Object Oriented Programming in C++

Segment-5

Course Code: 2301

Prepared by Sumaiya Deen Muhammad

Lecturer, CSE, IIUC

Contents

Inheritance:

- Defining derived classes,
- Single inheritance,
- multiple inheritance,
- multilevel inheritance,
- Hierarchical inheritance,
- Virtual base classes,
- Constructors in derived classes,
- nesting of classes.

Inheritance

- Inheritance is an essential part of OOP. Its big payoff is that it permits code *reusability*.
- Inheritance is the process of creating new classes, called *derived classes, from* existing or *base classes.*
- The *derived class* inherits the members of the *base class*, on top of which it can add its own members.
- The members of the class can be **Public**, **Private** or **Protected**.
- Classes that are derived from others inherit all the accessible members of the base class. That means that if a base class includes a member A and we derive a class from it with another member called B, the derived class will contain both member A and member B.

Advantage of Inheritance

- The main advantage of Inheritance is, it provides an opportunity to **reuse** the code functionality and fast implementation time.

Syntax of Inheritance

- **Syntax:**

```
class DerivedClass : AccessSpecifier BaseClass
```

- The default access specifier is **Private.**
- Inheritance helps user to create a new class (derived class) from a existing class (base class).
- Derived class inherits all the features from a Base class including additional feature of its own.

Access Control

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

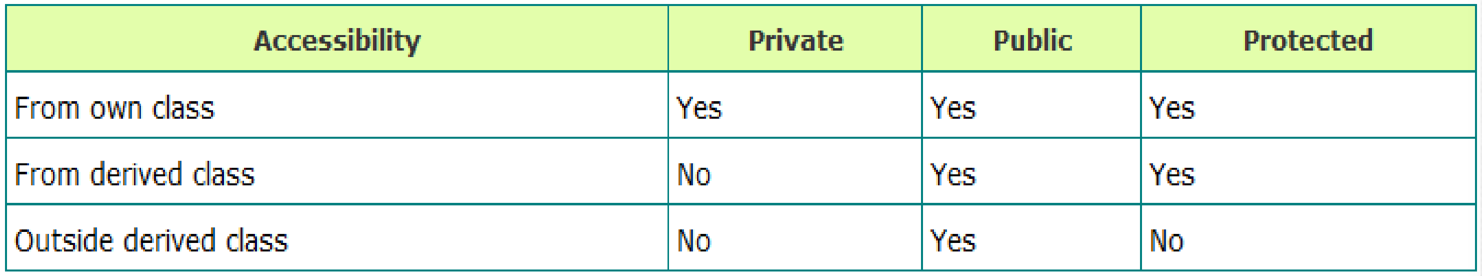
A derived class inherits all base class methods with the following exceptions :

•Constructors, destructors and copy constructors of the base class.

•Overloaded operators of the base class.

•The friend functions of the base class.

Access Control (contd.)



Accessibility	Private	Public	Protected
From own class	Yes	Yes	Yes
From derived class	No	Yes	Yes
Outside derived class	No	Yes	No

Types of Inheritance

- **Following are the types of Inheritance.**

  1. Single Inheritance
  2. Multiple Inheritance
  3. Multilevel Inheritance
  4. Hierarchical Inheritance
  5. Hybrid Inheritance

1. Single Inheritance

- In Single Inheritance, one class is derived from another class.
- It represents a form of inheritance where there is only one base and derived class.

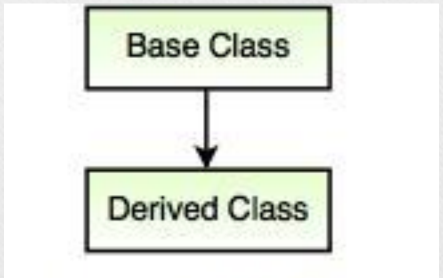
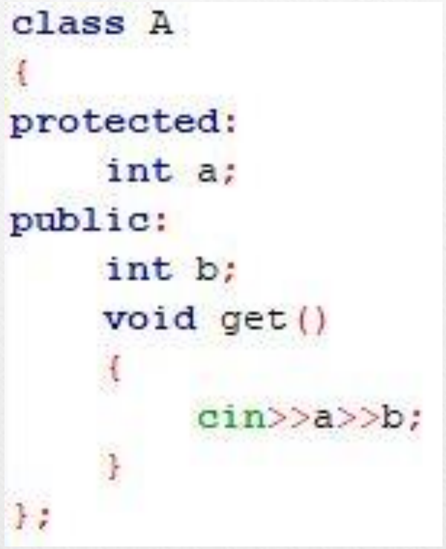
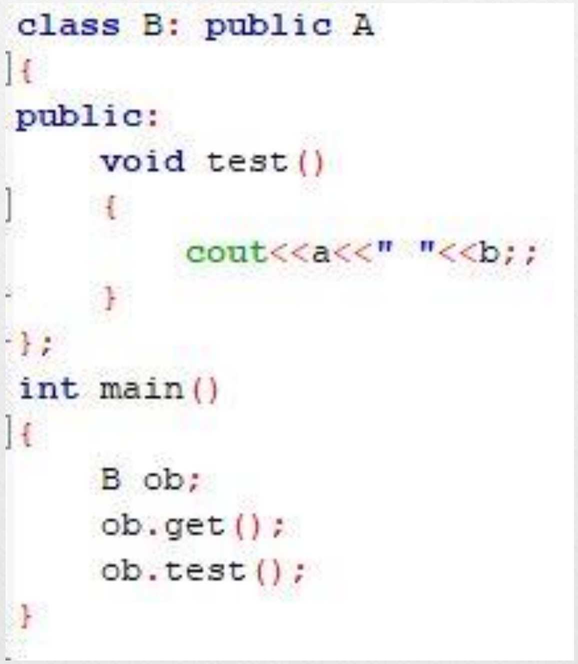


Fig. Single Inheritance

Example: Program demonstrating Single Inheritance



```
class A
{
protected:
      int a;
public:
      int b;
      void get()
      {
            cin>>a>>b;
      }
};
```



```
class B: public A
{
public:
      void test()
      {
            cout<<a<<" "<<b;;
      }
};
int main()
{
      B ob;
      ob.get();
      ob.test();
}
```

2. Multiple Inheritance

- ✓ In Multiple Inheritance, one class is derived from multiple classes.
- ✓ In the above figure, Class D is derived from class A, Class B and Class C.
- ✓ We need to use following **syntax** for deriving a class from multiple classes.
- ✓ class derived_class_name : access_specifier base_classname1, access_specifier base_classname2, access_specifier base_classname3

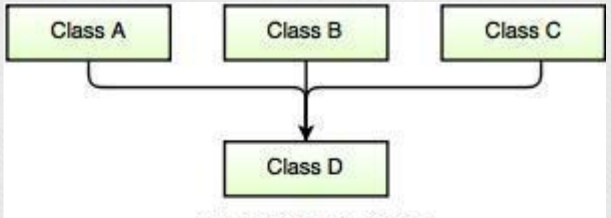
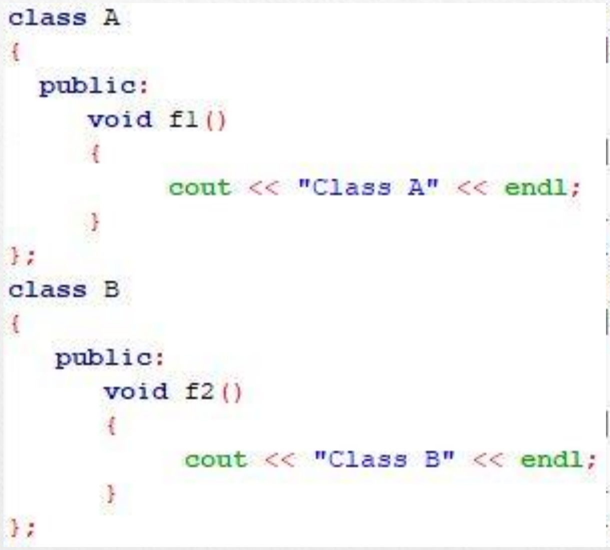
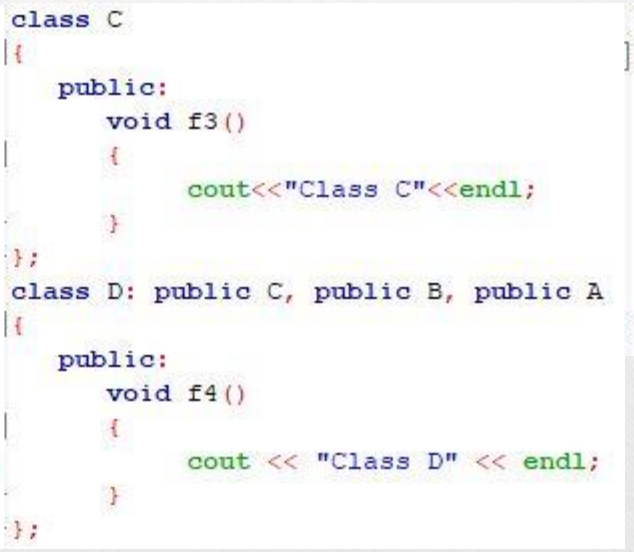


Fig. Multiple Inheritance

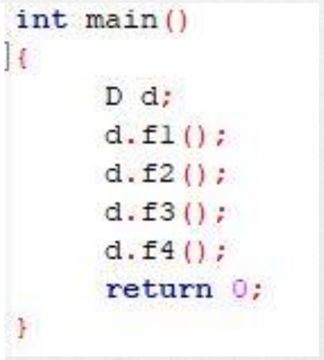
Multiple Inheritance Example



```
class A
{
  public:
     void f1()
     {
          cout << "Class A" << endl;
     }
};
class B
{
   public:
      void f2()
      {
           cout << "Class B" << endl;
      }
};
```



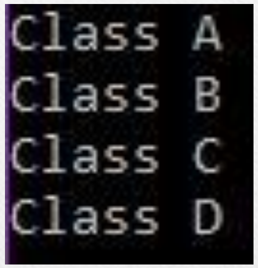
```
class C
{
   public:
      void f3()
      {
           cout<<"Class C"<<endl;
      }
};
class D: public C, public B, public A
{
   public:
      void f4()
      {
           cout << "Class D" << endl;
      }
};
```



```
int main()
{
     D d;
     d.f1();
     d.f2();
     d.f3();
     d.f4();
     return 0;
}
```

Multiple Inheritance Example contd.

- **Output:**



```
Class A
Class B
Class C
Class D
```

3. Multilevel Inheritance

- In Multilevel inheritance, one class is derived from a class which is also derived from another class.
- It represents a type of inheritance when a derived class is a base class for another class.

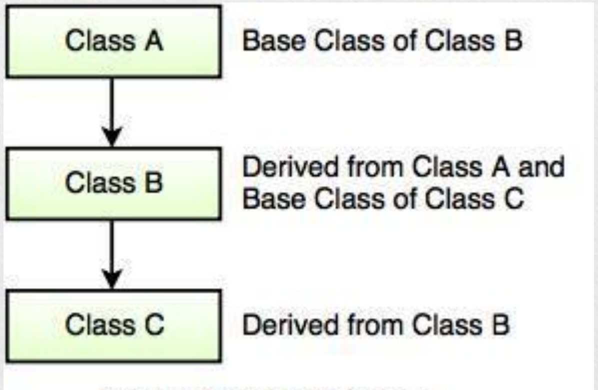
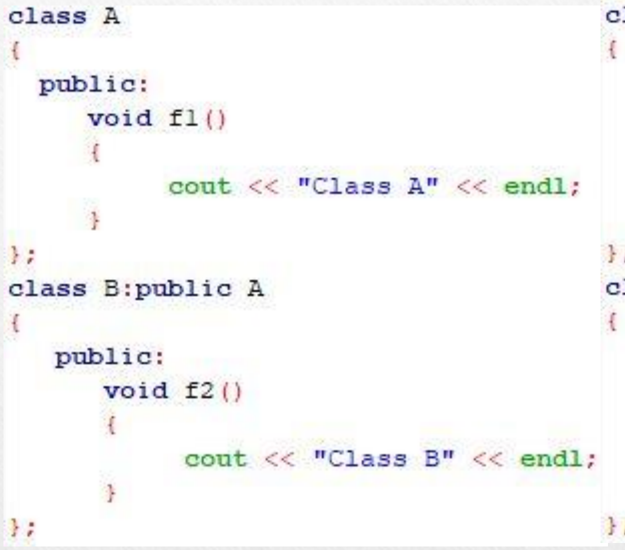
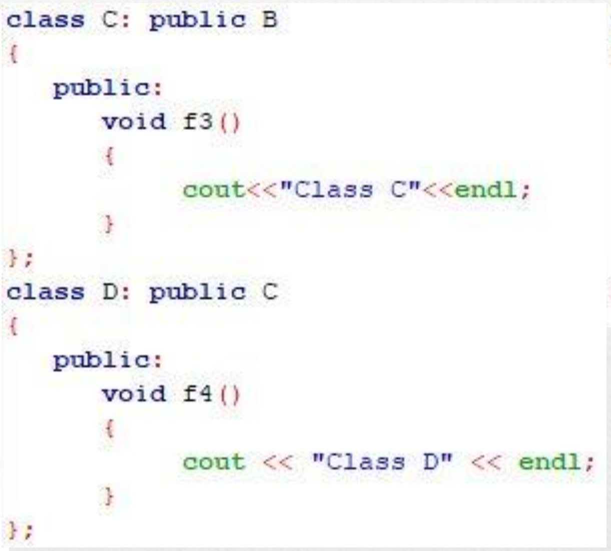


Fig. Multilevel Inheritance

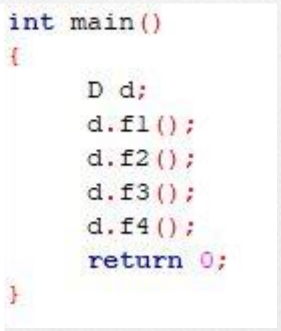
Multilevel Inheritance Example



```
class A
{
  public:
     void f1()
     {
          cout << "Class A" << endl;
     }
};
class B:public A
{
   public:
      void f2()
      {
           cout << "Class B" << endl;
      }
};
```



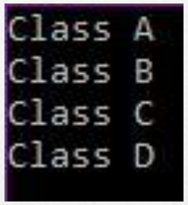
```
class C: public B
{
   public:
      void f3()
      {
           cout<<"Class C"<<endl;
      }
};
class D: public C
{
   public:
      void f4()
      {
           cout << "Class D" << endl;
      }
};
```



```
int main()
{
     D d;
     d.f1();
     d.f2();
     d.f3();
     d.f4();
     return 0;
}
```

Multilevel Inheritance Example (contd.)

- **Output:**



```
Class A
Class B
Class C
Class D
```

- In the above program, **Class D** is derived from **Class C. Class C** is derived from **Class B. Class B** is derived from **Class A. Object d** is created for **Class D** in **main()** function. When the **show()** function is called, **Class A** is executed **show()** because there is no **show()** function **Class C** and **Class B.** At the time of execution the program first looks for **f1()** function in **Class D**, but cannot find it. Then looks in **Class C** because **Class D** is derived from **Class C** and again cannot find it. Then looks in **Class B** because **Class C** is derived from **Class B** and again cannot find it. Finally looks for **f1()** function in **Class A** and executes it and displays the output. Thus multilevel inheritance works.

4. Hierarchical Inheritance

- In Hierarchical Inheritance, there are multiple classes derived from one class.
- In this case, multiple derived classes allow to access the members of one base class.

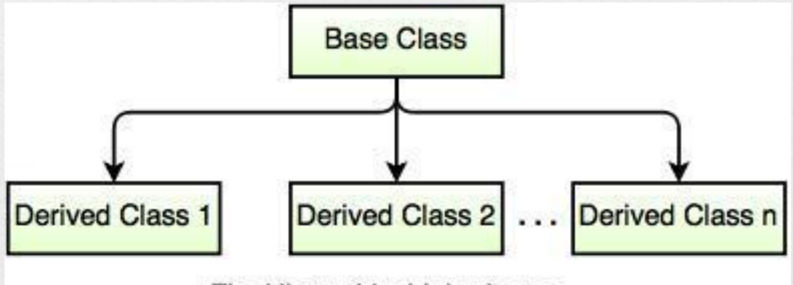
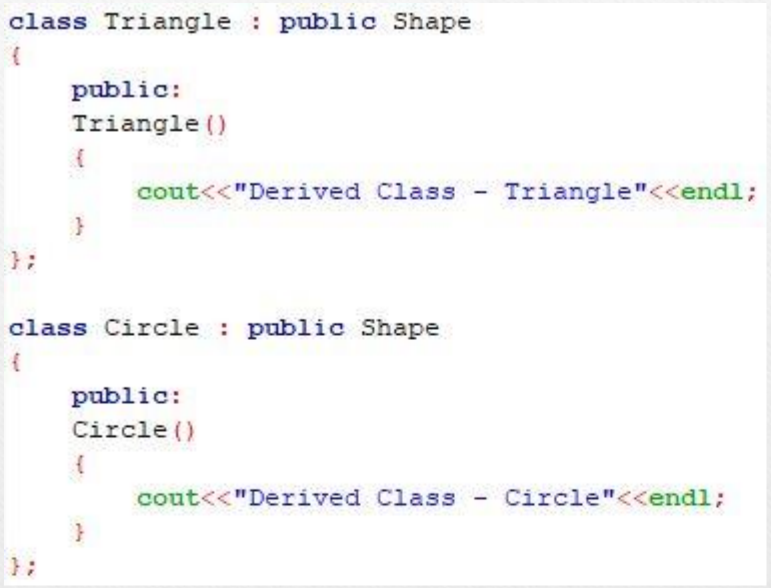


Fig. Hierarchical Inheritance

Hierarchical Inheritance Example

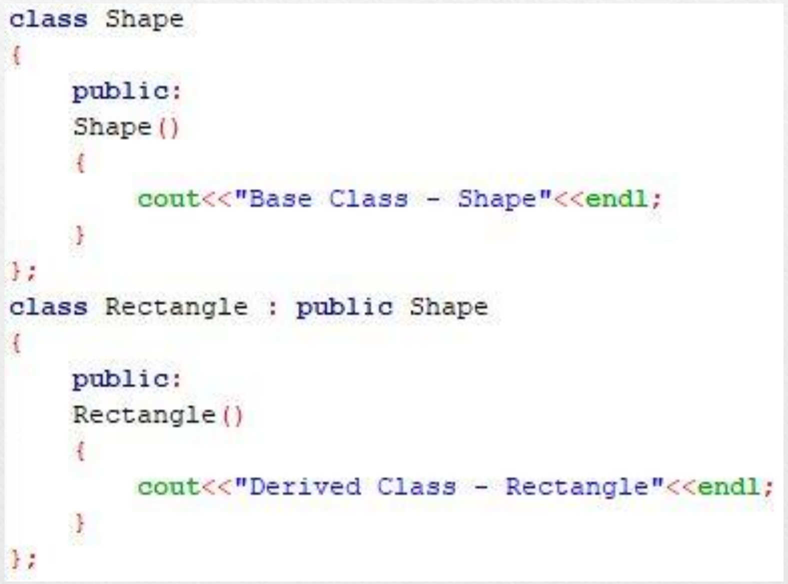
```
class base_class
{
  . . .
};
class first_derived_class : access_specifier base_class
{
  . . .
};
class second_derived_class : access_specifier base_class
{
  . . .
};
class third_derived_class : access_specifier base_class
{
  . . .
};
```

Hierarchical Inheritance Example (contd.)



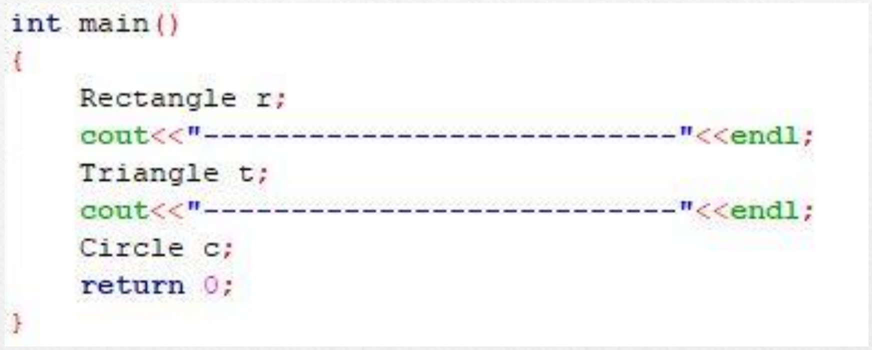
```
class Triangle : public Shape
{
    public:
    Triangle()
    {
        cout<<"Derived Class - Triangle"<<endl;
    }
};

class Circle : public Shape
{
    public:
    Circle()
    {
        cout<<"Derived Class - Circle"<<endl;
    }
};
```



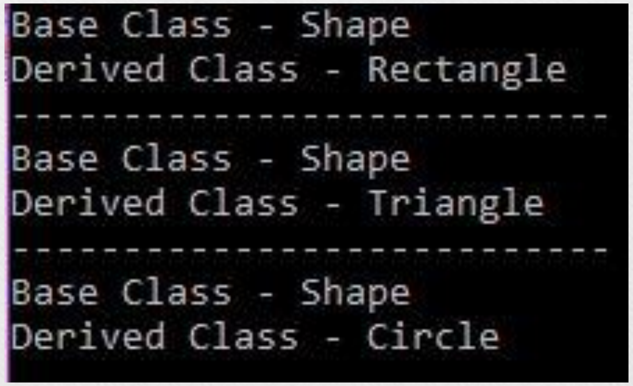
```
class Shape
{
    public:
    Shape()
    {
        cout<<"Base Class - Shape"<<endl;
    }
};
class Rectangle : public Shape
{
    public:
    Rectangle()
    {
        cout<<"Derived Class - Rectangle"<<endl;
    }
};
```

Hierarchical Inheritance Example (contd.)



```
int main()
{
    Rectangle r;
    cout<<"----------------------------"<<endl;
    Triangle t;
    cout<<"----------------------------"<<endl;
    Circle c;
    return 0;
}
```

Output:

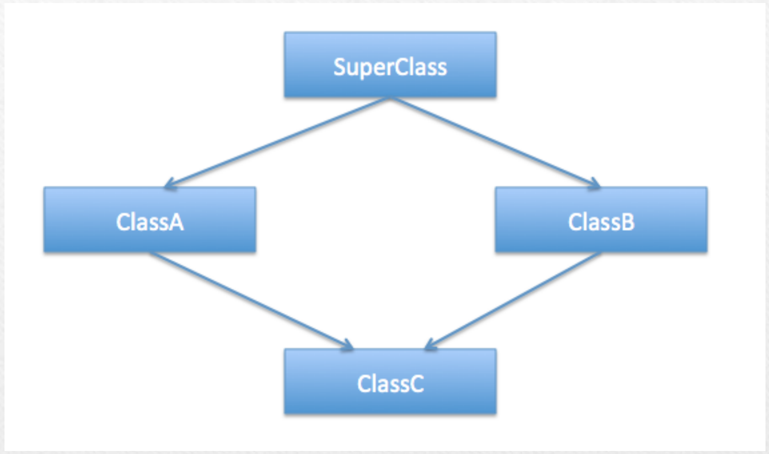


```
Base Class - Shape
Derived Class - Rectangle
----------------------------
Base Class - Shape
Derived Class - Triangle
----------------------------
Base Class - Shape
Derived Class - Circle
```

5. Hybrid Inheritance

- Hybrid inheritance is also known as *Virtual Inheritance.*
- It is a combination of two or more inheritance.
- In hybrid inheritance, when derived class have multiple paths to a base class, a diamond problem occurs. It will result in duplicate inherited members of the base class.
- To avoid this problem easily, we use **Virtual Base Class**. In this case, derived classes should inherit base class by using Virtual Inheritance.

Diamond Problem



Virtual Base Class

Suppose we have two derived classes B and C that have a common base class A, and we also have another class D that inherits from B and C. We can declare the base class A as *virtual* to ensure that B and C share the same subobject of A.

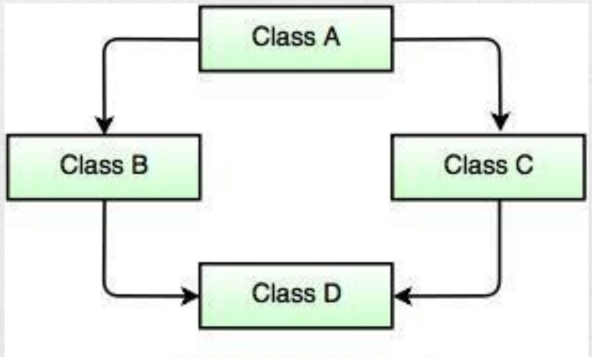
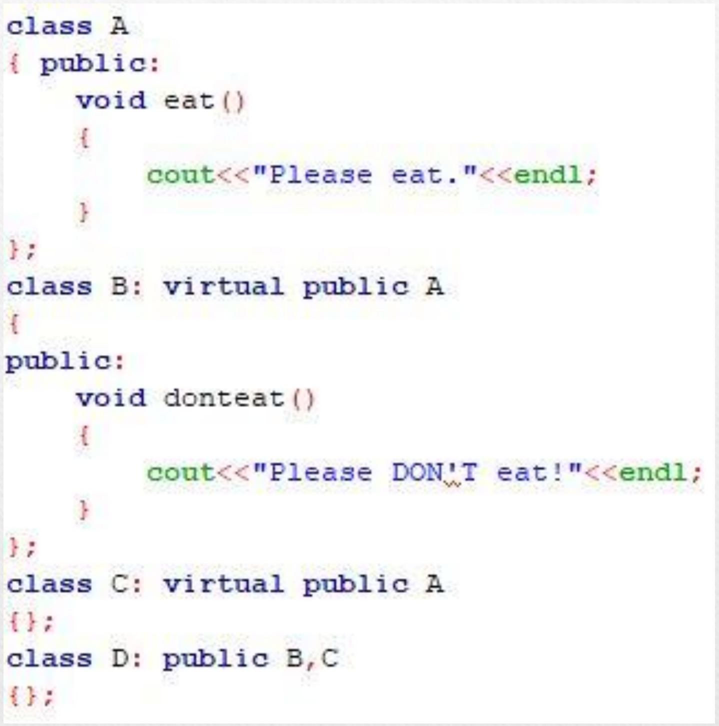
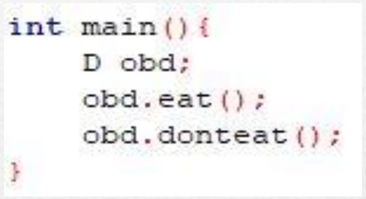


Fig. Hybrid Inheritance

Diamond Problem Solution

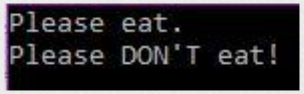


```
class A
{ public:
    void eat()
    {
        cout<<"Please eat."<<endl;
    }
};
class B: virtual public A
{
public:
    void donteat()
    {
        cout<<"Please DON'T eat!"<<endl;
    }
};
class C: virtual public A
{};
class D: public B,C
{};
```



```
int main(){
    D obd;
    obd.eat();
    obd.donteat();
}
```

Output:

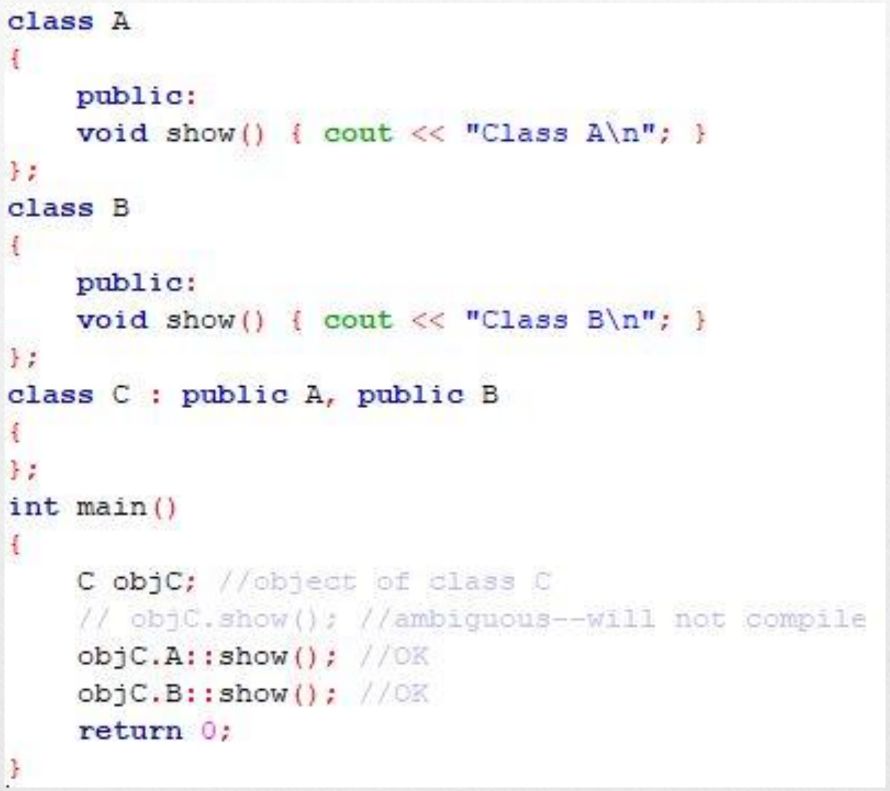


```
Please eat.
Please DON'T eat!
```

Ambiguity in Multiple Inheritance

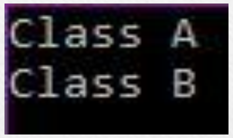
- Two base classes have functions with the same name, while a class derived from both base classes. How do objects of the derived class access the correct base class function? The name of the function alone is insufficient, since the compiler can't figure out which of the two functions is meant.

Solving Ambiguity in Multiple Inheritance



```
class A
{
    public:
    void show() { cout << "Class A\n"; }
};
class B
{
    public:
    void show() { cout << "Class B\n"; }
};
class C : public A, public B
{
};
int main()
{
    C objC; //object of class C
    // objC.show(); //ambiguous--will not compile
    objC.A::show(); //OK
    objC.B::show(); //OK
    return 0;
}
```

Output:



```
Class A
Class B
```

Ambiguity in Multiple Inheritance (contd.)

The problem is resolved using the scope-resolution operator to specify the class in which the function lies. Thus

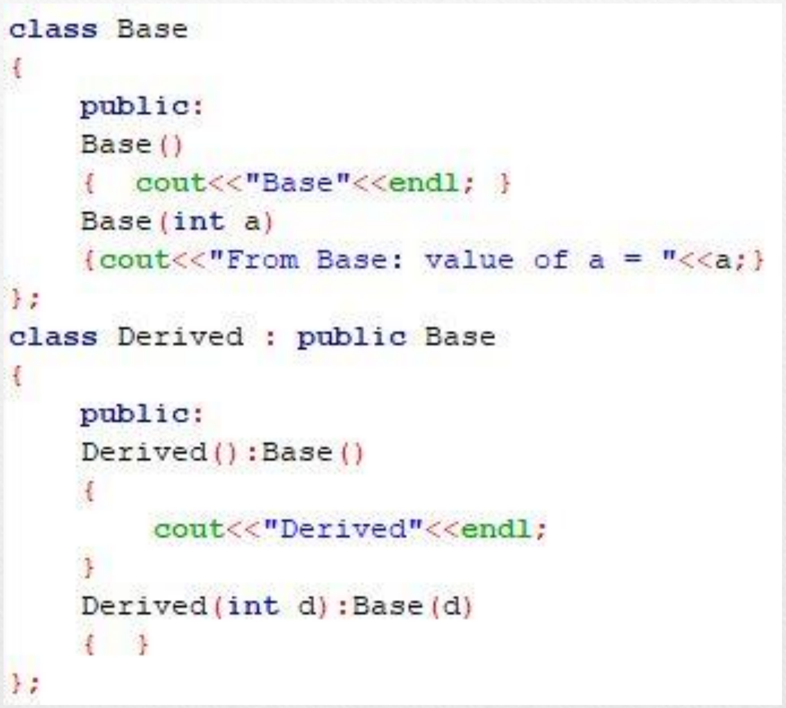
```
objC.A::show();
```

refers to the version of show() that's in the A class, while

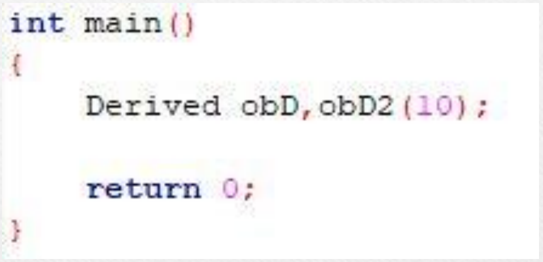
```
objC.B::show();
```

refers to the function in the B class.

Constructors in derived classes



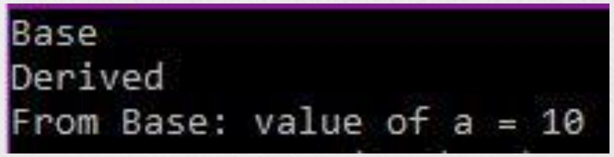
```
class Base
{
    public:
    Base()
    {   cout<<"Base"<<endl; }
    Base(int a)
    {cout<<"From Base: value of a = "<<a;}
};
class Derived : public Base
{
    public:
    Derived():Base()
    {
        cout<<"Derived"<<endl;
    }
    Derived(int d):Base(d)
    {   }
};
```



```
int main()
{
    Derived obD,obD2(10);

    return 0;
}
```

Output:

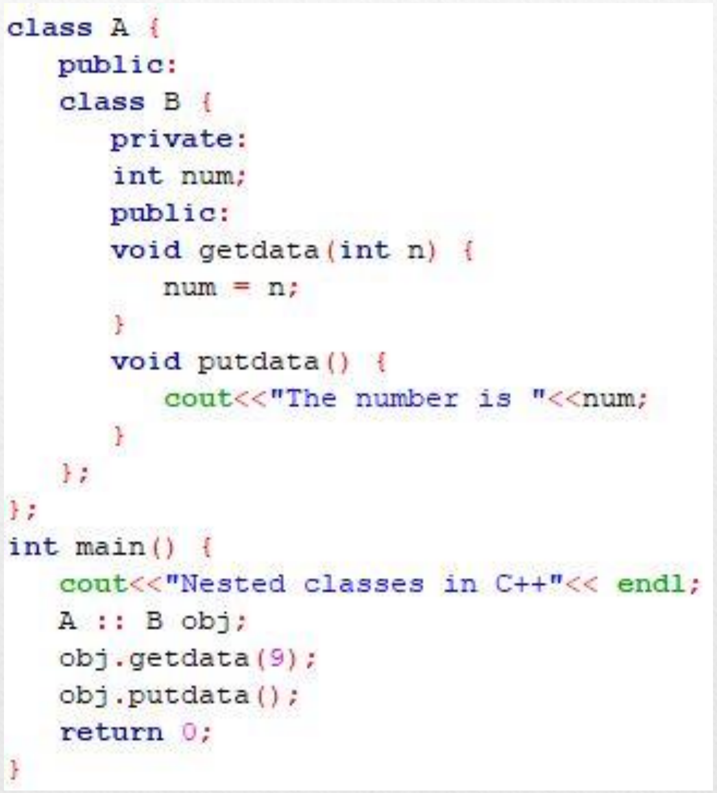


```
Base
Derived
From Base: value of a = 10
```

Nesting of Classes

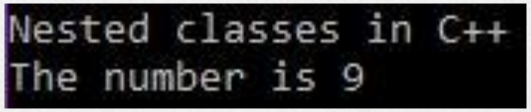
- A nested class is a class which is declared in another enclosing class.
- A nested class is a member and as such has the same access rights as any other member.
- The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed.
- A *nested class* is declared within the scope of another class.

Nesting of Classes (contd.)



```
class A {
   public:
   class B {
      private:
      int num;
      public:
      void getdata(int n) {
         num = n;
      }
      void putdata() {
         cout<<"The number is "<<num;
      }
   };
};
int main() {
   cout<<"Nested classes in C++"<< endl;
   A :: B obj;
   obj.getdata(9);
   obj.putdata();
   return 0;
}
```

Output:



```
Nested classes in C++
The number is 9
```