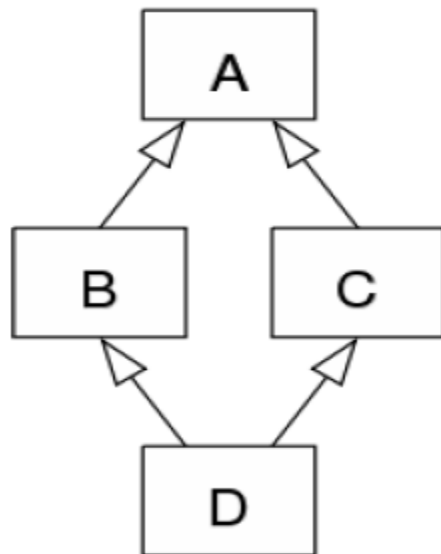


Course Code: CL-217	Course : Object Oriented Programming Lab
Instructor(s) :	Nida Munawar

## Contents:

1. Diamond Problem in Hybrid Inheritance
2. Constructors

In case of hybrid inheritance, a Diamond problem may arise. The “dreaded diamond” refers to a class structure in which a particular class appears more than once in a class’s inheritance hierarchy.



### Virtual base classes

To share a base class, simply insert the “virtual” keyword in the inheritance list of the derived class. This creates what is called a **virtual base class**, which means there is only one base object. The base object is shared between all objects in the inheritance tree and it is only constructed once.

**For Example:**

```

c++ > diamond.cpp > main()
1  #include <iostream>
2  using namespace std;
3
4  class A
5  {
6  public:
7  int a;
8
9  };
10
11 class B : public A{
12 public:
13 int b;
14 };
15
16 class C : public A{
17 | public:
18 int c;
19 };
20
21 class D : public B , public C{
22 | public:
23 int d;
24 };
25 int main(){
26 D obj;
27 obj.a=10;
28
29 }

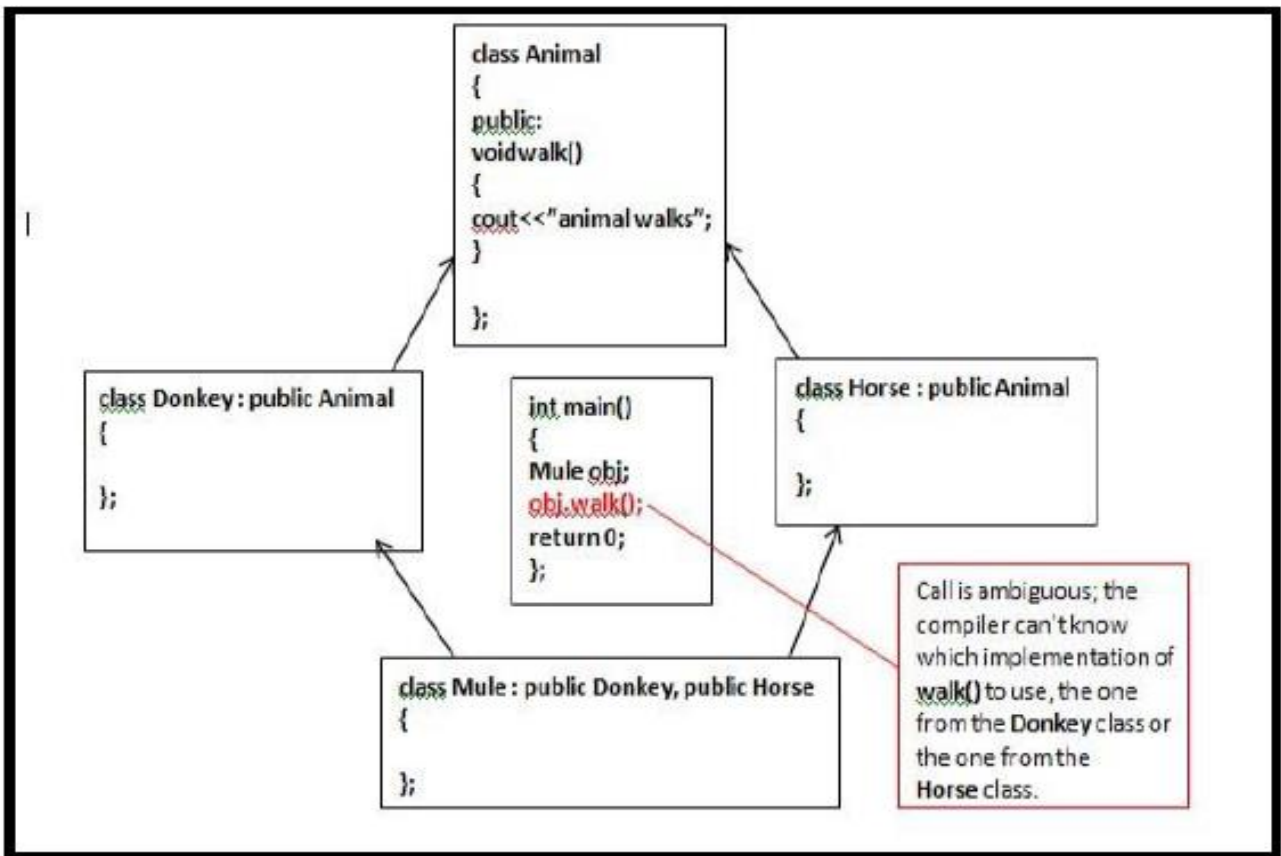
```

How to solve this Problem??? Virtual Base Class Inheritance

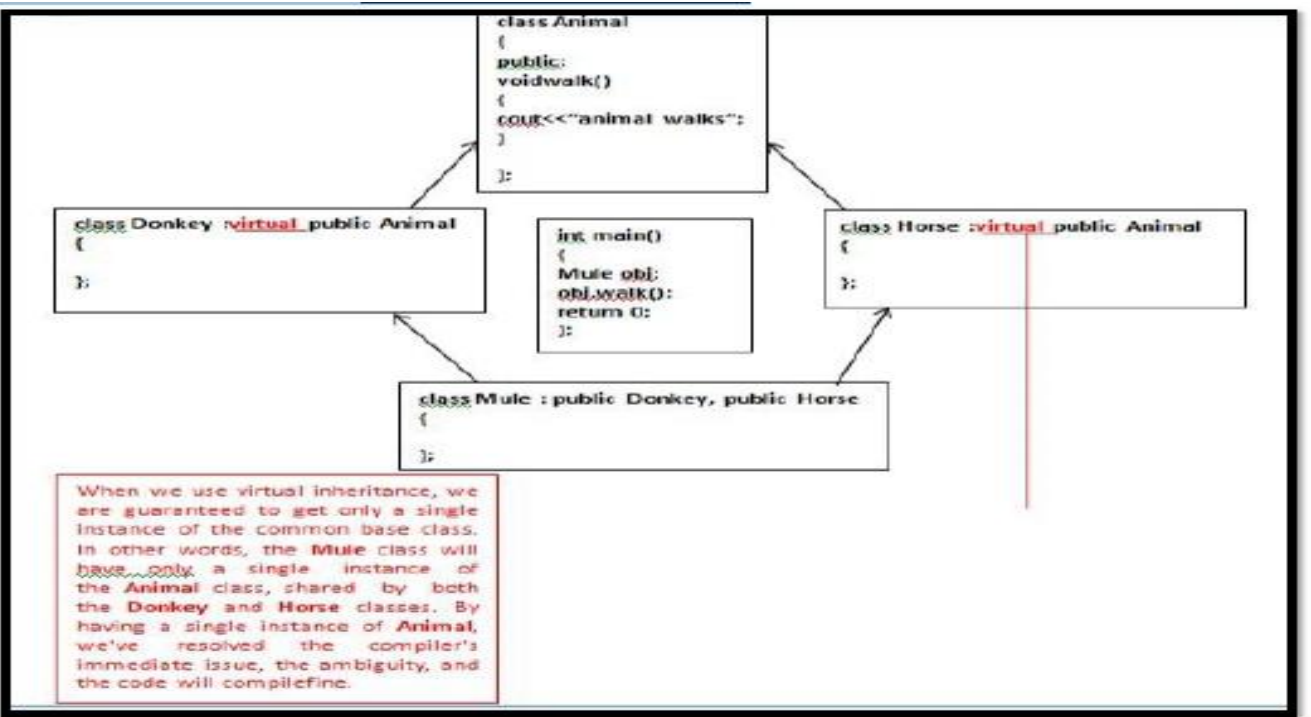
```

c++ > diamond.cpp > D
1  #include <iostream>
2  using namespace std;
3
4  class A
5  {
6  public:
7  int a;
8
9  };
10
11 class B : virtual public A{
12 public:
13 int b;
14 };
15
16 class C : public virtual A{
17 | public:
18 int c;
19 };
20
21 class D : public B , public C{
22 | public:
23 int d;
24 };
25 int main(){
26 D obj;
27 obj.a=10;
28
29 }

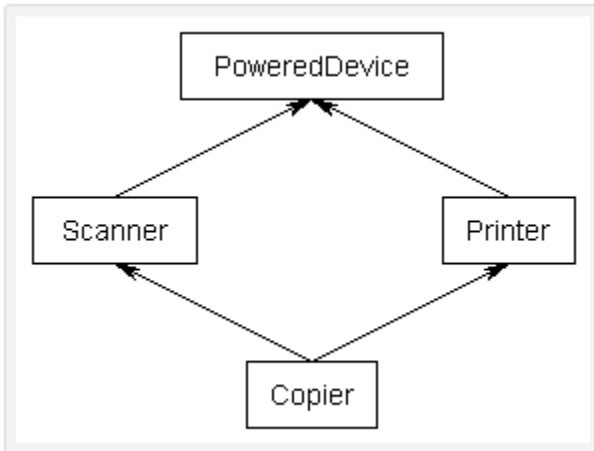
```



How to solve this Problem??? Virtual Base Class Inheritance



Here is another example (with some constructors) illustrating the diamond problem:



```
#include <iostream>
using namespace std;
class PoweredDevice
{
public:
    PoweredDevice(int power)
    {
        cout << "PoweredDevice: " << power << '\n';
    }
};

class Scanner: public PoweredDevice
{
public:
    Scanner(int scanner, int power)
        : PoweredDevice(power)
    {
        cout << "Scanner: " << scanner << '\n';
    }
};

class Printer: public PoweredDevice
{
public:
    Printer(int printer, int power)
```

```

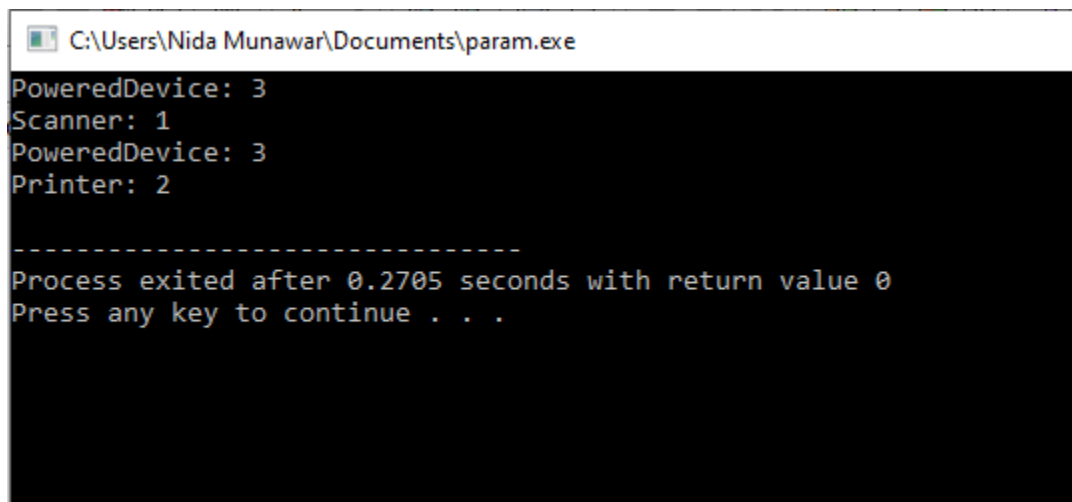
        : PoweredDevice(power)
    {
        cout << "Printer: " << printer << '\n';
    }
};

class Copier: public Scanner, public Printer
{
public:
    Copier(int scanner, int printer, int power)
        : Scanner(scanner, power), Printer(printer, power)
    {
    }
};

int main()
{
    Copier copier(1, 2, 3);

    return 0;}

```



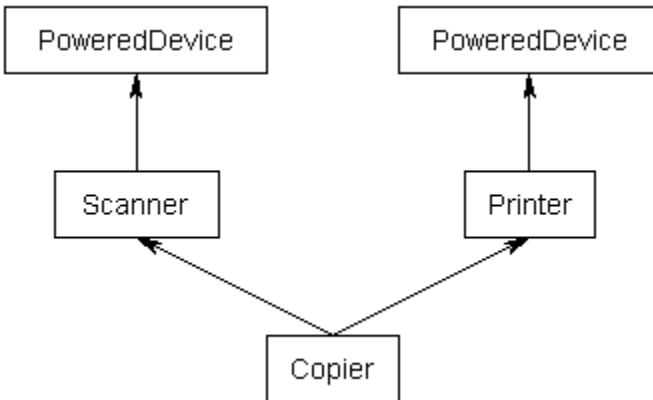
```

C:\Users\Nida Munawar\Documents\param.exe
PoweredDevice: 3
Scanner: 1
PoweredDevice: 3
Printer: 2

-----
Process exited after 0.2705 seconds with return value 0
Press any key to continue . . .

```

If you were to create a Copier class object, by default you would end up with two copies of the PoweredDevice class -- one from Printer, and one from Scanner. This has the following structure:



## By using Virtual Base class

```
#include <iostream>
```

```
class PoweredDevice {
```

```
public:
```

```
    PoweredDevice(int power)
```

```
    { std::cout << "PoweredDevice: " << power << '\n';
```

```
    };
```

```
class Scanner: virtual public PoweredDevice // note: PoweredDevice is now a virtual base class
```

```
{
```

```
public:
```

```
    Scanner(int scanner, int power)
```

```
        : PoweredDevice(power) // this line is required to create Scanner objects, but ignored in this case
```

```
    {    std::cout << "Scanner: " << scanner << '\n';
```

```
    }
```

```
};
```

```
class Printer: virtual public PoweredDevice // note: PoweredDevice is now a virtual base class
```

```
{
```

```
public:
```

```
    Printer(int printer, int power)
```

```
        : PoweredDevice(power) // this line is required to create Printer objects, but ignored in this case
```

```

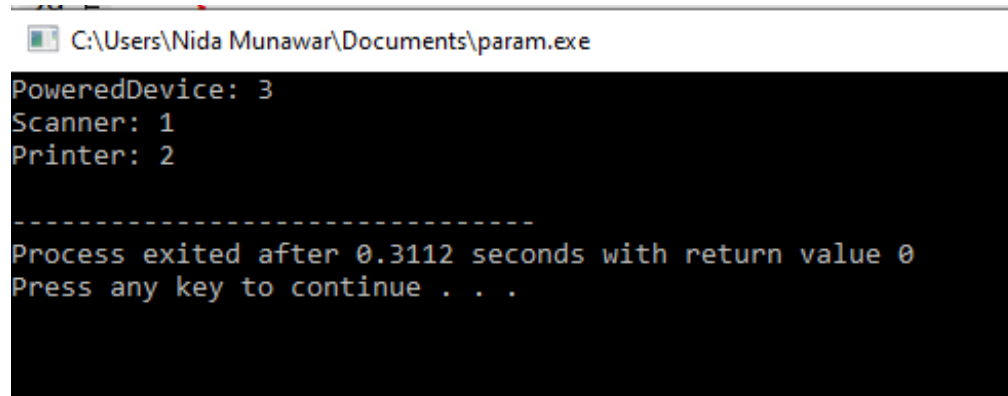
    {    std::cout << "Printer: " << printer << '\n';
    };

class Copier: public Scanner, public Printer
{
public:
    Copier( int scanner ,int printer, int power)
        : PoweredDevice(power), // PoweredDevice is constructed here
        Scanner(scanner, power),Printer(printer, power)
    {
    }
};

int main()
{
    Copier copier(1, 2, 3);
    return 0;}

```

As you can see, PoweredDevice only gets constructed once.



```

C:\Users\Nida Munawar\Documents\param.exe
PoweredDevice: 3
Scanner: 1
Printer: 2
-----
Process exited after 0.3112 seconds with return value 0
Press any key to continue . . .

```

Now, when you create a Copier class object, you will get only one copy of PoweredDevice per Copier that will be shared by both Scanner and Printer.

However, this leads to one more problem: if Scanner and Printer share a PoweredDevice base class, who is responsible for creating it? The answer, as it turns out, is Copier. The

Copier constructor is responsible for creating PoweredDevice. Consequently, this is one time when Copier is allowed to call a non-immediate-parent constructor directly:

### Example#02:

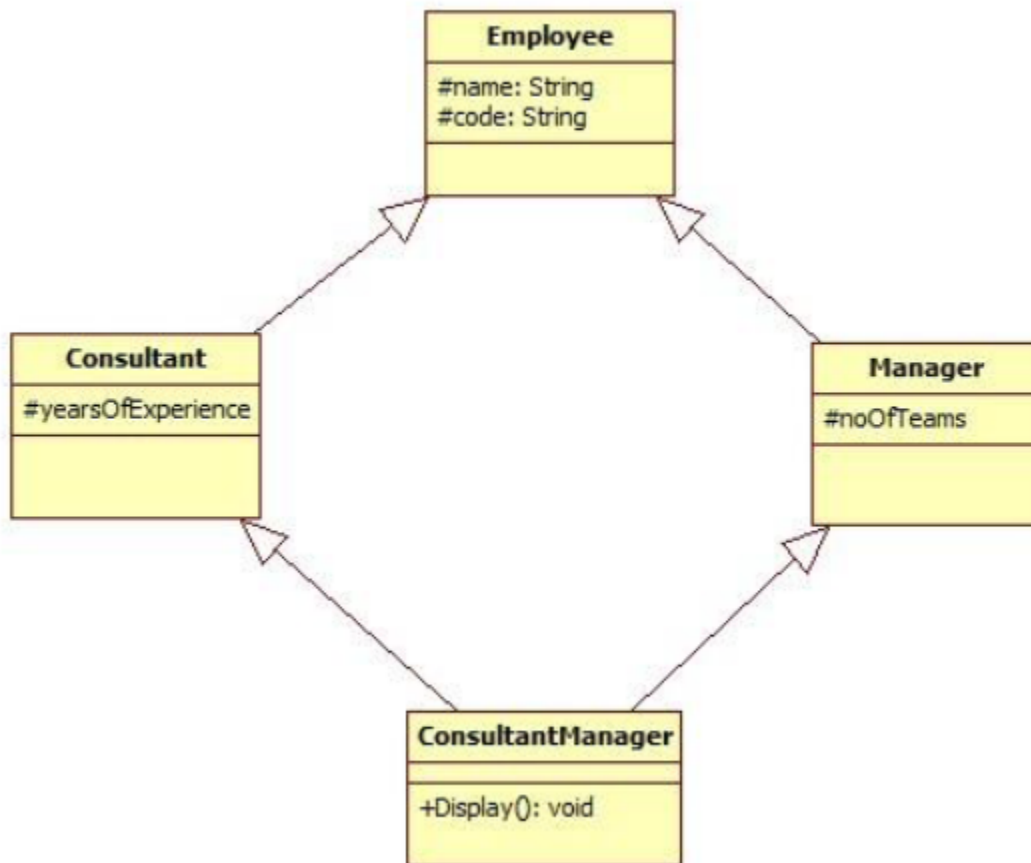
Parametrized Constructor Calling	
<pre>#include&lt;iostream&gt;  using namespace  std; class Person {  public:      Person(int x) { cout &lt;&lt; "Person::Person(int ) called" &lt;&lt; endl; }      Person() { cout &lt;&lt; "Person::Person() called" &lt;&lt; endl; }  };  class Faculty : virtual public Person { public:      Faculty(int x):Person(x) {  cout&lt;&lt;"Faculty::Faculty(int ) called"&lt;&lt; endl;      }};  class Student : virtual public Person {  public:      Student(int x):Person(x) {</pre>	<pre>        cout&lt;&lt;"Student::Student(int ) called"&lt;&lt; endl;      }  };  class TA : public Faculty, public Student { public:      TA(int x):Student(x), Faculty(x), <b>Person(x)</b> {  cout&lt;&lt;"TA::TA(int ) called"&lt;&lt; endl;      }  };  int main() {      TA t(30);  }</pre>



## Exercise

### QUESTION#1

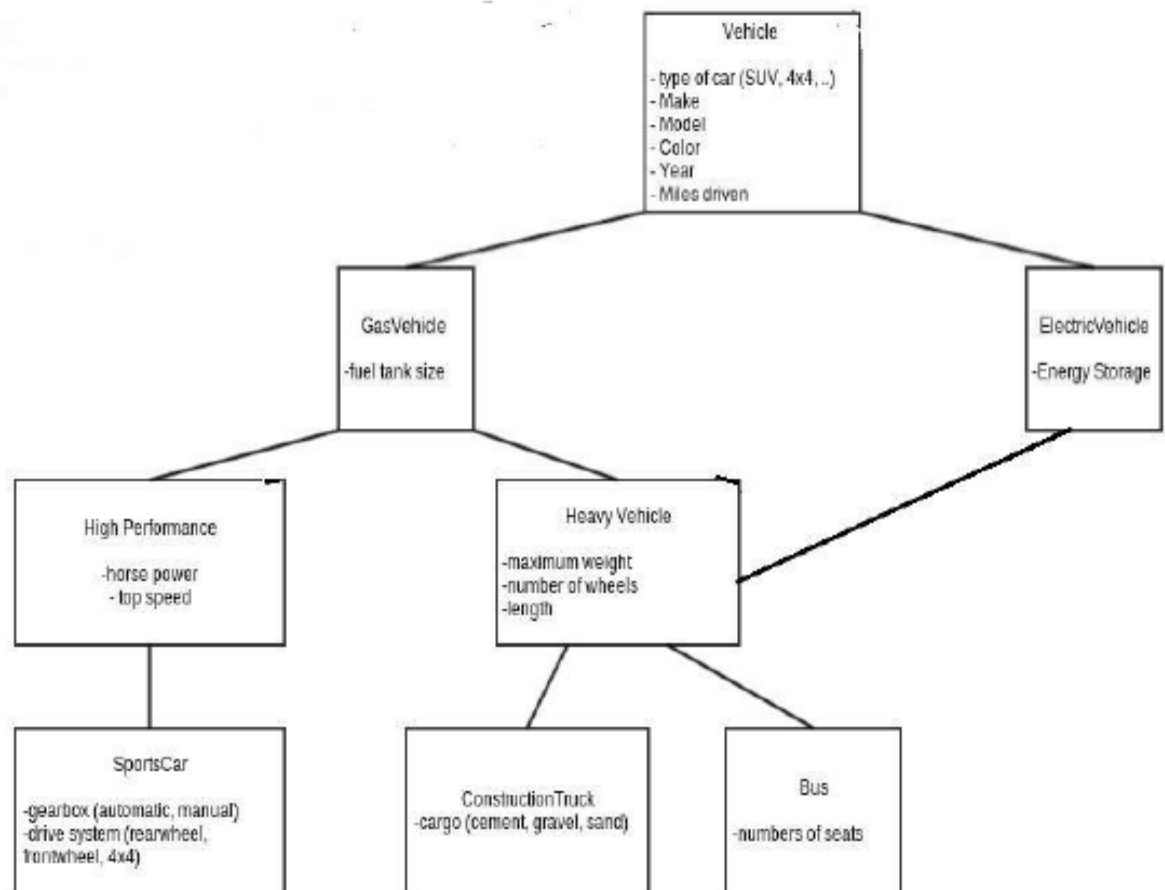
Implement the following scenario in C++:



1. No accessors and mutators are allowed to be used.
2. The `Display()` function in "ConsultantManager" should be capable of displaying the values of all the data members declared in the scenario (name,code,yearsOfExperience,noOfTeams) without being able to alter the values.
3. The "int main()" function should contain only three program statements which are as follows:
  - a) In the first statement, create object of "ConsultantManager" and pass the values for all the data members:  
**`ConsultantManager obj("Ali","S-123",17,5);`**
  - b) In the second statement, call the `Display()` function.
  - c) In the third statement, return 0.

### QUESTION#2

Implement the following scenario in C++:



1. All the values are required to be set through constructor's parameter.
2. Provide necessary accessor functions where required.
3. Create an object of class bus by initializing it through parametrized constructor in the main function and display all data members by calling display function of class bus.