这时，基类的同名成员会在派生类中出现“命名冲突”问题。

解决方法是采用基类名受限访问，访问格式为<基类名>::<基类成员>。

一. 简答题（见课件和课本）

1. 多继承时，如果多个基类中的同名成员在派生类中由于标识符不唯一，将会出现什么问题？在派生类中如何消除该问题？

2. 什么是虚基类？它有什么作用？

虚基类的作用：在多继承路径中,如果直接基类B、C有公共的基类A，就会出现重复继承——即公共基类的数据成员在多继承的派生类D中就有多个拷贝。

为了避免重复继承，就需要使用virtual关键字把公共基类A声明为虚基类，格式为:

```
class B: virtual public A { ... };
class C: virtual public A { ... };
```

虚基类的特点：

(1) 虚基类的构造函数由最新派生出来的类的构造函数调用。

(2) 虚基类的构造函数先于非虚基类的构造函数执行。

二. 简答题

1. 写出以下程序的运行结果

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

class B1 {
   public:
      B1(int i) {  cout << "constructing B1" << i << endl;   }
      ~B1( ) {    cout << "destructing B1" << endl;    }
};

class B2 {
   public:
      B2( ) {    cout << "constructing B3" << endl;    }
      ~B2( ) {    cout << "destructing B3" << endl;    }
};

class C: public B2, virtual public B1 {
      int j;
   public:
      C(int a, int b, int c): B1(a), memberB1(b), j(c) { }
   private:
      B1 memberB1;
      B2 memberB2;
};

int main( ){
      C obj(1,2,3);
}
```

该程序的运行结果为：

```
constructing B11
constructing B3
constructing B12
constructing B3
destructing B3
destructing B1
destructing B3
destructing B1
```

2. 写出以下程序的运行结果

```
#include<iostream.h>
class A {
   public:
      int n;
};
class B: public A {};
class C: public A {};
class D: public B, public C {
      int getn() {    return B::n;    }
};

void main()
{    D d;
     d.B::n = 10;
     d.C::n = 20;
     cout << d.B::n << "," << d.C::n <<endl;
}
```

该程序的运行结果为：

10,20

3. 写出以下程序的运行结果

```
#include <iostream.h>
class A {
      int a;
public:
      A(int i) {    a=i; cout << "constructing class A" << endl;    }
      void print() {    cout << a << endl;    }
      ~A()    {    cout << "destructing class A" << endl;    }
};

class B1: public A {
      int b1;
public:
      B1(int i, int j): A(i) {    b1=j; cout << "constructing class B1" << endl;    }
      void print()
      {
            A::print();
            cout << b1 << endl;
      }
      ~B1() {    cout << "destructing class B1" << endl;    }
};
```

```
class B2: public A {
        int b2;
public:
        B2(int i, int j): A(i) {    b2=j;cout << "constructing class B2" << endl;    }
        void print()
        {
                A::print();
                cout << b2 << endl;
        }
        ~B2() {    cout << "destructing class B2" << endl;    }
};
```

当需要打印或者读取基类的private数据成员时，需要使用基类的public成员函数。

出现重复继承

```
class C: public B1, public B2 {
        int c;
public:
        C(int i, int j, int k, int l, int m) : B1(i,j), B2(k,l), c(m) {
            cout << "constructing class C" << endl;
        }
        void print() {
            B1::print();
            B2::print();
            cout << c << endl;
        }
        ~C( ){    cout << "destructing class C" << endl;    }
};

void main()
{
         C c1(1,2,3,4,5);
         c1.print();
}
```

该程序的运行结果为：

```
constructing class A
constructing class B1
constructing class A
constructing class B2
constructing class C
1
2
3
4
5
destructing class C
```

destructing class B2
destructing class A
destructing class B1
destructing class A

三. 编程题：即教材 P262 第 19 题

```
//请定义一个类 A， 使得在程序中只能创建该类的唯一一个对象，
//当试图创建该类的第二个对象时， 返回第一个对象的指针。
//（提示： 类 A 的设计模式采用 Singleton 模式）

class Singleton {
public:
    static Singleton* Instance();
protected:
    Singleton();
private:
    static Singleton* _instance;
}

Singleton* Singleton::_instance = 0;
Singleton* Singleton::Instance()
{
    if (_instance == 0)
    {
        _instance = new Singleton;
    }
    return _instance;
};

//使用Singleton
void main()
{
    ......

    Singleton* singleton = Singleton::Instance();

    ......
}
```