Unit-3

Inheritance

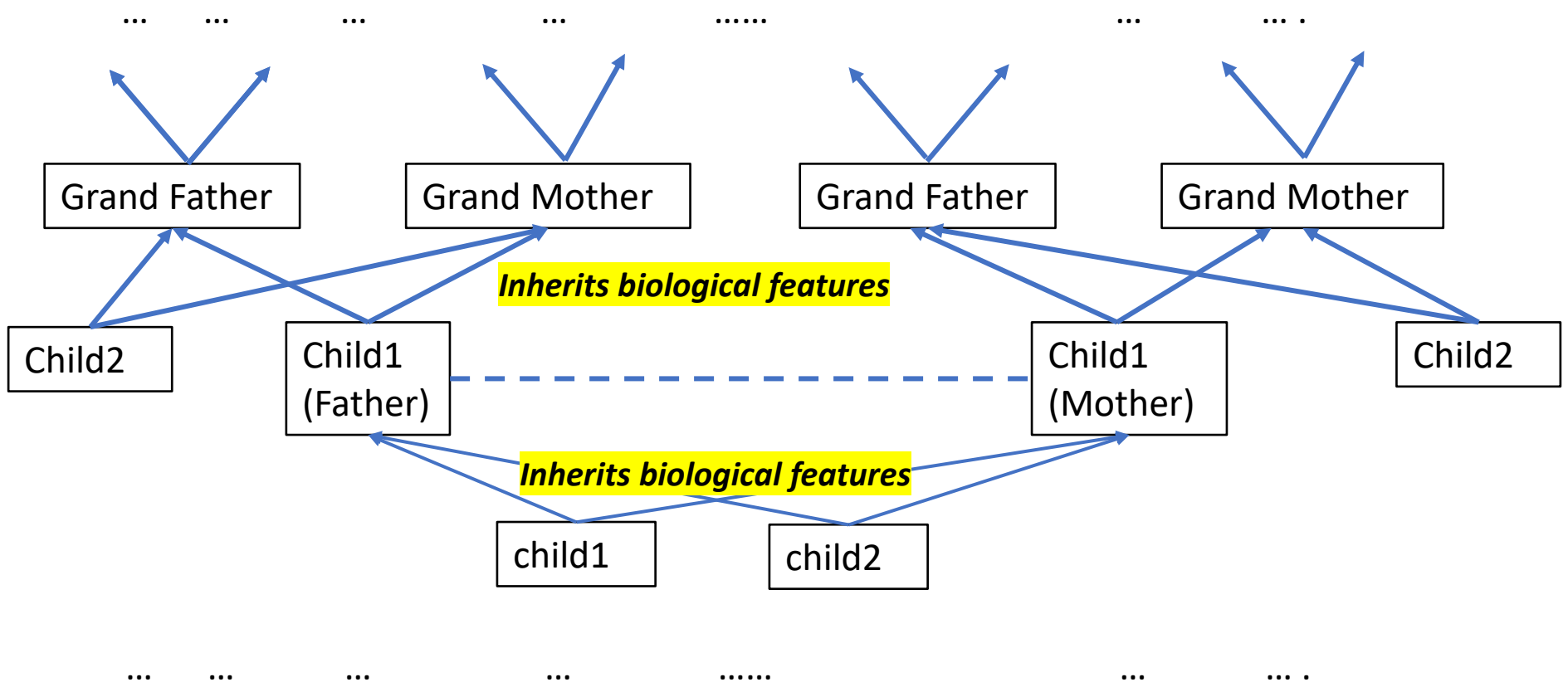
Contents

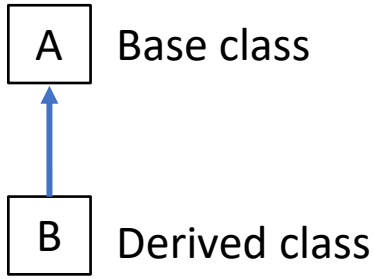
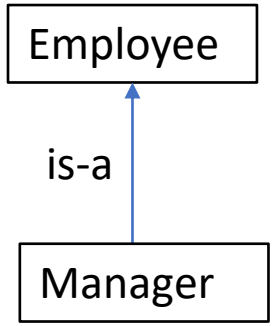
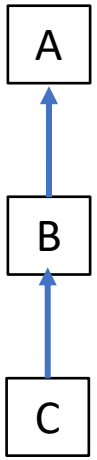
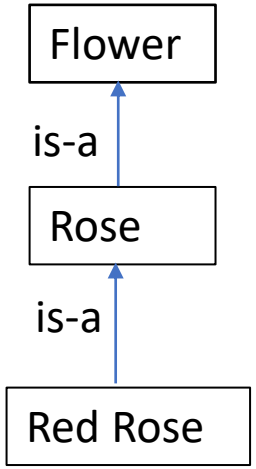
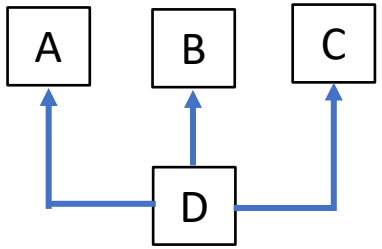
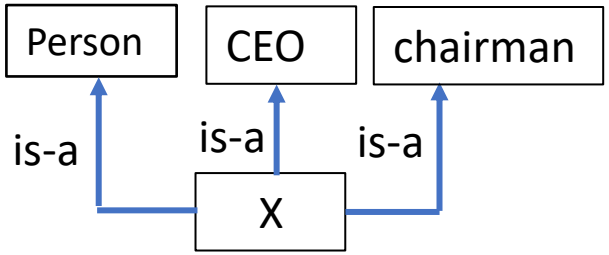
Inheritance: Access specifiers and simple inheritance, protected data with private inheritance, Types of Inheritance: Single, Multilevel, Multiple, Hierarchical, Hybrid and Multipath, Virtual Base Classes.

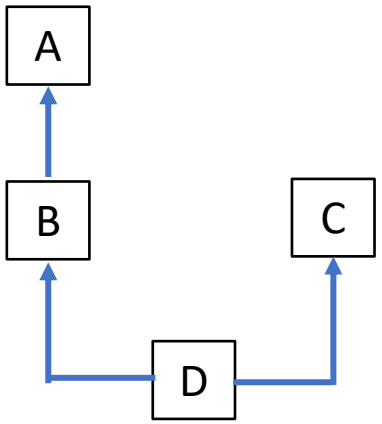
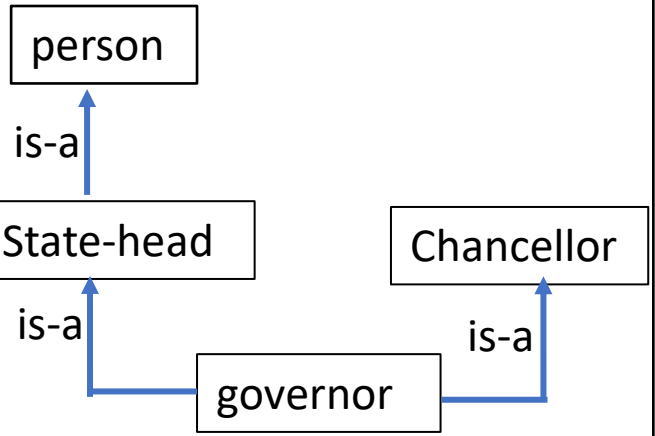
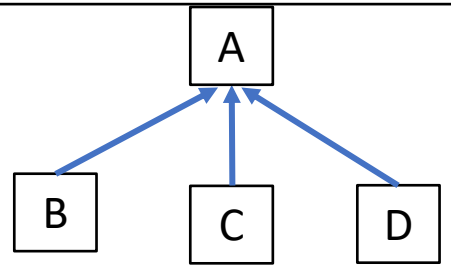
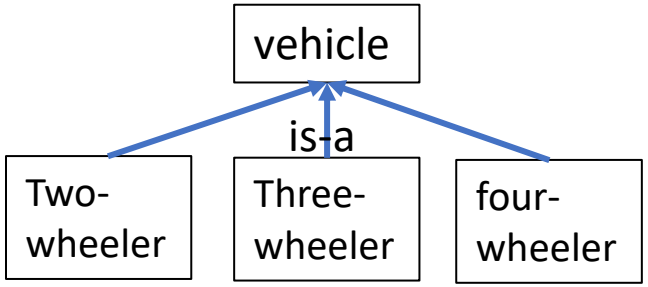
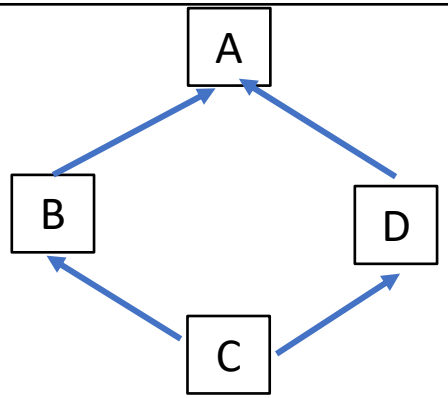
Pointers: void pointer, wild pointer, The this pointer.

Binding, Polymorphism, and Virtual Functions: Binding in C++, Pointer to Base and Derived class, Virtual Function, Rules for Virtual functions, Pure Virtual Functions, Abstract Class.

Inheritance	Reason
Why ?	• To provide generalization and specialization among objects. • Reusability: To extend already existing class and adding their own features (data members) and redefining the functionality(member functions will be override and overload). • To provide a logical relationship among the classes.



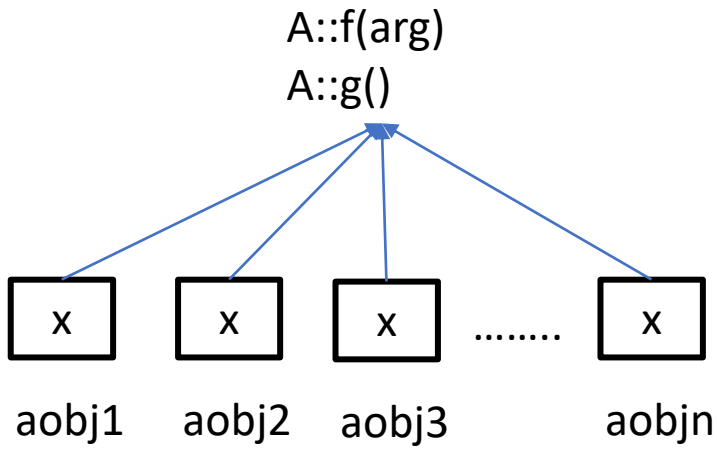
Type of inheritance	Inheritance graph	Syntax	Example
Single inheritance		class A { /*…*/} class B: public A {/*…*/};	
Multi-level inheritance		class A { /*…*/} class B: public A { /*…*/} class C: public B { /*…*/}	
Multiple inheritance		class A { /*…*/} class B { /*…*/} class C { /*…*/} class D: public A,public B,public C {/*…*/}	

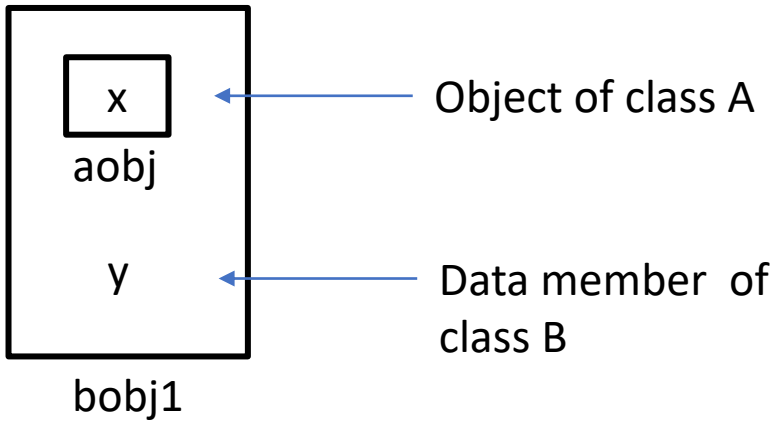
Type of inheritance	Inheritance graph	Syntax	Example
Hybrid inheritance		class A { /*…*/} class C { /*…*/} class B: public A {/*…*/}; class D: public B, public C {/*…*/};	
Hierarchical Inheritance		class A; { /*…*/} class B: public A { /*…*/} class C: public A { /*…*/} class D: public A { /*…*/}	
Multipath Inheritance		class A { /*…*/} class B { /*…*/} class C { /*…*/} class D: public A,public B,public C {/*…*/}	

Example: Explicitly linking the relationship between the classes by including the class A object in class B.

```
class A {
    int x;
public:
    void f(int arg) { x = arg; }
    int g() { return x; }
};
class B {
public:
    A aobj;
    int y;
};
int main() {
    B bobj;
    bobj.aobj.f(20);
    cout << bobj.aobj.g() << endl;
    bobj.y=20;
    cout<<bobj.y<<endl;
 //cout << bobj.g() << endl;
    return 0;
}
```

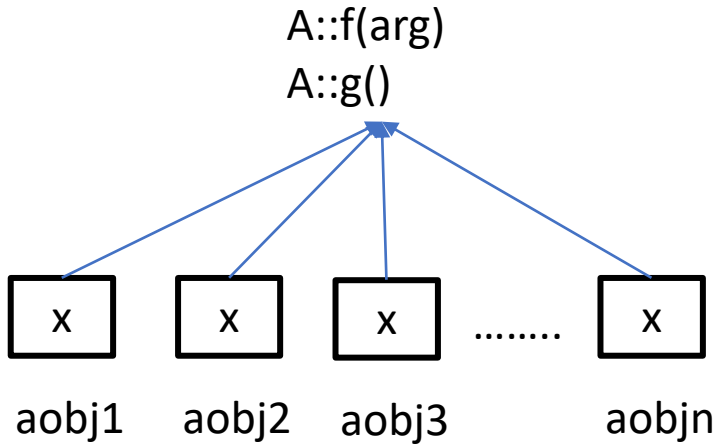
Not accessible because g is not a member of B

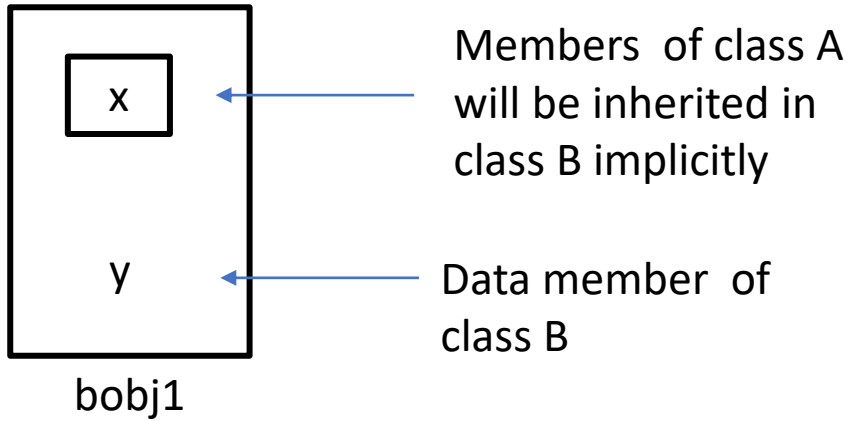




Advantage of Inheritance

```
class A {
    int x;
public:
    void f(int arg) { x = arg; }
    int g() { return x; }
};
class B: public A {
public:
    int y;
};
int main() {
    B bobj;
    bobj.f(20);
    cout << bobj.g() << endl;
    bobj.y=20;
    cout<<bobj.y<<endl;
    return 0;
}
```





- Here , class B doesn't include any member of class A
- Due to inheritance, class B contains the members of class A implicitly. i.e., f and g are accessed using class B object.

In inheritance , do all members of base class will be available in derived class ? No

Without Inheritance

```
class A {
    int x;
  public:
    void f(int arg) { x = arg; }
    int g() { return x; }
};
class B {
   public:
    A aobj;
    int y;
};
int main() {
    B bobj;
    bobj.aobj.f(20);
    cout << bobj.aobj.g() << endl;
    bobj.y=20;
    cout<<bobj.y<<endl;
 // cout << bobj.g() << endl;
    return 0;
}
```

With Inheritance

```
class A {
    int x;
public:
    void f(int arg) { x = arg; }
    int g() { return x; }
};
class B: public A {
public:
    int y;
};
int main() {
    B bobj;
    bobj.f(20);
    cout << bobj.g() << endl;
    bobj.y=20;
    cout<<bobj.y<<endl;
    return 0;
}
```

Access Specifiers with inheritance

```
class A{
    private : int a=10;
    protected: int b=20;
    public : int c=30;
};
class B:public A{
    public:
    void display(){
     //   cout<<a<<endl;
        cout<<b<<endl;
        cout<<c<<endl;
    }
};
int main(){
    B obj;
    obj.display();
    return 0;
}
```

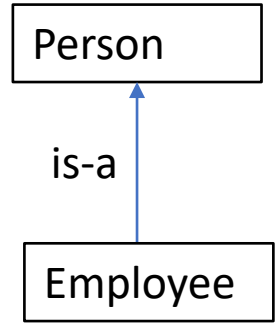
	Access Specifier		
	private	public	protected
Base	yes	yes	yes
Derived	No	yes	yes

Access Specifiers with inheritance + general

```
class A{
    private : int a=10;
    protected: int b=20;
    public : int c=30;
};
class B:public A{
};
int main(){
    B obj;
//   cout<<obj.a<<endl;
//   cout<<obj.b<<endl;
   cout<<obj.c<<endl;
    return 0;
}
```

		Access Specifier		
		private	public	protected
Inheritance	Base	yes	yes	yes
	Derived	No	yes	Yes
Without Inheritance	Accessing using object	No	yes	No

Single Inheritance



Base class

```
class person {
   protected:
     string name;
     string gender;
};
```

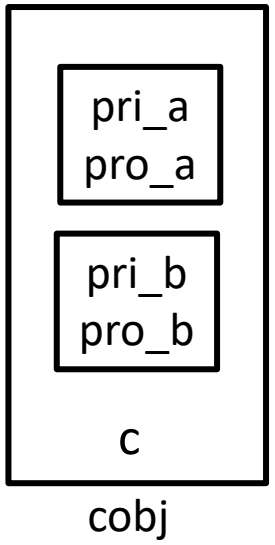
Derived class

```
class student: public person{
   int rollno;
   string branch;
   public:
     student(string n,string g,int rno,string b){
        name=n;gender=g,rollno=rno,branch=b;
     }
     void display(){
        cout<<"Student details....."<<endl;
        cout<<"name="<<name<<endl;
        cout<<"gender="<<gender<<endl;
        cout<<"rollno="<<rollno<<endl;
        cout<<"branch="<<branch<<endl;
     }
};
```

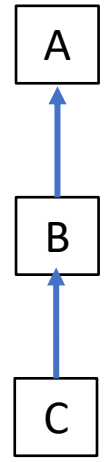
```
int main(){
   student s1("raju","male",501,"CSE");
   student s2("kamala","female",1201,"IT");
   student s3("wilson","male",401,"ECE");
   s1.display();
   s2.display();
   s3.display();
   return 0;
}
```

Multi-level Inheritance

```
using namespace std;
class A{
   private:int pri_a;
   protected: int pro_a;
   public:
      void set_a(int x){pri_a=x;}
      int get_a(){ return pri_a;}
};
class B:public A{
   private:int pri_b;
   protected: int pro_b;
   public:
      void set_b(int x){pri_b=x;}
      int get_b(){ return pri_b;}
};
```



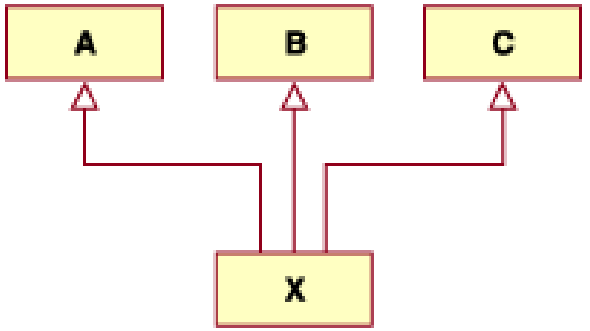

```
class C:public B{
   int c;
   public:
    void setData(){
        set_a(10);  pro_a=20;
        set_b(30);  pro_b=40;
        c=50;
    }
    void display(){
        cout<<"class A data accessed in C"<<endl;
        cout<<"private data : "<<get_a()<<endl;
        cout<<"protected data : "<<pro_a<<endl;
        cout<<"class B data  accessed in C"<<endl;
        cout<<"private data : "<<get_b()<<endl;
        cout<<"protected data : "<<pro_b<<endl;
        cout<<"class C data"<<endl;
        cout<<"private data : "<<c<<endl;
     }
};
 int main(){
    C cobj;
    cobj.setData();
    cobj.display();
    return 0; }
```



- protected and public members are inherited in derived class.
- private data is not inherited in derived class so, use public members functions to access the data.

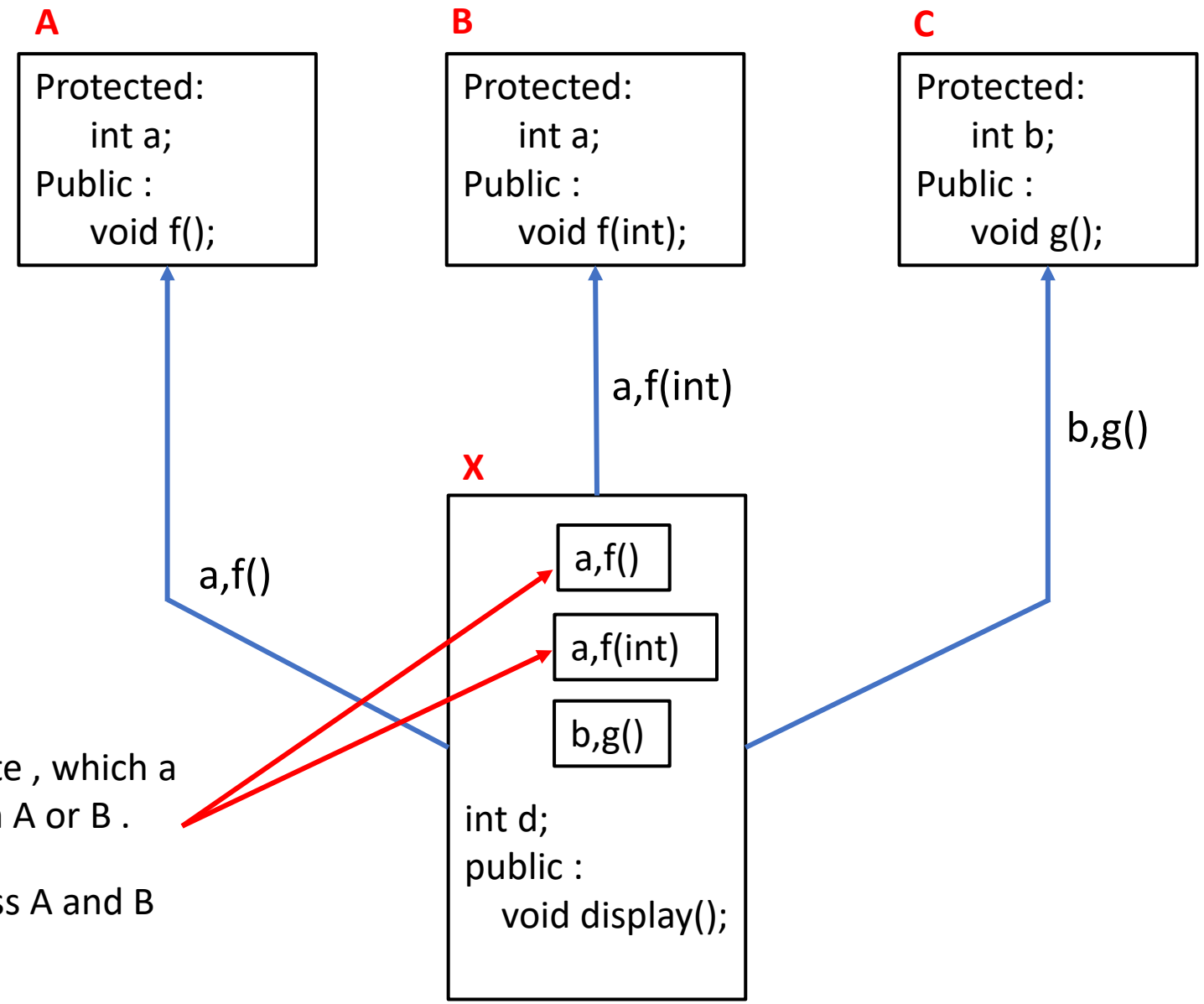
Multiple Inheritance

```
class A{
   protected: int a;
   public:
   A(){ a=10; cout<<"A constructor was called"<<endl;}
   void f(){  cout<<"a value="<<a<<endl; }
};

class B{
   protected: int a;
   public:
   B(){ a=20; cout<<"B constructor was called"<<endl;}
   void f(int y){  a=a+y;
            cout<<"a value="<<a<<endl;
   }
};

class C{
   protected: int b;
   public:
      C(){ b=30; cout<<"C constructor was called"<<endl;}
      void g(){
         cout<<"b value="<<b<<endl;
      }
};
```




```
class X:public A, public B, public C{
   int d;
   public:
   void display(){
//   f();       Error
//   f(10);    Error
      A::f();
      B::f(10);
      g();
   }
};
int main(){
   X obj;
   obj.display();
}
```

Multiple Inheritance

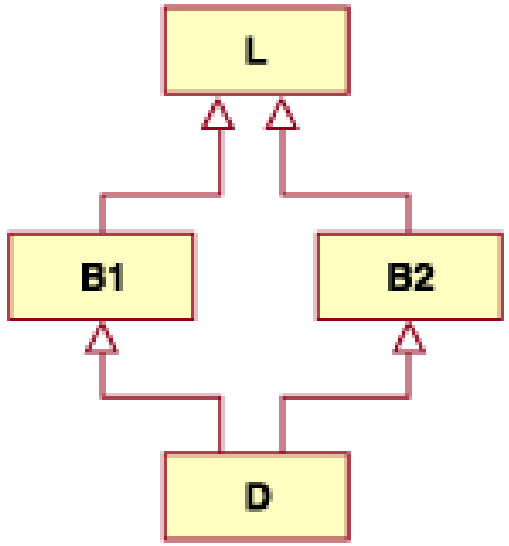


- class D is in ambiguous state , which a and f() will access i.e., from A or B .
- So, use :: operator to access A and B
  A::a and A::f()
  B::a and B::f(int)

Multi-path Inheritance

An object of class D has two distinct sub objects of class L, one through class B1 and another through class B2. You can use the keyword virtual in front of the base class specifiers in the *base lists* of classes B1 and B2 to indicate that only one sub object of type L, shared by class B1 and class B2, exists.

```
class L { /* ... */ }; // indirect base class
class B1 : virtual public L { /* ... */ };
class B2 : virtual public L { /* ... */ };
class D : public B1, public B2 { /* ... */ }; // valid
```



Using the keyword virtual ensures that an object of class D inherits only one subobject of class L.

Multi-path Inheritance

```
class L {
   protected: int a;
   public:
      L(){
         a=10;
         cout<<"class L constructor is called"<<endl;
      }
};
```

class B1: virtual public L

```
class B1: public L{
   protected: int b;
   public:
      B1(){
         b=20;
         cout<<"class B1 constructor is called"<<endl;
      }
};
```

class B2: virtual public L

```
class B2: public L{
   protected: int c;
   public:
      B2(){
         c=30;
         cout<<"class B2 constructor is called"<<endl;
      }
};
```

```
class D:public B1,public B2{
   protected: int d;
   public:
      void display(){
         cout<<"a value "<<a<<endl;
         cout<<"b value "<<b<<endl;
         cout<<"c value "<<c<<endl;
      }
};
```

- **error: reference to 'a' is ambiguous**
- **To resolve the ambiguity using virtual keyword preceding base class specifier**

Access control of base class members

When you declare a derived class, an access specifier can precede each base class in the base list of the derived class

Class Derived : **Access-Specifier** Baseclass { /* ..*/}

You can derive classes using any of the three access specifiers:

- In a **public** base class, **public and protected members** of the base class remain public and protected members of the derived class.
  Class Derived: **public** BaseClass{ /* ..*/}
- In a **protected** base class, public and protected members of the base class are protected members of the derived class.
  Class Derived: **protected** BaseClass { /* ..*/}
- In a **private** base class, public and protected members of the base class become private members of the derived class.
  Class Derived: **private** BaseClass { /* ..*/}