**Problem with multiple inheritance:**

Taking a look at the graphics below helps in explaining the diamond problem.

|  |
| --- |
| http://www.programmerinterview.com/images/Diamond_inheritance.png |

|  |
| --- |
|  |

Suppose we have 2 classes B and C that derive from the **same** class – in our example above it would be class A. We also have class D that derives from **both** B and C by using multiple inheritance. You can see in the figure above that the classes essentially form the shape of a diamond – which is why this problem is called the diamond problem. Now, let’s take the graphic above and make it more concrete by translating it into actual code:

|  |
| --- |
| /\*  The Animal class below corresponds to class  A in our graphic above  \*/    class Animal { /\* ... \*/ }; // base class  {  int weight;  public:  int getWeight() { return weight;};  };  class Tiger : public Animal { /\* ... \*/ };  class Lion : public Animal { /\* ... \*/ }    class Liger : public Tiger, public Lion { /\* ... \*/ }; |

In the code above, we’ve given a more concrete example of the diamond problem. The Animal class corresponds to the topmost class in the hierarchy (A in our graphic above), Tiger and Lion respectively correspond to B and C in the graphic, and the Liger class corresponds to D.

Now, the question is what is the problem with having an inheritance hierarcy like this. Take a look at the code below so that we can best answer that question:

|  |
| --- |
| int main( )  {  Liger lg ;  /\*COMPILE ERROR, the code below will not get past  any C++ compiler \*/  int weight = lg.getWeight();  } |

|  |
| --- |
|  |

In our inheritance hierarchy, we can see that both the Tiger and Lion classes derive from the Animal base class. And here is the problem: **because Liger derives from both the Tiger and Lion classes – which each have their own copy of the data members and methods of the Animal class- the Liger object "lg" will contain two subobjects of the Animal base class.**

So, you ask, what’s the problem with a Liger object having 2 **sub**-objects of the Animal class? Take another look at the code above – the call "lg.getWeight()" will result in a compiler error. This is because the compiler **does not know whether the call to getWeight refers to the copy of getWeight that the Liger object lg inherited through the Lion class or the copy that lg inherited through the Tiger class**. So, the call to getWeight in the code above is ambiguous and will not get past the compiler.

## Solution to the Diamond Problem

We’ve given an explanation of the diamond problem, but now we want to give you a solution to the diamond problem. If the inheritance from the Animal class to both the Lion class and the Tiger class is marked as virtual, then C++ will ensure that only one subobject of the Animal class will be created for every Liger object. This is what the code for that would look like:

|  |
| --- |
| class Tiger : virtual public Animal { /\* ... \*/ };  class Lion : virtual public Animal { /\* ... \*/ } |

You can see that the only change we’ve made is to add the "virtual" keyword to the Tiger and Lion class declarations. Now the Liger class object will have only one Animal subobject, and the code below will compile just fine:

|  |
| --- |
| int main( )  {  Liger lg ;  /\*THIS CODE WILL NOW COMPILE OK NOW THAT WE'VE  USED THE VIRTUAL KEYWORD IN THE TIGER AND LION  CLASS DECLARATIONS \*/  int weight = lg.getWeight();  } |

## What is a virtual base class?

- An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.  
- C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

## What is Virtual base class? Explain its uses.

- When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class’ name with the word virtual.  
- Consider the following example :

class A   
{   
   public:   
       int i;   
};  
  
class B : virtual public A   
{   
   public:   
       int j;   
};  
  
class C: virtual public A   
{   
   public:   
       int k;   
};  
  
class D: public B, public C   
{   
   public:   
       int sum;   
};  
  
int main()   
{   
   D ob;   
   ob.i = 10; //unambiguous since only one copy of i is inherited.   
   ob.j = 20;   
   ob.k = 30;   
   ob.sum = ob.i + ob.j + ob.k;   
   cout << “Value of i is : ”<< ob.i<<”\n”;   
   cout << “Value of j is : ”<< ob.j<<”\n”; cout << “Value of k is :”<< ob.k<<”\n”;   
   cout << “Sum is : ”<< ob.sum <<”\n”;   
  
   return 0;   
}

Document Created by

**Md. Abdul Mazed**

CSTE 9th Batch

Noakhali Science & Technology University