Object Oriented Programming (CS1143)

Week 8

Department of Computer Science

Capital University of Science and Technology (CUST)

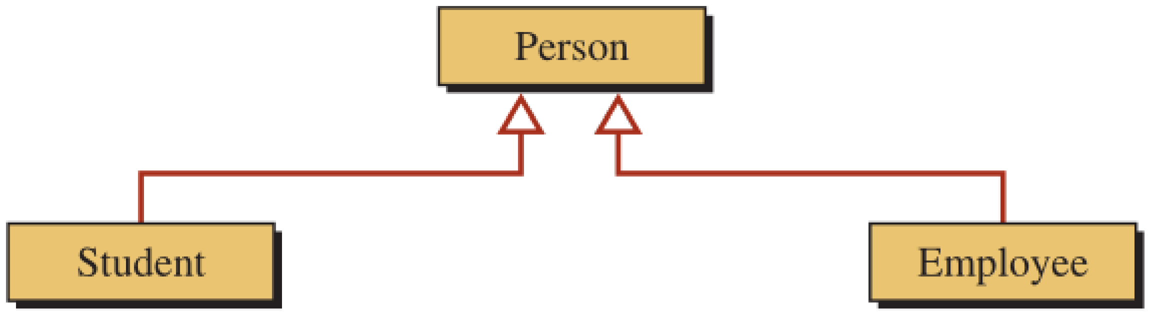
Outline

- Multi-level Inheritance
- Multiple Inheritance
- Diamond Problem
- Solve Diamond Problem (Virtual Inheritance)

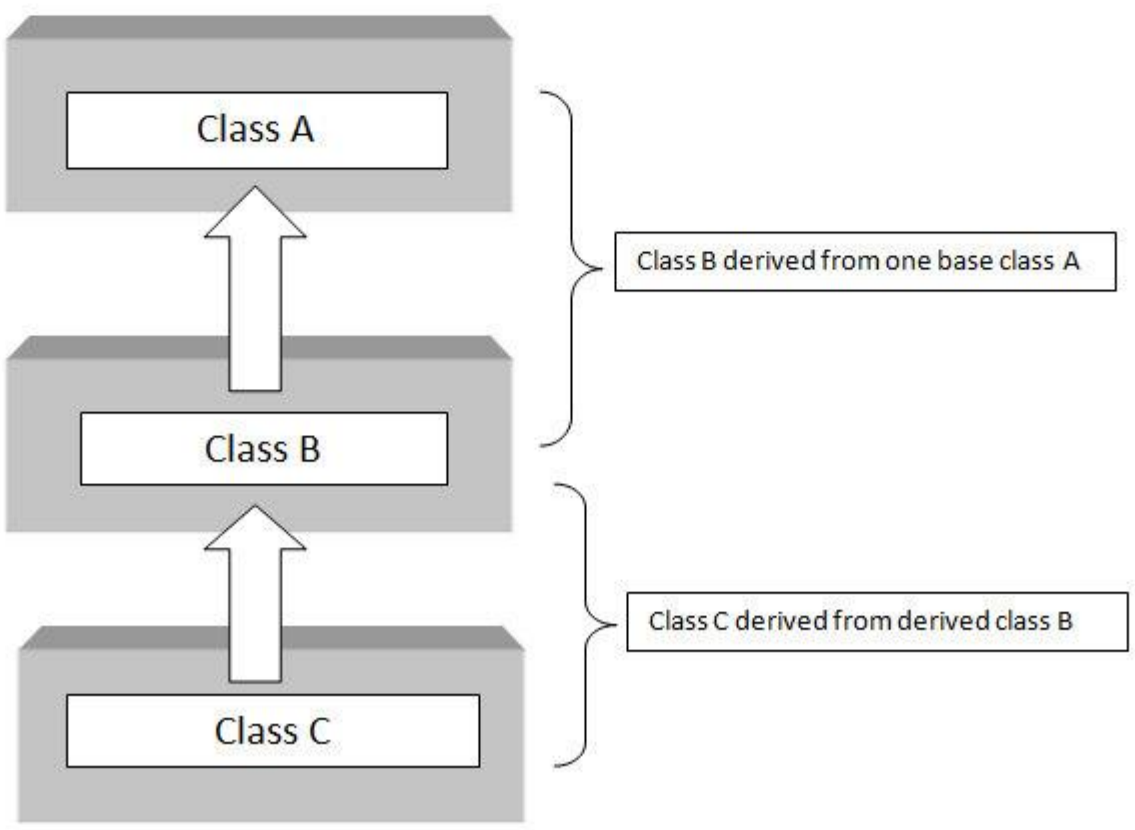
Class Hierarchies

- So far we have only seen one derived class inheriting from one base class. However all of the following cases are possible.
  1. Multiple classes can be derived from the same base class.
  2. A base class can also derive from another base class (Multi-level inheritance)
  3. A derived class can derive from multiple base classes (Multiple Inheritance)

Multiple Classes Deriving from the Same Base Class

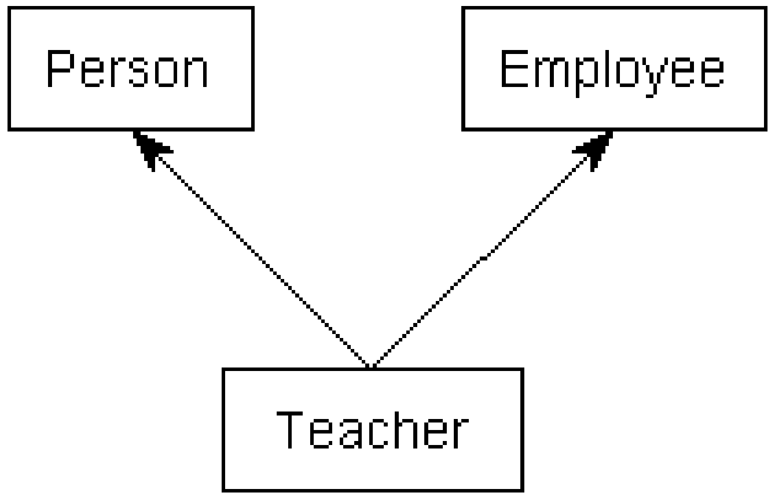


Multi-level Inheritance

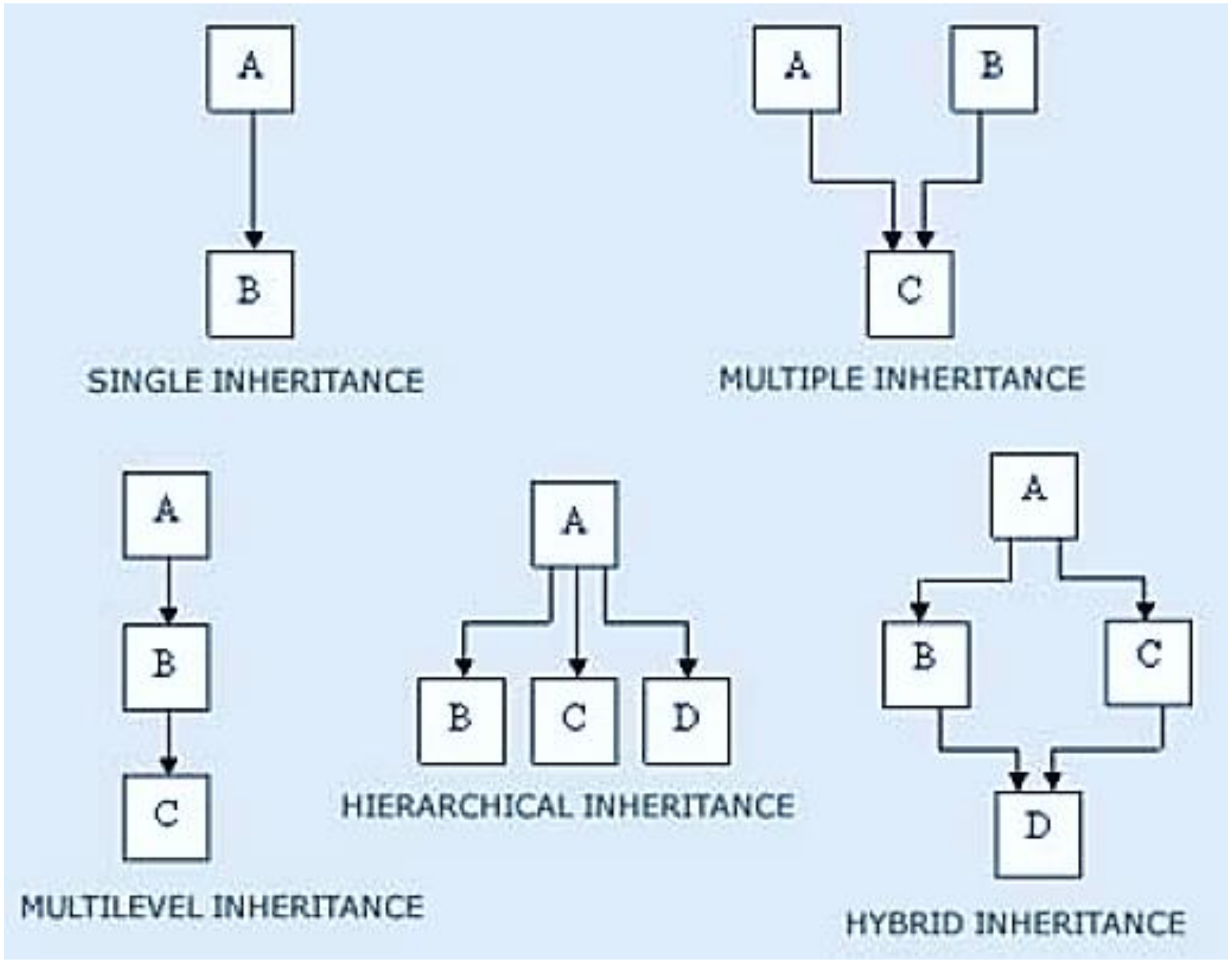

Class A
Class B
Class C
Class B derived from one base class A
Class C derived from derived class B

Multiple Inheritance

- A feature of object-oriented computer programming language in which an object or class can inherit characteristics and features from more than one parent object or parent class



Types of Inheritance



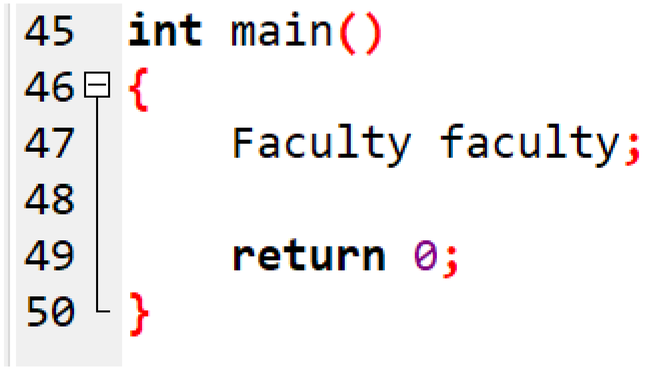
Multi-level Inheritance

- We will explain this concept using an example.

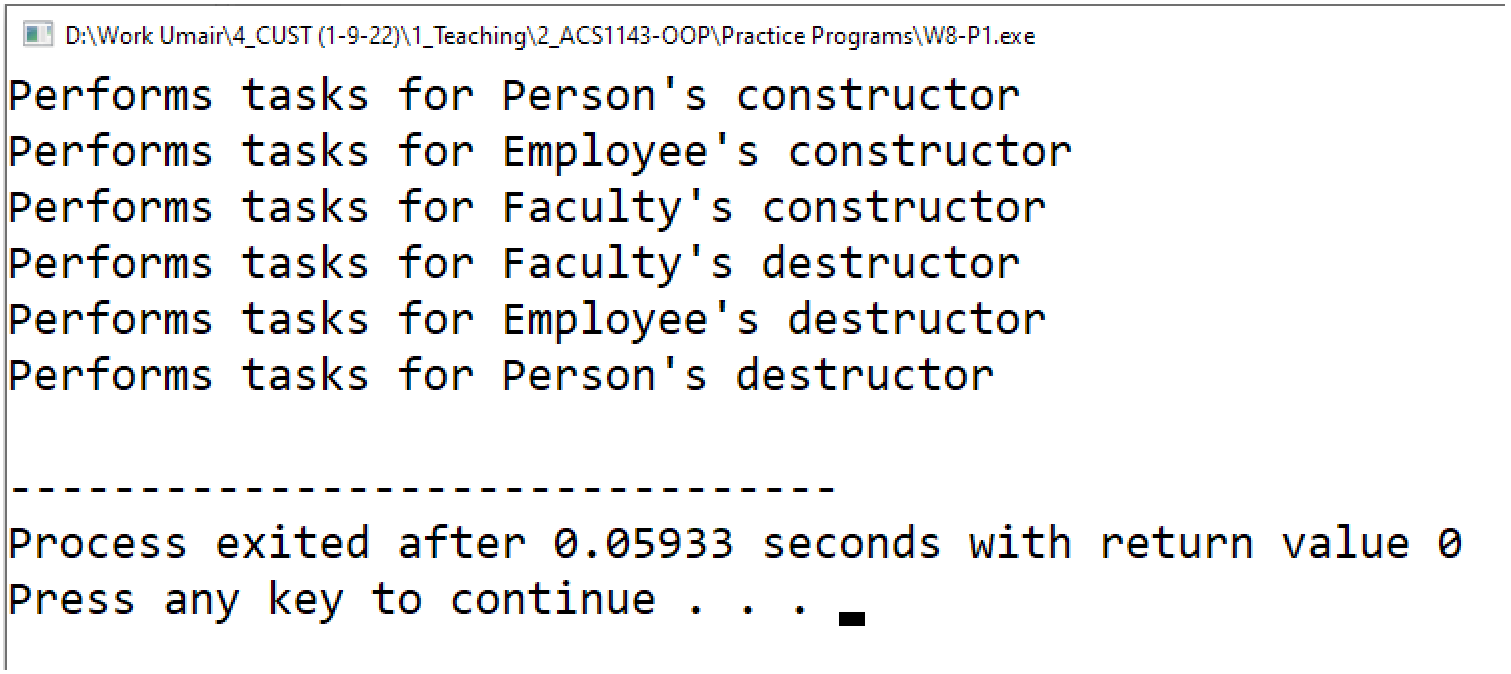
```cpp
#include <iostream>
using namespace std;
class Person
{
	public:
	Person()
	{
		cout << "Performs tasks for Person's constructor" << endl;
	}

	~Person()
	{
		cout << "Performs tasks for Person's destructor" << endl;
	}
};
```

```cpp
class Employee: public Person
{
	public:
	Employee()
	{
		cout << "Performs tasks for Employee's constructor" << endl;
	}

	~Employee()
	{
		cout << "Performs tasks for Employee's destructor" << endl;
	}
};
```

```cpp
class Faculty: public Employee
{
	public:
	Faculty()
	{
		cout << "Performs tasks for Faculty's constructor" << endl;
	}

	~Faculty()
	{
		cout << "Performs tasks for Faculty's destructor" << endl;
	}
};
```



```cpp
int main()
{
    Faculty faculty;

    return 0;
}
```

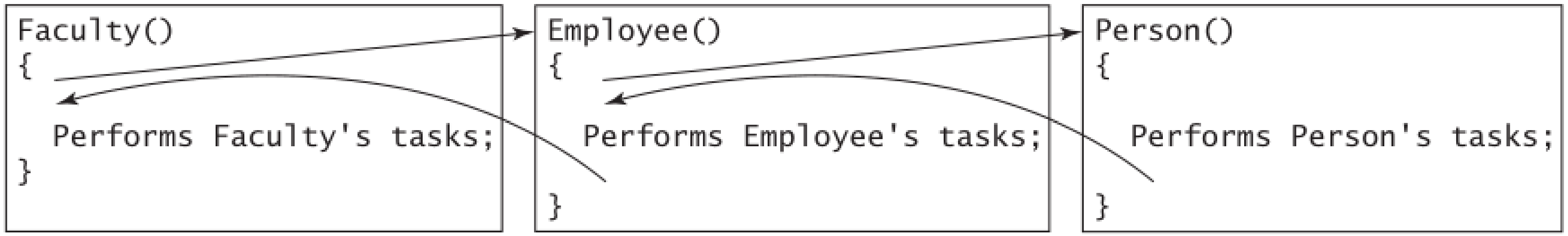


D:\Work Umair\4_CUST (1-9-22)\1_Teaching\2_ACS1143-OOP\Practice Programs\W8-P1.exe

```
Performs tasks for Person's constructor
Performs tasks for Employee's constructor
Performs tasks for Faculty's constructor
Performs tasks for Faculty's destructor
Performs tasks for Employee's destructor
Performs tasks for Person's destructor


--------------------------------
Process exited after 0.05933 seconds with return value 0
Press any key to continue . . .
```

Description

- The program creates an instance of Faculty.
- Since Faculty is derived from Employee and Employee is derived from Person , Faculty ’s constructor invokes Employee ’s constructor before it performs its own task.
- Employee ’s constructor invokes Person ’s constructor before it performs its own task,



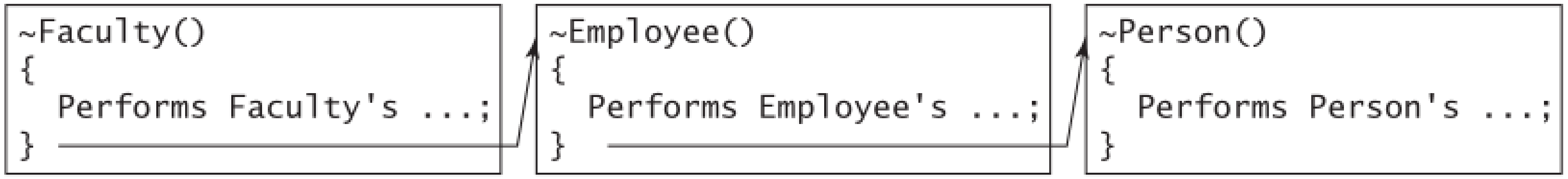
```
Faculty()
{

  Performs Faculty's tasks;
}
```

```
Employee()
{

  Performs Employee's tasks;

}
```

```
Person()
{

  Performs Person's tasks;

}
```

Description Continued

- When the program exits, the Faculty object is destroyed. So the Faculty ’s destructor is called, then Employee ’s, and finally Person ’s



```
~Faculty()
{
  Performs Faculty's ...;
}
```

```
~Employee()
{
  Performs Employee's ...;
}
```

```
~Person()
{
  Performs Person's ...;
}
```

Program Discussion (W8-P2-S23.cpp)

Outline

- Multi-level Inheritance
- Multiple Inheritance
- Diamond Problem
- Solve Diamond Problem (Virtual Inheritance)

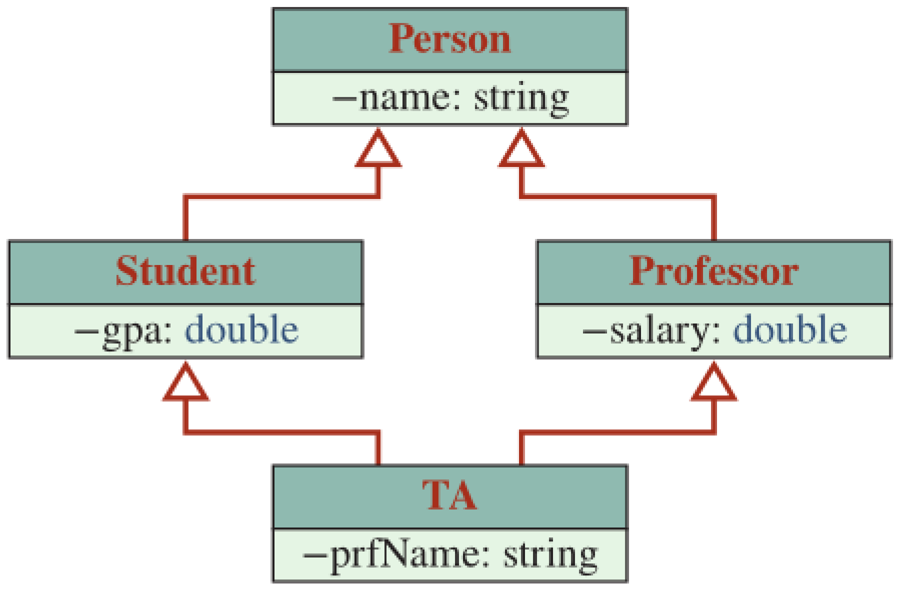
Multiple Inheritance

- C++ allows multiple inheritance, the derivation of a class from more than one class.
- As a simple example, we can have a class named TA (teaching assistant) that is inherited from two classes: Student and Professor

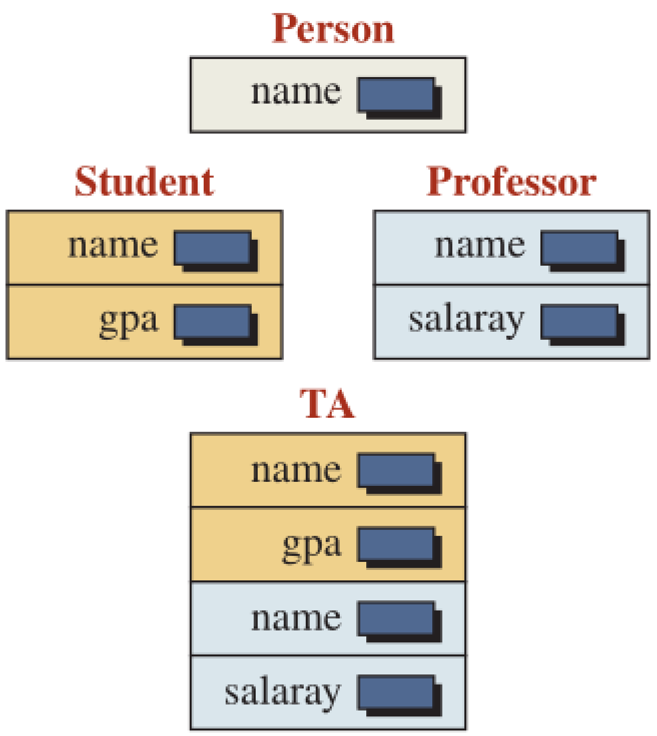
Program Discussion

Multiple Inheritance-Diamond Problem

- Suppose we have a class Person that defines one single data member, name.
- Both Student and Professor classes inherit from Person class.
- This data member is inherited in both the Student object and the Professor object.
- Since the TA (teaching assistant) class inherits the Student and Professor classes, the data member name is duplicated in the TA class.
- We cannot use inheritance in this case without removing the duplicate data member.

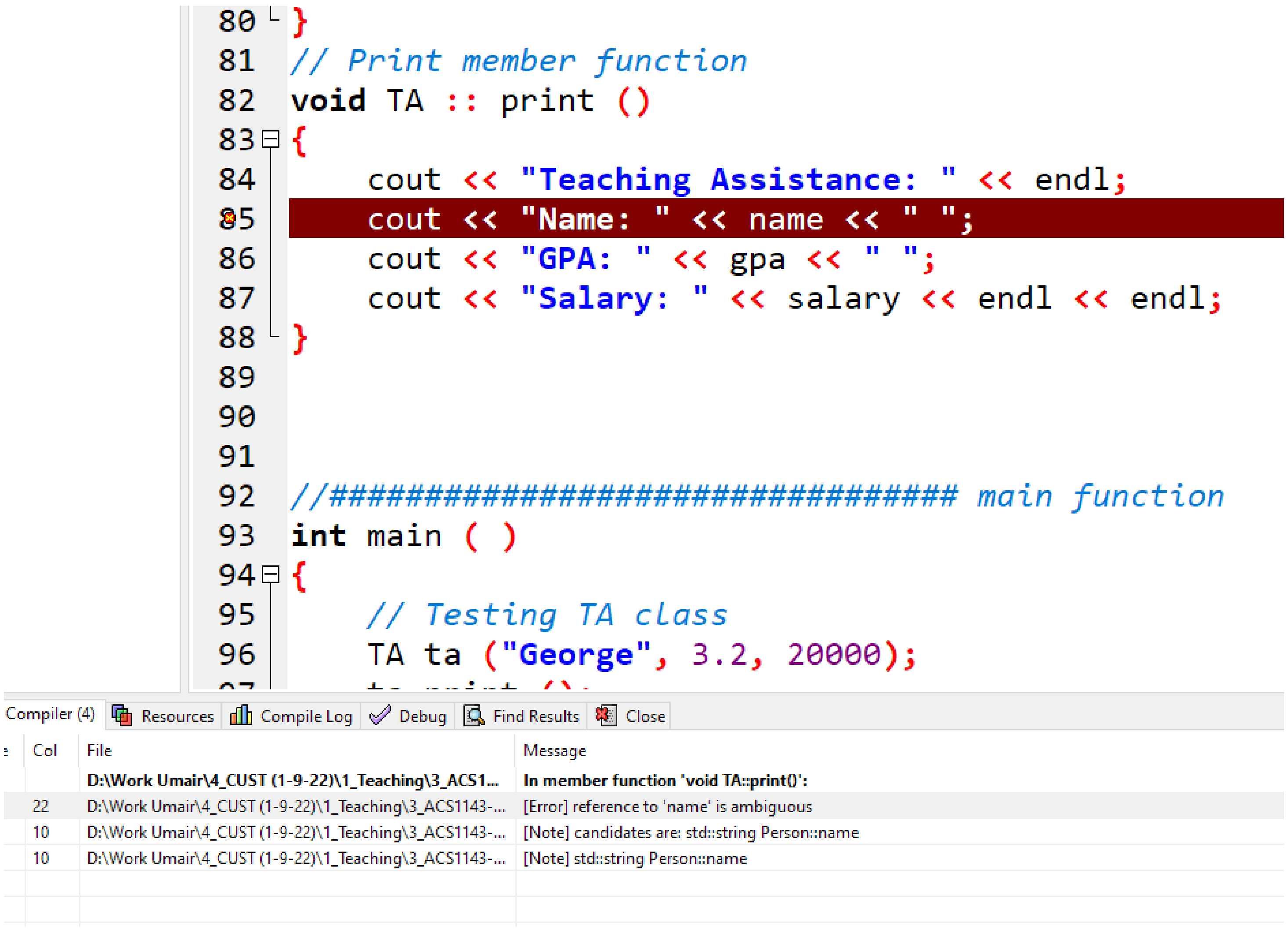


Classes



Objects

Program Discussion (W8-P4-S23.cpp)

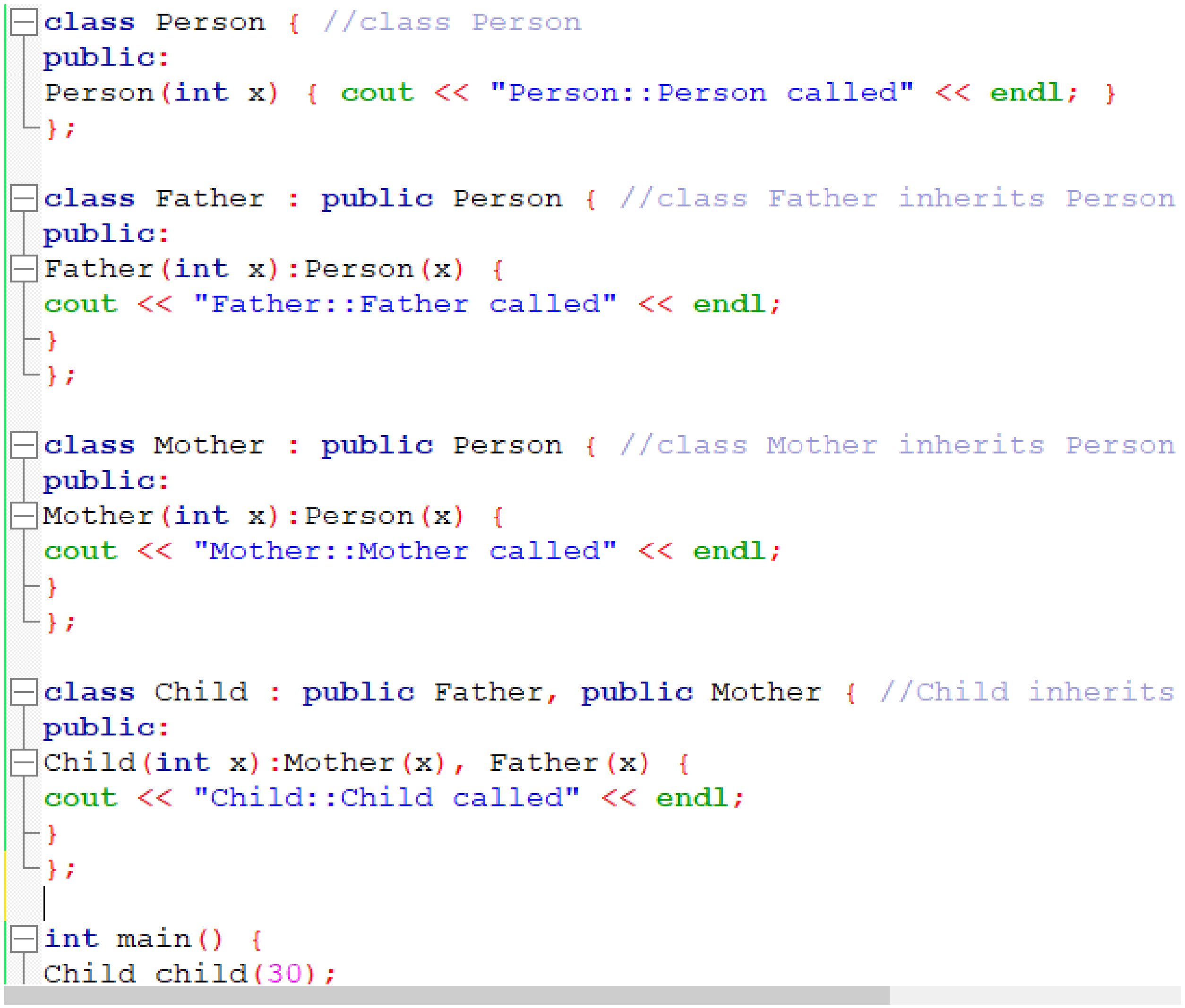


```cpp
}
// Print member function
void TA :: print ()
{
    cout << "Teaching Assistance: " << endl;
    cout << "Name: " << name << " ";
    cout << "GPA: " << gpa << " ";
    cout << "Salary: " << salary << endl << endl;
}



//################################## main function
int main ( )
{
    // Testing TA class
    TA ta ("George", 3.2, 20000);
```

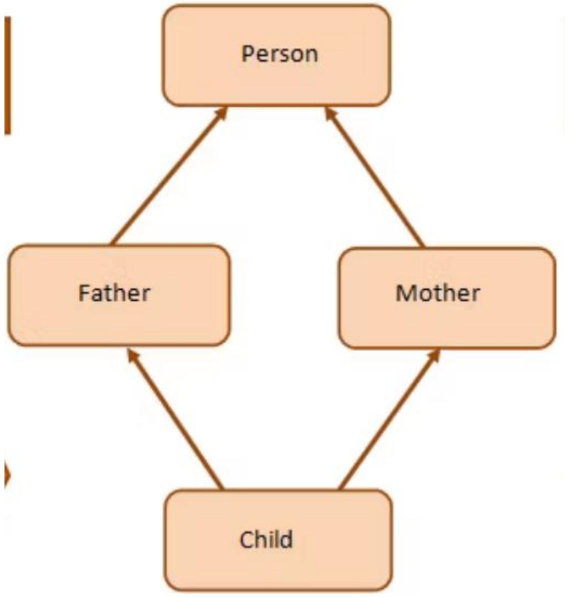
Col	File	Message
	D:\Work Umair\4_CUST (1-9-22)\1_Teaching\3_ACS1...	In member function 'void TA::print()':
22	D:\Work Umair\4_CUST (1-9-22)\1_Teaching\3_ACS1143-...	[Error] reference to 'name' is ambiguous
10	D:\Work Umair\4_CUST (1-9-22)\1_Teaching\3_ACS1143-...	[Note] candidates are: std::string Person::name
10	D:\Work Umair\4_CUST (1-9-22)\1_Teaching\3_ACS1143-...	[Note] std::string Person::name

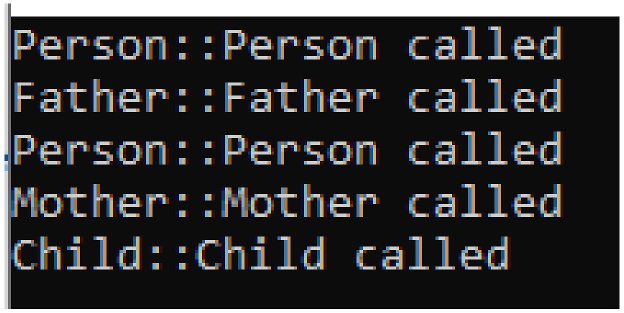
Solution

- Diamond problem also distrube the order of construction’s execution.
- One solution for the problem of duplicated shared data members in multiple inheritance is to use **virtual inheritance.**



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

class Father : public Person { //class Father inherits Person
public:
Father(int x):Person(x) {
cout << "Father::Father called" << endl;
}
};

class Mother : public Person { //class Mother inherits Person
public:
Mother(int x):Person(x) {
cout << "Mother::Mother called" << endl;
}
};

class Child : public Father, public Mother { //Child inherits
public:
Child(int x):Mother(x), Father(x) {
cout << "Child::Child called" << endl;
}
};

int main() {
Child child(30);
```





```
Person::Person called
Father::Father called
Person::Person called
Mother::Mother called
Child::Child called
```

Fix the diamond problem

- One may face such ambiguities while using multiple inheritance
  - This particular ambiguity is known as Diamond Problem
- The solution to this problem is by making inheritance virtual

i.e.

```
Class Faculty: virtual public Person
Class Student: virtual public Person
```

- This will ensure that only one sub-object of the Person class will be created for every TA object.
- **Hence, The Diamond Problem is fixed using virtual inheritance**

```cpp
#include<iostream>
using namespace std;
class Person { //class Person
public:
Person() { cout << "Person::Person() called" << endl; } //Base constructor
Person(int x) { cout << "Person::Person(int) called" << endl; }
};

class Father : virtual public Person { //class Father inherits Person
public:
Father(int x):Person(x) {
cout << "Father::Father(int) called" << endl;
}
};

class Mother : virtual public Person { //class Mother inherits Person
public:
Mother(int x):Person(x) {
cout << "Mother::Mother(int) called" << endl;
}
};

class Child : public Father, public Mother { //class Child inherits Father and Mother
public:
Child(int x):Mother(x), Father(x) {
cout << "Child::Child(int) called" << endl;
}
};

int main() {
Child child(30);
}
```

```
Person::Person() called
Father::Father(int) called
Mother::Mother(int) called
Child::Child(int) called
```

Multiple Inheritance (Fix the diamond problem)

- One thing to note about virtual inheritance is that even if the parameterized constructor of the Person class is explicitly called by Father and Mother class constructors through initialization lists, **only the dafault constructor of the Person class will be called**.
- This is because there's only a single instance of a virtual base class that's shared by multiple classes that inherit from it.
- To prevent the base constructor from running multiple times, the constructor for a virtual base class is not called by the class inheriting from it. Instead, the constructor is called by the constructor of the concrete class.