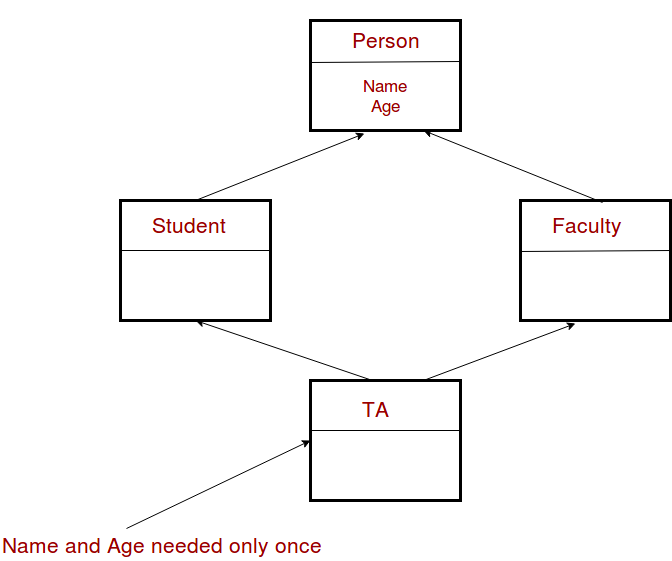
**The diamond problem** The diamond problem occurs when two superclasses of a class have a common base class. For example, in the following diagram, the TA class gets two copies of all attributes of Person class, this causes ambiguities.



#include<iostream>

using namespace std;

class Person {

// Data members of person

public:

Person(int x) { cout << "Person::Person(int ) called" << endl; }

};

class Faculty : public Person {

// data members of Faculty

public:

Faculty(int x):Person(x) {

cout<<"Faculty::Faculty(int ) called"<< endl;

}

};

class Student : public Person {

// data members of Student

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);

}

Person::Person(int ) called

Faculty::Faculty(int ) called

Person::Person(int ) called

Student::Student(int ) called

TA::TA(int ) called

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’ keyword*. We make the classes ‘Faculty’ and ‘Student’ as virtual base classes to avoid two copies of ‘Person’ in ‘TA’ class.

#include<iostream>

using namespace std;

class Person {

public:

Person(int x) { cout << "Person::Person(int ) called" << endl; }

Person() { cout << "Person::Person() called" << endl; }

};

class Faculty : virtual public Person {

public:

Faculty(int x):Person(x) {

cout<<"Faculty::Faculty(int ) called"<< endl;

}

};

class Student : virtual 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);

}

For example, consider the following program.

#include<iostream>

using namespace std;

class Person {

public:

Person(int x) { cout << "Person::Person(int ) called" << endl; }

Person() { cout << "Person::Person() called" << endl; }

};

class Faculty : virtual public Person {

public:

Faculty(int x):Person(x) {

cout<<"Faculty::Faculty(int ) called"<< endl;

}

};

class Student : virtual 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);

}

Output:

Person::Person() called

Faculty::Faculty(int ) called

Student::Student(int ) called

TA::TA(int ) called