## **What is Diamond Problem in C++?**

In C++, inheritance is the concept that allows one class to inherit the properties and methods of another class. Multiple inheritance is one such type of inheritance that allows a class to inherit from more than one base class. While this feature provides greater flexibility in modelling real-world relationships, it also introduces complexities, one of which is the Diamond Problem.

### **Diamond Problem**

The Diamond Problem is an ambiguity error that arises in multiple inheritance when a derived class inherits from two or more base classes that share a common ancestor. This results in the inheritance hierarchy forming a diamond shape, hence the name "Diamond Problem." The ambiguity arises because the derived class has multiple paths to access members or methods inherited from the common ancestor, leading to confusion during method resolution and member access.

## **Example of Diamond Problem in C++**

C++

1

// C++ Program to illustrate the diamond problem

2

#include <iostream>

3

using namespace std;

4

​

5

// Base class

6

class Base {

7

public:

8

void fun() { cout << "Base" << endl; }

9

};

10

​

11

// Parent class 1

12

class Parent1 : public Base {

13

public:

14

};

15

​

16

// Parent class 2

17

class Parent2 : public Base {

18

public:

19

};

20

​

21

// Child class inheriting from both Parent1 and Parent2

22

class Child : public Parent1, public Parent2 {

23

};

24

​

25

int main()

26

{

27

Child\* obj = new Child();

28

obj->fun(); // Abiguity arises, as Child now has two copies of fun()

29

return 0;

30

}

**Output**

main.cpp:30:9: error: request for member ‘fun’ is ambiguous

30 | obj.fun(); // Ambiguity error

| ^~~

main.cpp:8:10: note: candidates are: ‘void Base::fun()’

8 | void fun() { cout << "Base" << endl; }

| ^~~

main.cpp:20:10: note: ‘void Base::fun()’

## **Solution to the Diamond Problem in C++**

C++ addresses the Diamond Problem using virtual inheritance. Virtual inheritance ensures that there is only one instance of the common base class, eliminating the ambiguity.

### **Example**

C++

1

// C++ Program to illustrate the use of virtual inheritance

2

// to resolve the diamond problem in multiple inheritance

3

#include <iostream>

4

using namespace std;

5

​

6

// Base class

7

class Base {

8

public:

9

void fun() { cout << "Base" << endl; }

10

};

11

​

12

// Parent class 1 with virtual inheritance

13

class Parent1 : virtual public Base {

14

public:

15

};

16

​

17

// Parent class 2 with virtual inheritance

18

class Parent2 : virtual public Base {

19

public:

20

};

21

​

22

// Child class inheriting from both Parent1 and Parent2

23

class Child : public Parent1, public Parent2 {

24

};

25

​

26

int main()

27

{

28

Child\* obj = new Child();

29

obj->fun(); // No ambiguity due to virtual inheritance

30

return 0;

31

}

**Output**

Base

**Another approach is to rename conflicting methods** in the derived classes to avoid ambiguity. By providing distinct names for methods inherited from different base classes, developers can eliminate ambiguity without resorting to virtual inheritance. However, this approach may lead to less intuitive code and increased maintenance overhead.