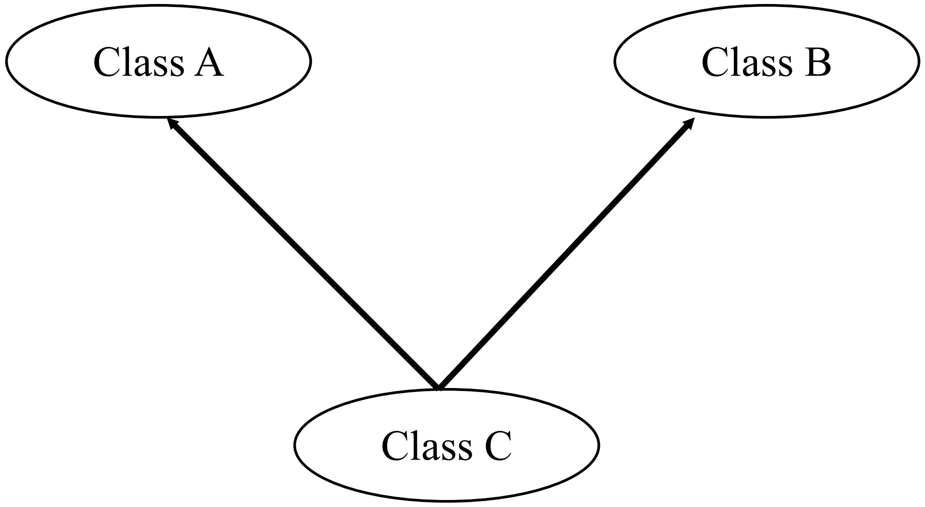
What is Multiple Inheritance?

Multiple Inheritance

- A class can be derived from more than one parent



```
class C : pubic A, public B
{
      ….
}
```

Order of Construction

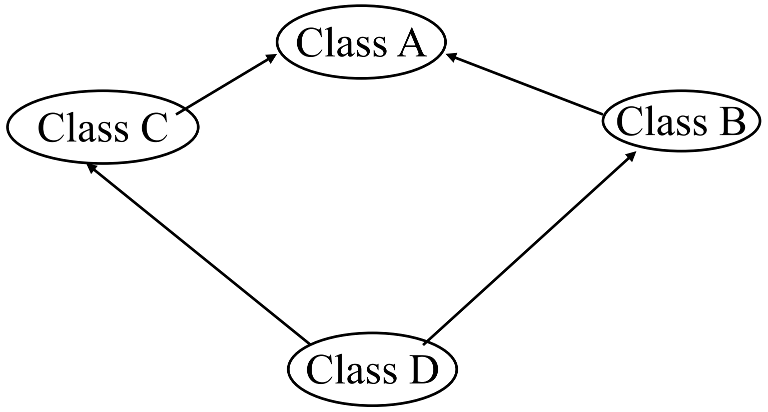
- The constructor member initialization list controls only the values that are used to initialize the base classes, not the order in which the base classes are constructed:
- The base class constructors are called in the order in which they appear in the class derivation list:

```
class C : public B, public A {
                ....
}
```

- In this example the constructor of class B is called first, and then the constructor of A.

Multiple Inheritance

- Although a base class may legally appear only once in a derivation list, a base class data member can appear multiple times within a derivation hierarchy that form a diamond shape as show, below. This can cause a member ambiguity.
- Why? (a class discussion)



Possible Ambiguity Error

```
class A
{
   private:
     int a;
};

class B: public A
{
 private:
     int b;
};

class C: public A
{
 private:
     int c;
}
```

```
class D: public  B, public C
{
    …
};
```

```
int main() {
   D myd;
   // trying to use a in class D,
   // gives ambiguity error. Why?
   return 0;
}
```

Possible Ambiguity Error (Continued)

- In the previous example class D will contain two copies of each member of class A. This can be a source of ambiguity.
  - To avoid this problem, class A must be declared as virtual base class for classes B and C.
- A virtual base class must be initialized by its most derived class. Means class D constructor is responsible to invoke the constructors of classes A, B and C.

What is Virtual Base Class

Declaring a base class, virtual, can resolve the issue of ambiguity.

```
class A
{
   ...
};

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

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

```
class D: public  B, public C
{
    ...
};
```

What is a private or protected base class?

Public, protected and private base classes

- Public base class: The inherited members of a public base class maintain their access level within the derived class.
- Protected base class: The access level of public members of a protected base class will change to protected within the derived class.
- Private base class: The access level of public and protected members of a private base class will change to private within the derived class.

- Further details will be discussed during the lectures

Inheriting operators and functions

- A derived class inherits all the member functions of each of its base classes. However, a child class still needs to have its own constructors (including copy constructor), and possibly destructor, and assignment operators to manage tasks such as memory management for its own data members (not those inherited from base class).

Derived Class Copy Constructor

- If a derived class explicitly defines its own copy constructor, the definition completely overrides the default definition, and is responsible for copying its base class component.
- Assuming we have a class called Base:

```
class Base { …};
```

- And, a class called Derived, as followed. How its copy-constructor should be?

```
class Derived: public Base {
public:
    // class Derived MUST have its own copy constructor:
};
```

- **Please try to write the definition of the copy ctor for class Derived. Solution will be discussed during lecture**

Derived Class Assignment Operator

- If a derived class explicitly defines its own assignment operator, the definition completely overrides the default definition, and is responsible for copying its base class component.
- Assuming we have previous classes Base, and Derived.

```
class Base { …};


class Derived: public Base {
 public:
  // class Derived MUST have its own assignment operator
 Derived& operator = (const Derived& rhs);
 };
```

Please try to write the definition of the assignment operator for class Derived.

Solution will be discussed during lecture

Derived Class Destructor

- Derived class destructor is never responsible for destroying the members of the base object.
- The compiler will always implicitly invoke the destructor of the base part of the derived object:

```
class Base { …};

class Derived: public Base {
public
      ~Derived () { /* ~Base is automatically called */ }
};
```

What is a private or protected base class?

Public, protected and private base classes

- Public base class: The inherited members of a public base class maintain their access level within the derived class.
- Protected base class: The access level of public members of a protected base class will change to protected within the derived class.
- Private base class: The access level of public and protected members of a private base class will change to private within the derived class.

Example

```
class A {
  public: int x;
  protected: int y;
  private: int z;
  public: void fun();  // x, y, z are accessible
};

class B: protected A {
  public: int k;
  protected: int l;
  private: int m;
  public: void fun(); // x, y, k, l, m are accessible
};

class C: private B {
  public: int p;
  protected: int r;
  private: int s;
  public: void fun(); //x, y, k, l, p, r, s are accessible
};
```

```
class D: public C {
  public: int u;
  protected: int v;
  private: int w;
  public: void fun();  // p, r, u, v, w are accessible
  // The followings are all inaccessible:
  //x, y, z,  k, l, m, s

};
```