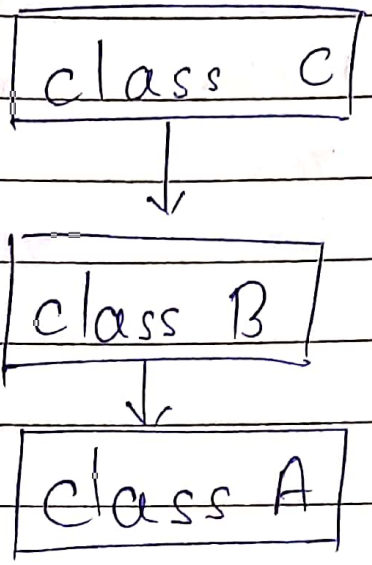
★ Multilevel Inheritance :- In this type of inheritance, a derived class is created from another derived class.



In this diagram, class B is derived from class C and then class A is derived from class B.

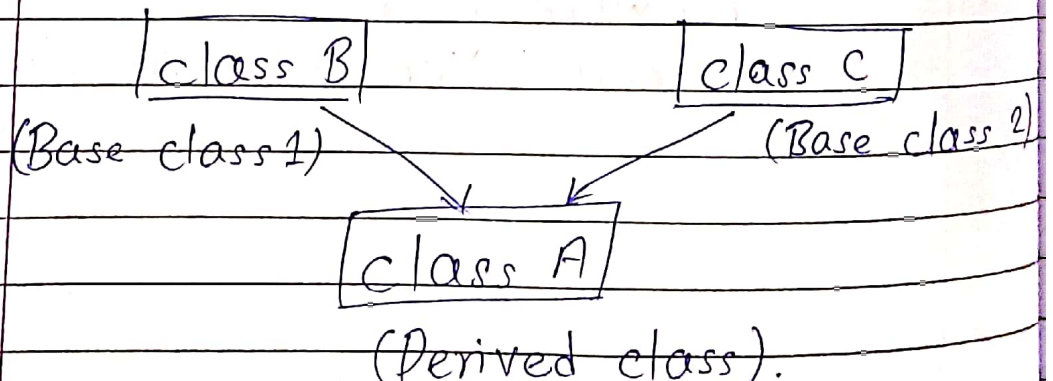
Example :-

```
// C++ program to implement multilevel inheritance.
#include <iostream>
using namespace std;

class vehicle {
    public:
        vehicle() {
            cout << "this is a vehicle" << endl;
        }
};

class FourWheeler : public vehicle {
    public:
        FourWheeler() {
            cout << "Vehicles with 4 wheels"
                 << endl.
        }
};

class Car : public FourWheeler {
    public:
        car() {
            cout << "Car is a fourwheeler" << endl;
        }
```

```
};

int main() {
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

O/P :- this is a vehicle
Vehicle with 4 wheels
Car is a ~~wheeler~~ fourwheeler.

★ Multiple inheritance :- Multiple inheritance is a feature of C++ where a class can inherit from more than one classes i.e one sub-class is inherited from more than one base classes.



Syntax:-

```
class subclass_name : access_mode base_class1,
    access_mode base_class2, .... {
    // body of subclass
};
```

Here, the number of base classes will be separated by a comma (,) and access mode for every base class must be specified.

Example:-

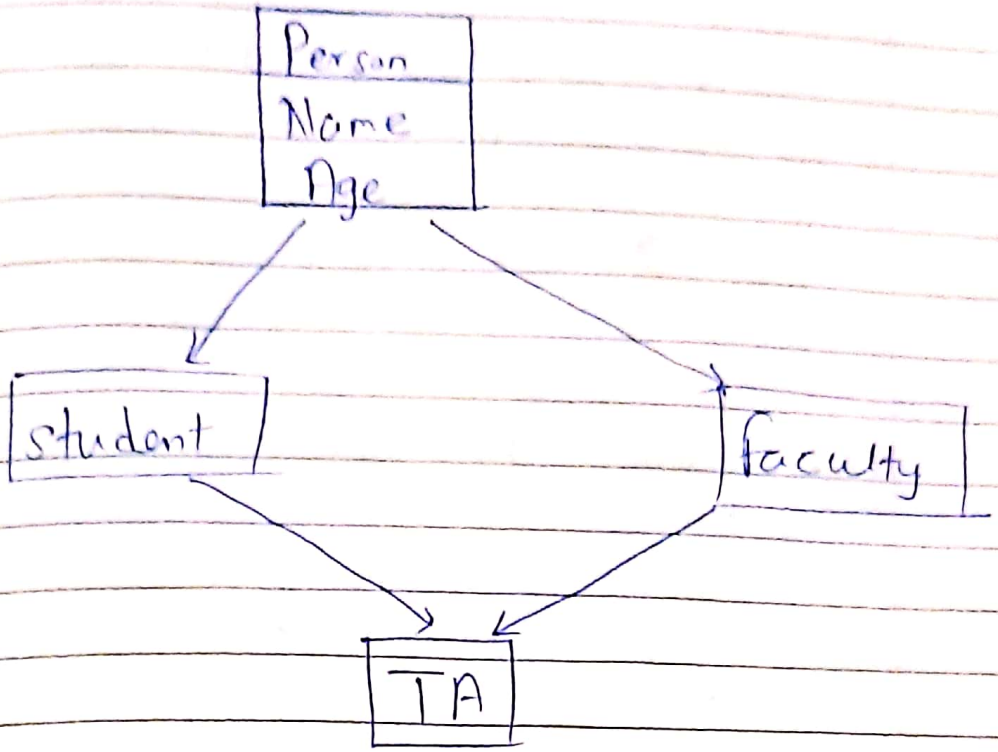
```
// C++ program to explain multiple inheritance
#include <iostream>
using namespace std;

class vehicle {
  public:
    vehicle() {
      cout << "this is a vehicle" << endl;
    }
};

class fourWheeler {
  public:
    fourWheeler() {
      cout << "this is a 4 wheeler vehicle"
           << endl;
```

```
};

class Car : public Vehicle, public FourWh{
};

int main(){
    // creating object of sub class will invoke
    // the constructor of base classes.
    Car obj;
    return 0;
}
```

O/P :- this is a vehicle.
this is a 4 wheeler vehicle.

The constructors of inherited classes are called in the same order in which they are inherited. and the destructors are called in reverse order of constructors.

The diamond problem :-
The diamond problem occurs when two superclasses of a class have a common base. for example, in the following diagram, the TA class gets two copies of all attributes of person class. this causes ambiguities.



Here in TA Name & Age only needed one time but get two times by both student & faculty classes.

Example :-

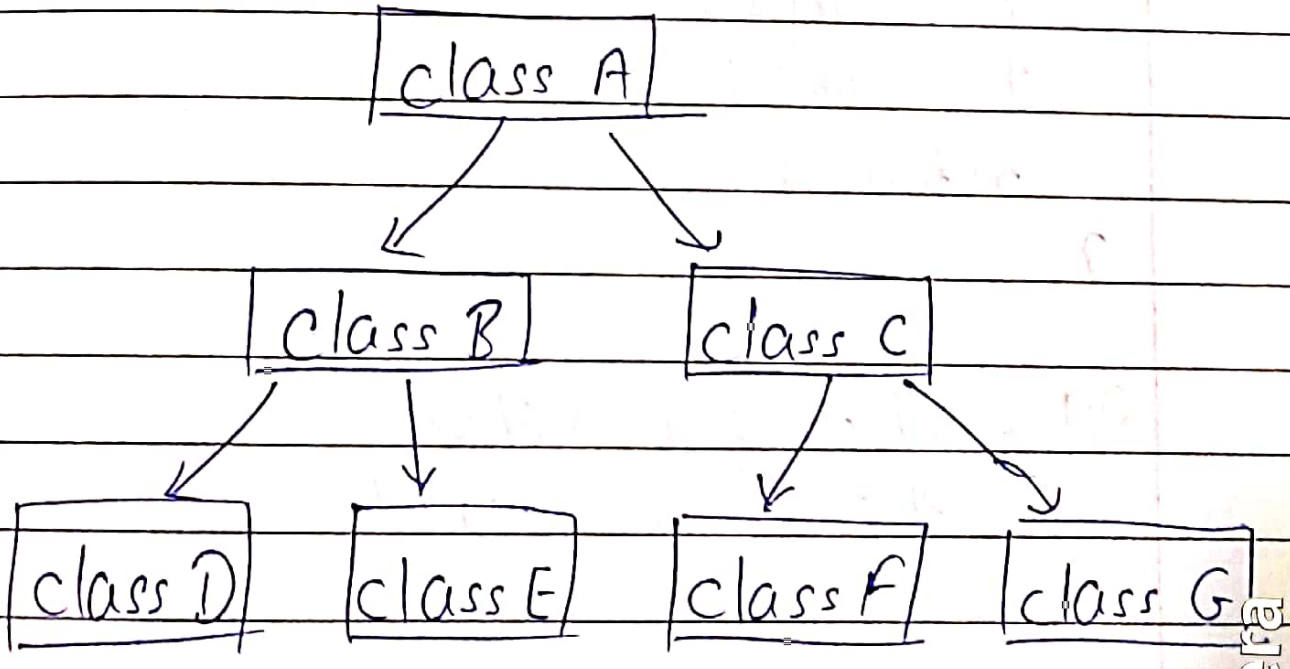
```
#include <iostream>
using namespace std;

class person {
    public:
        person(int x) {
            cout << "Person :: Person (int ) called" << endl;
        }
};
class faculty : public person {
    public:
```

```
    faculty(int x): person(x) {
        cout << "faculty::faculty(int) called"
                                        << endl;
    }
};

class student : public person {
  public:
    student(int x): person(x) {
        cout << "student::student(int) called" << endl;
    }
};

class TA : public faculty, public student {
  public:
    TA(int x): student(x), faculty(x) {
        cout << "TA::TA(int) called" << endl;
    }
};

int main() {
    TA ta1(30);
}
```

O/P :- ambiguity error.

In the above program, constructor of 'person' is called two times. Destructor of 'person' will also be called two times when object 'ta1' is destructed. so object 'ta1' has two copies of all members of 'person': this causes ambiguities. The solution to this problem is "~~virtual~~" "Virtual" keyword. we make the classes "faculty" and "student" as virtual base classes to avoid two copies of "person" in "TA". ~~for example.~~

☆ Hierarchical inheritance :- In this type of inheritance, more than one sub class is inherited from a single base class. i.e more than one derived class is created from a single base class.



Example :

```
#include <iostream>
using namespace std;

class vehicle {
    public:
        vehicle (){
            cout << " this is a vehicle " << endl;
        }
};

class Car : public vehicle {
};

class Bus : public vehicle {
};

int main {
    Car obj1;
    Bus obj2;
    return 0;
}
```

O/P :- this is a vehicle
this is a vehicle