    int a;
    void dis
};
class B
{
public:
    int a;
    void dis
};
```

class C :public A,public B



如果在main函数中有

C c1; c1.a=3; c1.display();

编译系统无法判别要访问哪一

解决方法是用基类名来限定:

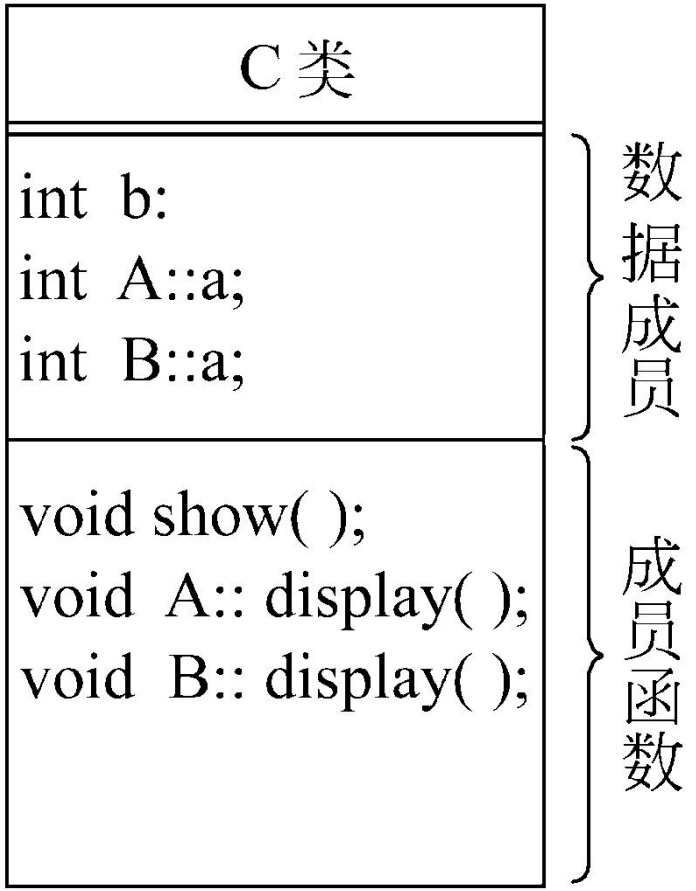
c1.A::a=3;

c1.A::display();

如果通过派生类C的成员函数s display和a，可以直接写成

A::a=3;

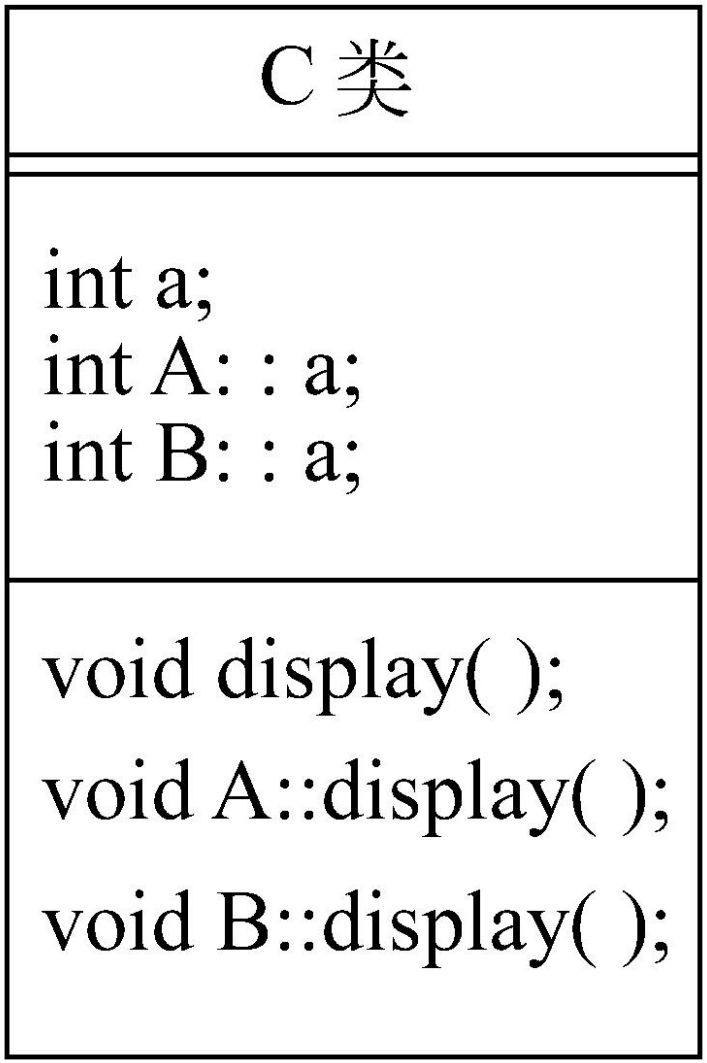
A::display( );



(2) 两个基类和派生类都有同名成员

例如：

```
class C :public A,public B
{
    int a;
    void display();
};
```



同名覆盖

如果在main函数中有:

C c1; c1.a=3; c1.display( );

程序访问的是派生类C中的成员。

规则: 基类的同名成员在派生类中被屏蔽，成为“不可见”的，即派生类新增加的同名成员覆盖了基类中的同名成员，称为同名覆盖。

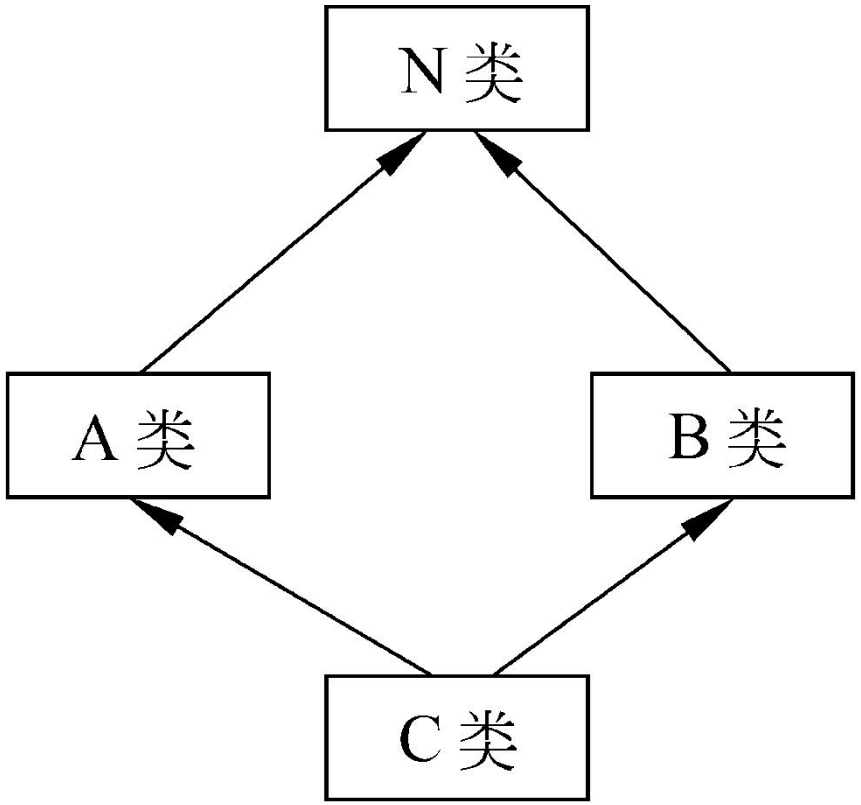
注意: 不同的成员函数，只有在函数名和参数个数相同、类型相匹配的情况下才发生同名覆盖，否则属于函数重载。

要在派生类外访问基类A中的成员，应指明作用域A:

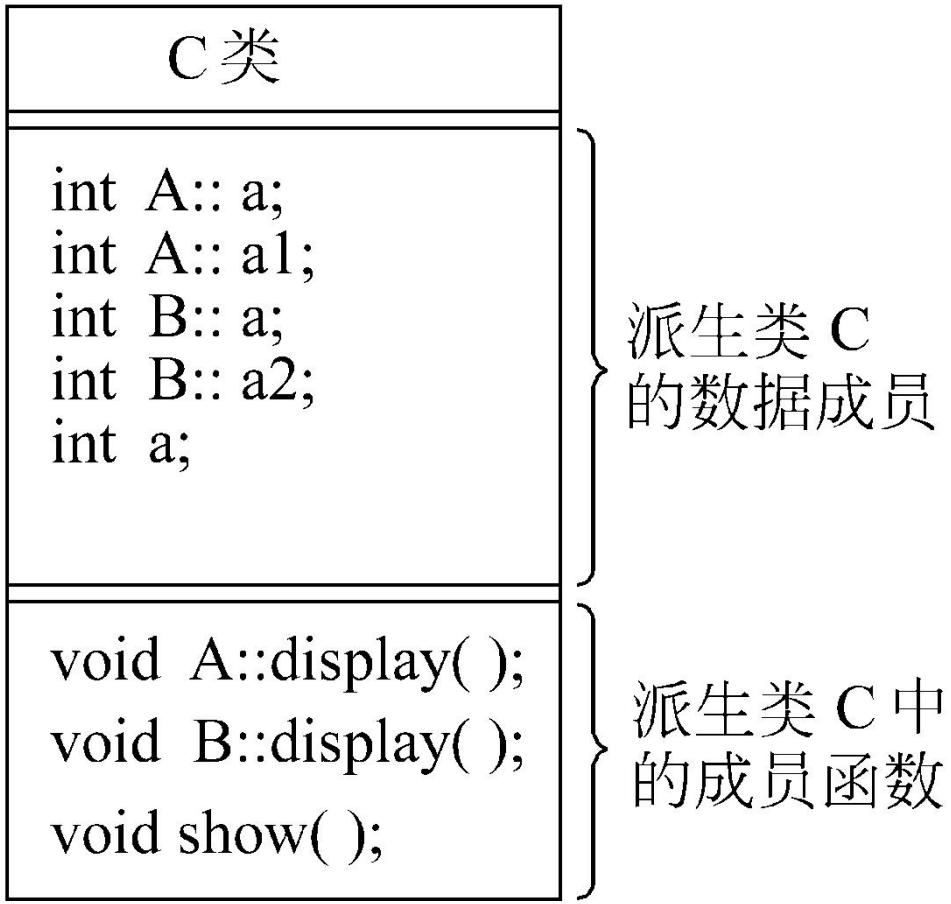
c1.A::a=3； c1.A::display();

(3) 类A和类B从同一个基类派生

```
class N
{
public:
    int a;
    void display()
    {
    cout<<“N::a="<<a<<endl;
     }
};
```



class A:public N




```
class C :public A,public B
{
public:
   int a3;
   void show( )
   { cout<<"a3="<<a3<<endl;}
};

int main( )
{
   C c1;//定义C类对象c1
    ┆
}
```

如何访问类A中从基类N继承下来的成员？

c1.a=3; c1.display( ); //错误

c1.N::a=3; c1.N::display(); //错误

均无法区别是类A中从基类N继承下来的成员，还是类B中从基类N继承下来的成员。

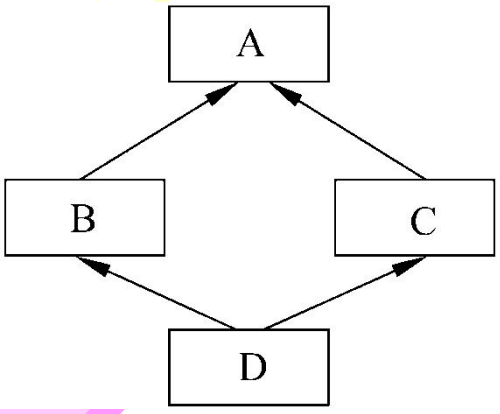
应当通过类N的直接派生类名来指出要访问的是类N的哪一个派生类中的基类成员。如

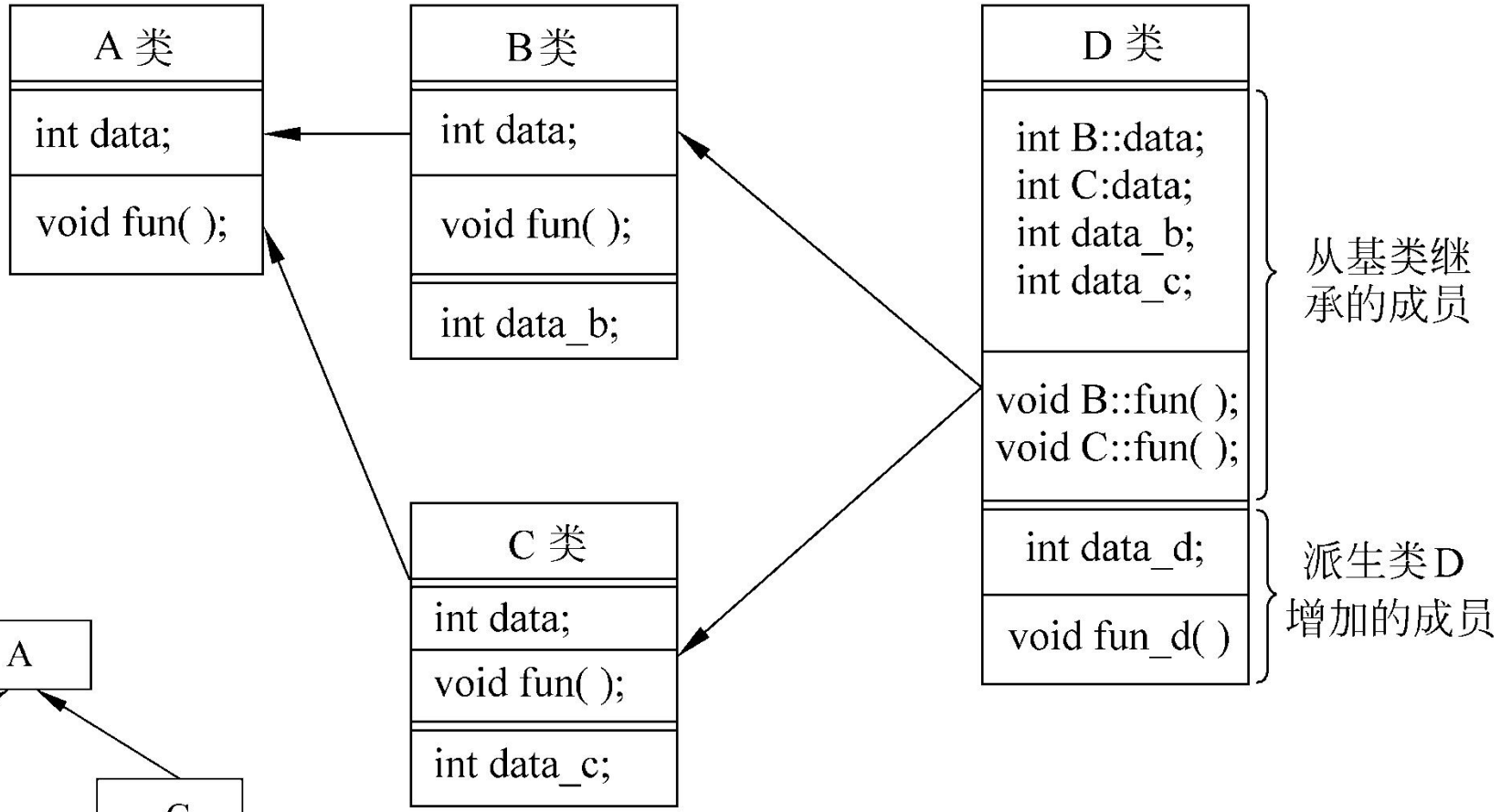
c1.A::a=3; c1.A::display();

11.6.4 虚基类

1. 虚基类的作用

C++提供虚基类(virtual base class)的方法，使得在继承间接共同基类时只保留一份成员。


A
B
C
D


A 类
int data;
void fun( );
B类
int data;
void fun( );
int data_b;
C 类
int data;
void fun( );
int data_c;
D 类
int B::data;
int C:data;
int data_b;
int data_c;
void B::fun( );
void C::fun( );
int data_d;
void fun_d( )
从基类继承的成员
派生类D增加的成员

(b) (c)

将类A声明为虚基类

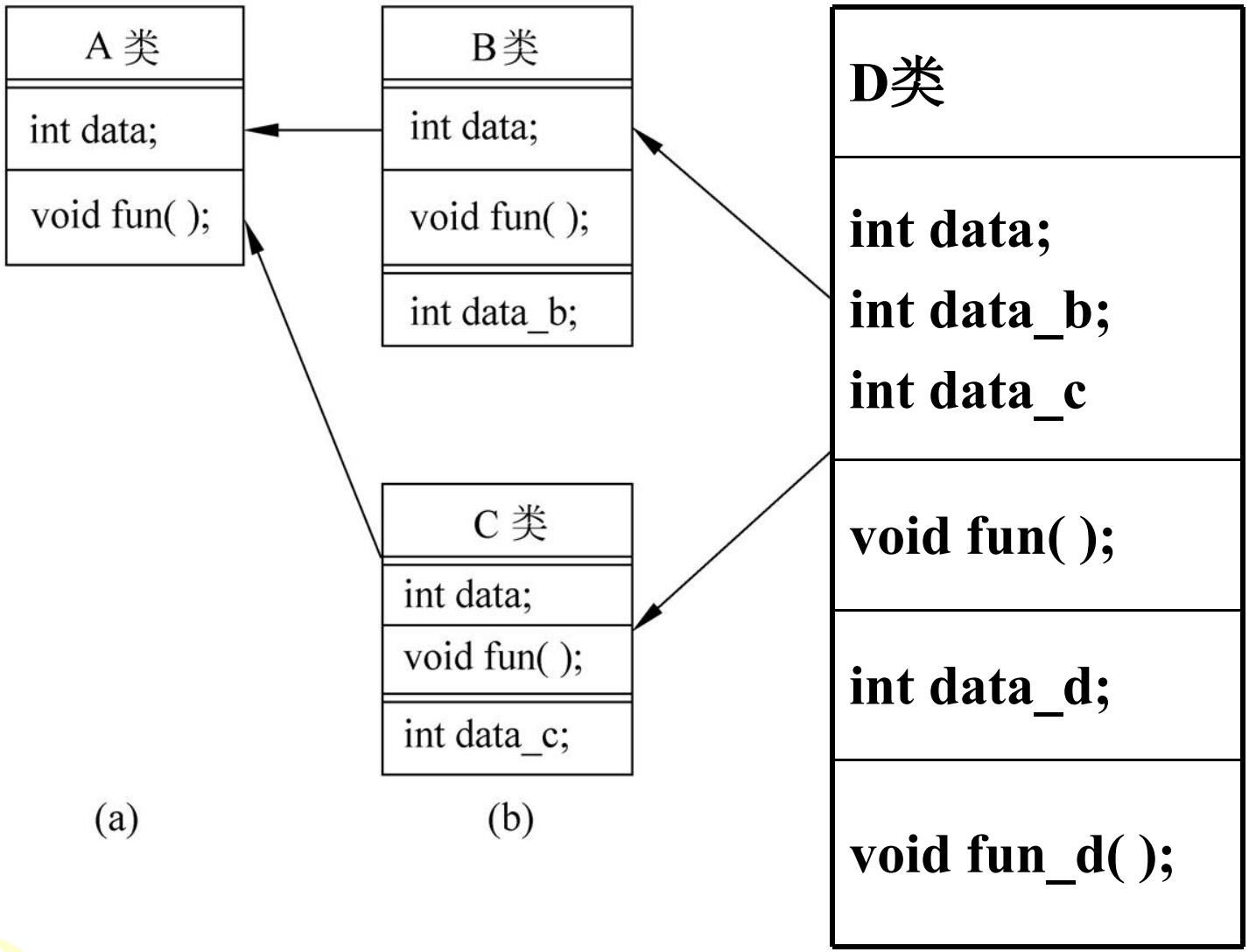
虚基类是在指定派生类继承方式时声明的。

```
class A                                  //声明基类A
 {…};
class B :virtual public A       //声明A是B的虚基类
 {…};
class C :virtual public A       //声明A是C的虚基类
 {…};
```

声明虚基类的一般形式为

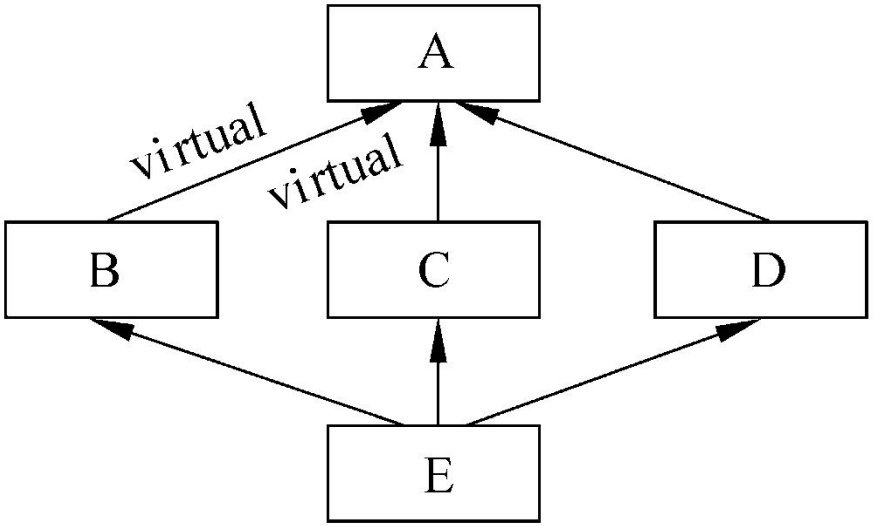
class 派生类名: virtual 继承方式 基类名

当虚基类通过多条派生路径被一个派生类继承时，该派生类只继承该基类一次。


A 类
int data;
void fun( );
B类
int data;
void fun( );
int data_b;
C 类
int data;
void fun( );
int data_c;
(a)
(b)
D类
int data;
int data_b;
int data_c
void fun( );
int data_d;
void fun_d( );

注意: 为了保证虚基类在派生类中只继承一次，应当在该基类的所有直接派生类中声明为虚基类。

在派生类E中，虽然从类B和C路径派生的部分只保留一份基类成员，但从类D路径派生的部分还保留一份基类成员。



2. 虚基类的初始化

如果虚基类中定义了带参数的构造函数，无默认构造函数，则应在派生类构造函数中对虚基类进行初始化。

```
class A                              //定义基类A
{A(int i){ }                         //基类构造函数
   …
};
class B :virtual public A            //A作为B的虚基类
{B(int n):A(n){ }                    //B类构造函数
   …
};
class C :virtual public A            //A作为C的虚基类
{C(int n):A(n){ }                    //C类构造函数
   …
};
```

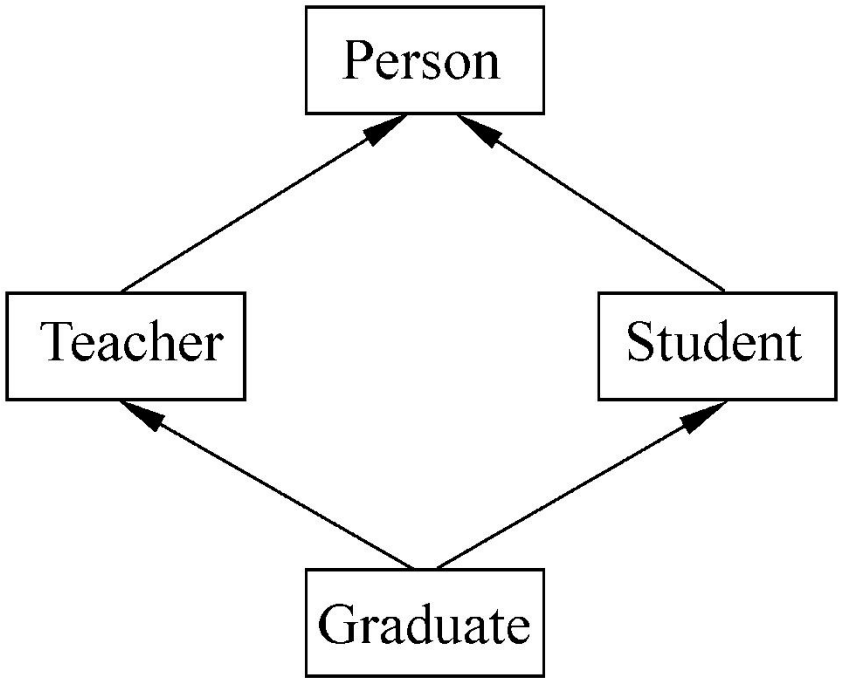
```
class D :public B,public C    //D类构造函数
{D(int n):A(n),B(n),C(n){ }
   …
};
```

规定: 在最后的派生类中不仅要负责对其直接基类进行初始化，还要负责对虚基类初始化。

C++编译系统只执行最后的派生类对虚基类的构造函数的调用，保证虚基类的数据成员不会被多次初始化。

3. 虚基类的简单应用举例

例11.9 在例11.8的基础上，在Teacher类和Student类之上增加一个共同的基类Person。



```
#include <iostream>
#include <string>
using namespace std;
class Person                                    //声明公共基类Person
{
public:
    Person(string nam,char s,int a)             //构造函数
    { name=nam;sex=s;age=a;}
protected:                                      //保护成员
    string name;
    char sex;
    int age;
};
```

```
class Teacher:virtual public Person
{
public:
    Teacher(string nam,char s,int a,string t):
 Person(nam,s,a),title(t){}
protected:                                  //保护成员
    string title;                           //职称
};
class Student:virtual public Person
{
public:
    Student(string nam,char s,int a,float
 sco) :Person(nam,s,a),score(sco){ }
protected:                                  //保护成员
    float score;                            //成绩
};
```

```
class Graduate:public Teacher,public Student
{
public:
   Graduate(string nam,char s,int a, string t,float sco,float w):
Person(nam,s,a),Teacher(nam,s,a,t),Student(nam,s,a,sco),wage
 (w){}
   void show( );
private:
   float wage;                    //工资
};
```

```
void Graduate::show( )
{
    cout<<"name:"<<name<<endl;
    cout<<"age:"<<age<<endl;
    cout<<"sex:"<<sex<<endl;
    cout<<"score:"<<score<<endl;
    cout<<"title:"<<title<<endl;
    cout<<"wages:"<<wage<<endl;
}

int main( )
{
    Graduate grad1("Wang-li",'f',24,"assistant",89.5,1234.5);
    grad1.show( );
    return 0;
}
```

不要提倡在程序中使用多重继承；
能用单一继承解决的问题就不要使用多重继承。

有些面向对象的程序设计语言(如Java，Smalltalk)并不支持多重继承。

11.7 基类与派生类的转换

只有公用派生类才是基类真正的子类型，它完整地继承了基类的功能。

基类与派生类对象之间有赋值兼容关系，由于派生类中包含从基类继承的成员，因此可以将派生类的值赋给基类对象，在用到基类对象的时候可以用其子类对象代替。

1. 派生类对象向基类对象赋值

```
A a1;          //定义基类A对象a1
B b1;          //定义类A的公用派生类B的对象b1
a1=b1;         //用派生类B对象b1对基类对象a1赋值
```

在赋值时舍弃派生类自己的成员。

注:只能用子类对象对基类对象赋值，反之则不行。

注: 赋值后不能企图通过对象a1去访问派生类对象b1的新增成员。

2. 派生类对象向基类对象的引用赋值或初始化

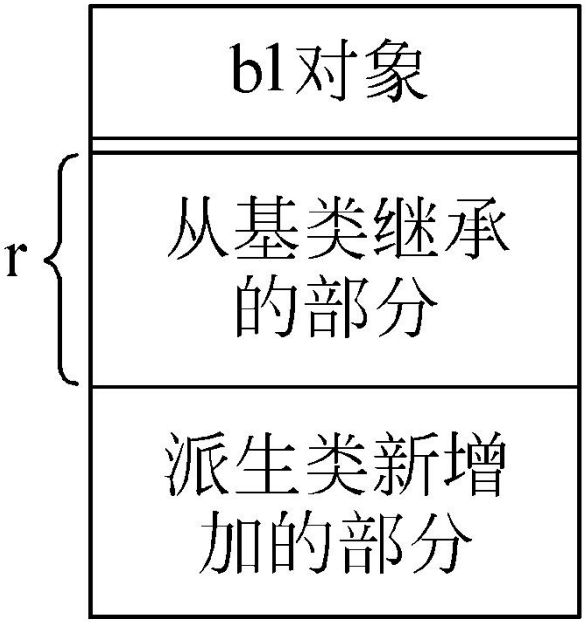
```
A a1;       //定义基类A对象a1
B b1;       //定义公用派生类B对象b1

A& r=b1; //定义基类A对象引用r，用b1对其初始化

A& r=a1; //定义基类A对象引用r，用a1对其初始化
r=b1;       //再用派生类B对象b1对a1的引用r赋值
```

注：两种方法的机理是不同的。

注意: 此时r并不是b1的别名，也不与b1共享同一段存储单元。它只是b1中基类部分的别名，r与b1中基类部分共享同一段存储单元。

r与b1具有相同的起始地址。

与实验结果不同。



指向派生类对象的基类引用的地址

```
int main( )
{
    Person person1[]={Person("Zhang-lin",'f',24),Person("Zhang-lin",'f',25)};
   Graduate grad1[]={Graduate("Wang-li",'f',24,"assistant",89.5,1234.5),Graduate("Wang-li",'f',24,"assistant",89.5,1234.5)};
   //Person & r=person1[0];
    //r=grad1[0];
    Person & r=grad1[0];
    cout<<&person1[0]<<"  "<<&person1[1]<<" "<<sizeof(person1[0])<<endl;
    cout<<&grad1[0]<<"  "<<&grad1[1]<<"  "<<sizeof(grad1[0])<<endl;
    cout<<&r<<endl;
   return 0;
}
```

3. 函数的参数是基类对象引用，相应的实参可以用子类对象。如

```
void fun(A& r)          //形参是类A的对象的引用
 {cout<<r.num<<endl;}
```

在调用fun函数时可以用派生类B的对象b1作实参:

```
fun(b1);
```

输出类B的对象b1的基类数据成员num的值。

与前相同，在fun函数中只能输出派生类中基类成员的值。

4. 派生类对象的地址可以赋给指向基类对象的指针变量，也就是说，指向基类对象的指针变量也可以指向派生类对象。

例11.10 定义一个基类Student(学生)，再定义Student类的公用派生类Graduate(研究生)，用指向基类对象的指针输出数据。

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

class Student                          //声明Student类
{
public:
    Student(int, string,float); //声明构造函数
    void display( );
private:
    int num;
    string name;
    float score;
};
```

```
Student::Student(int n, string nam,float s)
{ num=n; name=nam; score=s;}
void Student::display( )
{
    cout<<endl<<"num:"<<num<<endl;
    cout<<"name:"<<name<<endl;
    cout<<"score:"<<score<<endl;
}
class Graduate:public Student
{
public:
    Graduate(int, string ,float,float);        //声明构造函数
    void display( );                           //声明输出函数
private:
    float pay;                                 //工资
};
```

```
Graduate::Graduate(int n, string nam,float s,float
 p):Student(n,nam,s),pay(p){ }
void Graduate::display()
{
    Student::display();
    cout<<"pay="<<pay<<endl;
}
int main()
{
    Student stud1(1001,"Li",87.5);
    Graduate grad1(2001,"Wang",98.5,563.5);
    Student *pt=&stud1;
    pt->display( );              //调用stud1.display函数
    pt=&grad1;                   //指针指向grad1
    pt->display( );              //调用grad1.display函数
}
```

注意：pt是指向Student类对象的指针变量，即使让它指向了grad1，但实际上pt指向的是grad1中从基类继承的部分。

通过指向基类对象的指针，只能访问派生类中的基类成员，而不能访问派生类增加的成员。

pt->display()调用的是基类的display函数，所以只输出研究生grad1的num,name,score3个数据。

结论： 用指向基类对象的指针变量指向子类对象是合法的、安全的，不会出现编译上的错误。

但在应用上却不能完全满足人们的希望，人们有时希望通过使用基类指针能够调用基类和子类对象的成员。

解决这个问题的办法是使用虚函数。

11.8 继承与组合

对象成员的类型可以是派生类的基类，也可以是另外一个已定义的类。

在一个类中以另一个类的对象作为数据成员的，称为类的组合(composition)。

类的组合和继承都是软件重用的重要方式。但二者的概念和用法不同。

继承是纵向的，组合是横向的。

```
class Teacher                    //教师类
{ …};
class BirthDate                  //生日类
{ …};
class Professor:public Teacher      //教授类
{ public:
    ┆
  private:
  BirthDate birthday;
};
void fun1(Teacher &);
void fun2(BirthDate &);
```

若在main函数中调用这两个函数:

fun1(prof1);//形参为基类引用，实参为子类对象

fun2(prof1.birthday);//正确，实参与形参类型相同

fun2(prof1);//错误，实参与形参不匹配

如果修改了成员类的部分内容，只要成员类的公用接口不变，组合类可以不修改，但需要重新编译。

11.9 继承在软件开发中的重要意义

- 继承使软件的重用成为可能，许多厂商开发了各类实用的类库，类库的出现使得软件的重用更加方便。

- 类库常随C++编译系统卖给用户，不是C++编译系统的一部分。

- 不同的C++编译系统提供的由不同厂商开发的类库一般是不同的。

- 类库中类的声明一般放在头文件中，类的实现是单独编译的，以目标代码形式存放。
- 在程序编译时只需对派生类新增的功能进行编译，大大提高了调试程序的效率。
- 如果修改了基类，只要基类的公用接口不变，派生类不必修改，但基类需要重新编译，派生类也必须重新编译，否则不起作用。

使用继承机制的原因

- (1) 基类是多个多个程序使用的。
- (2) 用户往往得不到基类的源代码。
- (3) 类库中的基类与多种组件建立了某种关系，因此是不容许修改的。
- (4) 许多基类是专门作为基类设计的。
- (5) 在OOP中，需要设计类的层次结构，从最初的抽象类逐步向着具体实现前进。

作业

- P392：7、8

实验七 疑难解析

```
#include<iostream.h>
class Point
{
private:
   int x1,x2;
public:
   //Point() {x1=0;x2=0;}
   Point(int x,int y) {x1=x;x2=y;}
   int x_cord(){return x1;}
   int y_cord(){return x2;}
};
void main()
{
   Point data(5,6);
   cout<<data.x_cord()<<endl;
   cout<<data.y_cord()<<endl;
   //Point more_data();//Point more_data[20];
}
```