OOPS

Multiple Inheritance

There are two ways that a derived class can inherit more than one base class.

First, a derived class can be used as a base class for another derived class, creating a multilevel class hierarchy. In this case, the original base class is said to be an indirect base class of the second derived class.

Second, a derived class can directly inherit more than one base class. In this situation, two or more base class are combined to help create the derived class.

There several issues that arise when multiple base classes are involved. Those will be discussed in this section. If a class B1 is inherited by a class D1, and D1 is inherited by a class D2, the constructor functions of all the three classes are called in order of derivation. Also the destructor functions are called in reverse order.

When a derived class inherits multiple base classes, it uses the expanded declaration:

```
class derived-class-name : access base1, access base2, ... , access baseN
{
     // ... body of class
};
```

Here **base1** through **baseN** are the base class names and access the access specifier, which can be different for each base class.

When multiple base classes are inherited, constructors are executed in the order, left to right that the base classes are specified. Destructors are executed in reverse order.

When a class inherits multiple base classes that have constructors that requires arguments, the derived class pass the necessary arguments to them by using this expanded form class constructor function:

```
derived-constructor(arg-list) : base1(arg-list) ,..., baseN(arg-list)
{
     // body of derived class constructor
}
```

Here base1 through baseN are the names of the classes.

Virtual base classes

A potential problem exists when multiple base classes are directly inherited by a derived class.

To understand what this problem is, consider the following class hierarchy:

Base class Base is inherited by both Derived1 and Derived2. Derived3 directly inherits both Derived1 and Derived2.

However, this implies that Base is actually **inherited twice** by Derived3. First it is inherited through Derived1, and then again through Derived2. This causes ambiguity when a member of Base is used by Derived3. Since two copies of Base are included in Derived3, is a reference to a member of Base referring to the Base inherited indirectly through Derived1 or to the Base inherited indirectly through Derived2?

To resolve this ambiguity, C++ includes a mechanism by which only one copy of Base will be included in Derived3. This feature is called a ***virtual base class***

In situations like this, in which a derived class indirectly inherits the same base class more than once, it is possible to prevent multiple copies of the base from being present in the derived class by having that base class inherited as virtual by any derived classes. Doing this prevents two or more copies of the base from being present in any subsequent derived class that inherits the base class indirectly. The virtual keyword precedes the base class access specifier when it is inherited by a derived class.

Diamond Problem - Programmer and Software Interview Questions and Answers

```
// This program uses a virtual base class.
#include <iostream>

using namespace std;
class Base {
  public:
   int i;
};
```

```
// Inherit Base as virtual
class Derived1 : virtual public Base {
  public:
   int j;
};

// Inherit Base as virtual here, too
class Derived2 : virtual public Base {
  public:
   int k;
};

// Here Derived3 inherits both Derived1 and Derived2.
// However, only one copy of base is inherited.
class Derived3 : public Derived1, public Derived2 {
  public:
   int product() { return i * j * k; }
};

int main() {
   Derived3 ob;
   ob.i = 10;  // unambiguous because virtual Base
   ob.j = 3;
   ob.k = 5;
   cout << "Product is: " << ob.product() << "\n";
   return 0;
}
```

Output:

```
Product is: 150
```

If Derived1 and Derived2 had not inherited Base as virtual, the statement ***ob.i=10*** would have been ambiguous and a compile-time error would have resulted.