Digital Assignment – 3

**Name: Hari Krishna Shah**

**VIT ID: 21BCS0167**

## Ques 1. Analyse the ways in which ambiguity arises in multiple inheritance and find the ways in which it can be resolved. Explain with suitable example.

## Answer:

An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class. This can be solved by using a virtual base class. When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

Example:

```
class a

{

private:

…………….

public:

void abc() {}

…………….

…………….

};

class b

{

private:

…………….
public:
void abc() {}
…………….
…………….
};
class c : public a , public b
{
private:
…………….
public:
…………….
…………….
};
void main()
{
c obj;
obj.abc();//Error
……………..
}
```

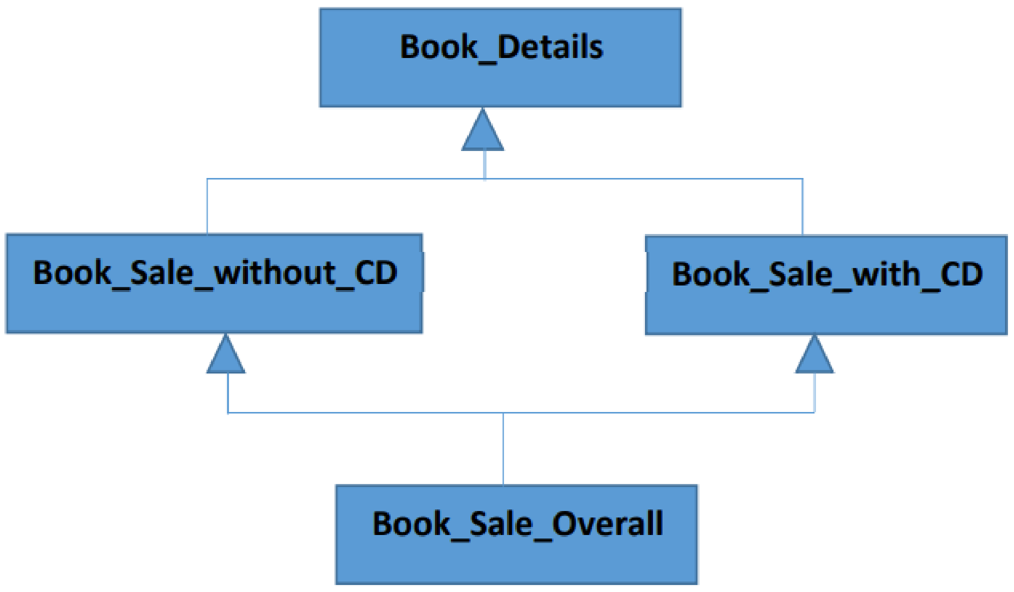
In this example, the two base classes a and b have function of same name abc() which are inherited to the derived class c. When the object of the class c is created and call the function abc() then the compiler is confused which base's class function is called by the compiler.

**Solution of ambiguity in multiple inheritance:** The ambiguity can be resolved by using the scope resolution operator to specify the class in which the member function lies as given below:

```
obj.a :: abc();
```

This statement invoke the function name abc() which lies in base class a.

## Ques 2. Implement the following hybrid inheritance.

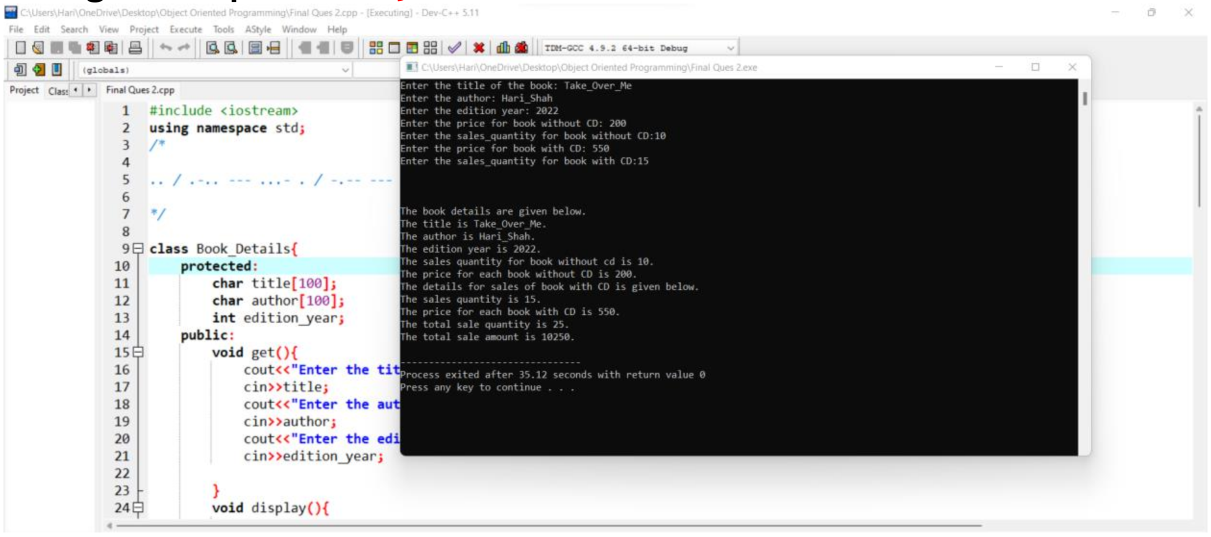


**Answer:**

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

.. / .-.. --- ...- . / -.-- --- ..- / -.-. .... .. - .-.
.-

*/

class Book_Details{
	protected:
		char title[100];
		char author[100];
		int edition_year;
	public:
		void get(){
			cout<<"Enter the title of the book: ";
			cin>>title;
			cout<<"Enter the author: ";
			cin>>author;
			cout<<"Enter the edition year: ";
			cin>>edition_year;

		}
		void display(){
			cout<<endl<<endl;
			cout<<"The book details are given 
below."<<endl;
			cout<<"The title is "<<title<<"."<<endl;
			cout<<"The author is "<<author<<"."<<endl;
			cout<<"The edition year is 
"<<edition_year<<"."<<endl;
		}
};

class Book_Sale_without_CD: public Book_Details{
	protected:
		int sales_quantity;
		float price;
	public:
		void gets(){
			Book_Details::get();
			cout<<"Enter the price for book without CD: 
";
			cin>>price;
			cout<<"Enter the sales_quantity for book 
without CD:";
			cin>>sales_quantity;
		}
		void display_class(){
			Book_Details::display();
			cout<<"The sales quantity for book without 
cd is "<<sales_quantity<<"."<<endl;
			cout<<"The price for each book without CD is 
"<<price<<"."<<endl;
		}


};
class Book_Sale_with_CD: public Book_Details{
	protected:
		int sale_quantity;
		float prices;
	public:
		void get(){
			cout<<"Enter the price for book with CD: ";
```

```
                cin>>prices;
                cout<<"Enter the sales_quantity for book 
with CD:";
                cin>>sale_quantity;
            }
            void display(){
                cout<<"The details for sales of book with CD 
is given below."<<endl;
                cout<<"The sales quantity is 
"<<sale_quantity<<"."<<endl;
                cout<<"The price for each book with CD is 
"<<prices<<"."<<endl;
            }

};

class Book_Overall:virtual public Book_Sale_without_CD, 
public Book_Sale_with_CD{
        protected:
            int total_sale_quantity;
            float total_sale_amount;
        public:
            void get_details(){
                Book_Sale_without_CD::gets();
                Book_Sale_with_CD::get();
            }

            void calculate_sale(){
                total_sale_quantity = 
Book_Sale_without_CD::sales_quantity + 
Book_Sale_with_CD::sale_quantity;
                total_sale_amount = 
(Book_Sale_without_CD::sales_quantity*Book_Sale_without_CD
::price) + 
(Book_Sale_with_CD::sale_quantity*Book_Sale_with_CD::price
s);
            }
            void display_details(){
                Book_Sale_without_CD::display_class();
                Book_Sale_with_CD::display();
                cout<<"The total sale quantity is 
"<<total_sale_quantity<<"."<<endl;
```

```
                cout<<"The total sale amount is "<<total_sale_amount<<"."<<endl;
        }

};

int main(){
     class Book_Overall b;
     b.get_details();
     b.calculate_sale();
     cout<<endl;
     b.display_details();
     return 0;
}


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



C:\Users\Hari\OneDrive\Desktop\Object Oriented Programming\Final Ques 2.cpp - [Executing] - Dev-C++ 5.11

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

.. / .-.. --- ...- . / -.-- ---

*/

class Book_Details{
    protected:
        char title[100];
        char author[100];
        int edition_year;
    public:
        void get(){
            cout<<"Enter the tit
            cin>>title;
            cout<<"Enter the aut
            cin>>author;
            cout<<"Enter the edi
            cin>>edition_year;

        }
        void display(){
```

C:\Users\Hari\OneDrive\Desktop\Object Oriented Programming\Final Ques 2.exe

```
Enter the title of the book: Take_Over_Me
Enter the author: Hari_Shah
Enter the edition year: 2022
Enter the price for book without CD: 200
Enter the sales_quantity for book without CD:10
Enter the price for book with CD: 550
Enter the sales_quantity for book with CD:15



The book details are given below.
The title is Take_Over_Me.
The author is Hari_Shah.
The edition year is 2022.
The sales quantity for book without cd is 10.
The price for each book without CD is 200.
The details for sales of book with CD is given below.
The sales quantity is 15.
The price for each book with CD is 550.
The total sale quantity is 25.
The total sale amount is 10250.

--------------------------------
Process exited after 35.12 seconds with return value 0
Press any key to continue . . .
```

## Ques 3. When do we require the virtual derivation of a class? Describe with a scenario.

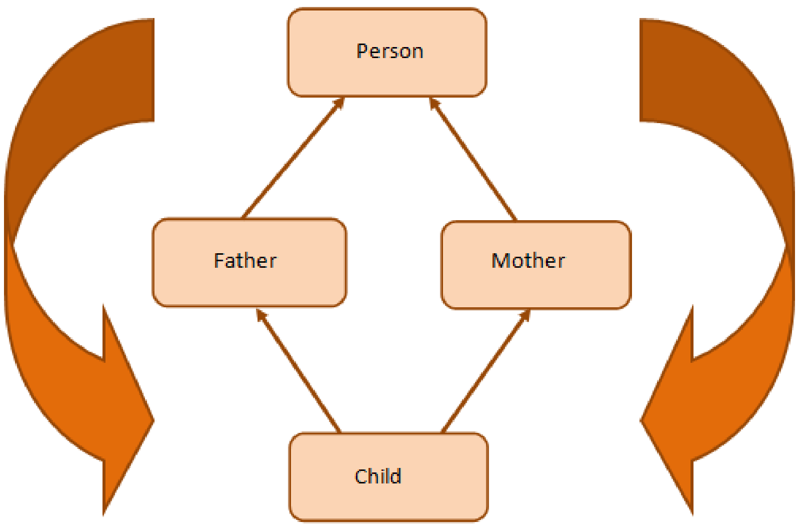
**Answer:** *Virtual inheritance* is a C++ technique that ensures that only one copy of a base class's member variables are inherited by second-level derivatives (a.k.a. grandchild derived classes). Without virtual inheritance, if two classes B and C inherit from class A, and class D inherits from both B and C, then D will contain two copies of A's member variables: one via B, and one via C. These will be accessible independently, using scope resolution.

Instead, if classes B and C inherit virtually from class A, then objects of class D will contain only one set of the member variables from class A.

This technique is useful when you have to deal with multiple inheritance and it's a way to solve the infamous diamond inheritance.

Consider the diamond problem given below, we will solve it using virtual function.

The Diamond Problem occurs when a child class inherits from two parent classes who both share a common grandparent class. This is illustrated in the diagram below:



Here, we have a class **Child** inheriting from classes **Father** and **Mother**. These two classes, in turn, inherit the class **Person** because both Father and Mother are Person.

As shown in the figure, class Child inherits the traits of class Person twice—once from Father and again from Mother. This gives rise to ambiguity since the compiler fails to understand which way to go.

This scenario gives rise to a diamond-shaped inheritance graph and is famously called "The Diamond Problem."

Example:

Code without fixing diamond problem

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

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

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

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

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

**Output:**

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

Now you can see the ambiguity here. The Person class constructor is called twice: once when the Father class object is created and next when the Mother class object is created. The properties of the Person class are inherited twice, giving rise to ambiguity.

Since the Person class constructor is called twice, the destructor will also be called twice when the Child class object is destructed.

Fixing the diamond problem.

The solution to the diamond problem is to use the **virtual** keyword. We make the two parent classes (who inherit from the same grandparent class) into virtual classes in order to avoid two copies of the grandparent class in the child class.

Code with diamond problem fixed

```
#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);
}
```

**Output:**

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