

# UNIT 5

# INHERITANCE

LH – 7HRS

PRESENTED BY:

**ER. SHARAT MAHARJAN**

OOP IN C++

# CONTENTS (LH – 7HRS)

- 5.1 Introduction to inheritance,
- 5.2 Derived Class and Base Class,
- 5.3 Access Specifiers (Private, Protected and Public),
- 5.4 Overriding member functions,
- 5.5 Types of inheritance (simple, multiple, hierarchical, multilevel, hybrid),
- 5.6 Ambiguity in multiple inheritance and in multipath hybrid inheritance,
- 5.7 Public and Private Inheritance,
- 5.8 Constructor and Destructor in derived classes,
- 5.9 Aggregation (class with in class)

# 5.1 Introduction to Inheritance

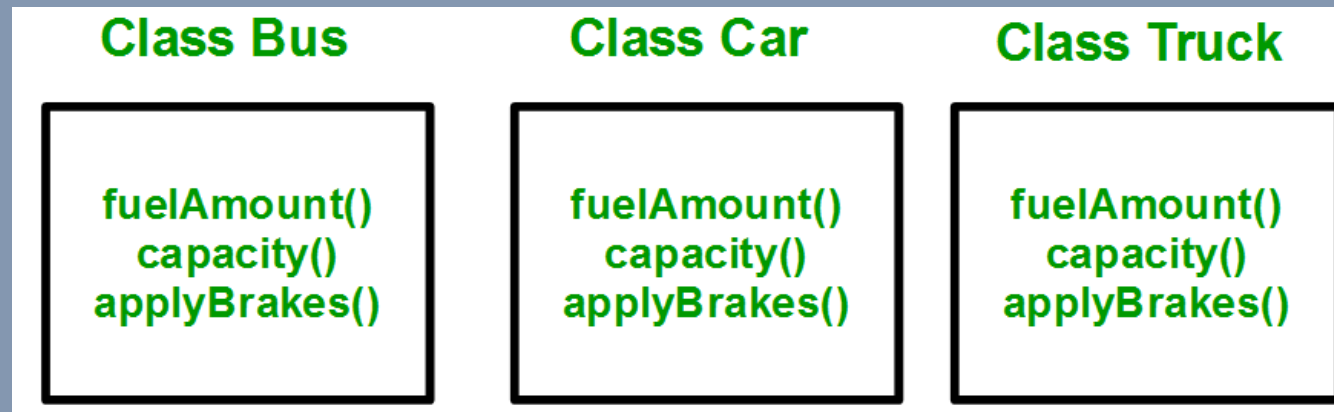
- The capability of a class to **derive properties and characteristics from another class** is called **Inheritance**.
- Inheritance is **one of the most important feature of Object Oriented Programming**.

**Sub Class**: The class that inherits properties from another class is called Sub class or **Derived Class**.

**Super Class**: The class whose properties are inherited by sub class is called **Base Class** or Super class.

## 5.2 Derived Class and Base Class

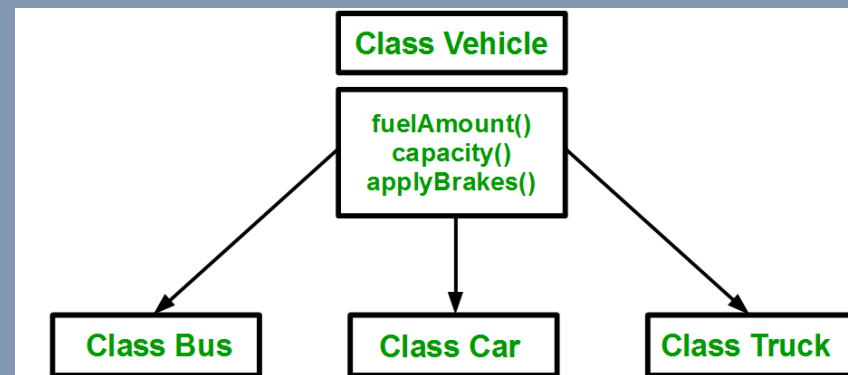
Consider a group of vehicles. We need to create classes for **Bus**, **Car** and **Truck**. The methods **fuelAmount()**, **capacity()**, **applyBrakes()** will be same for all of the three classes. If we create these classes **avoiding inheritance** then **we have to write all of these functions** in each of the three classes as shown in below figure:



We can clearly see that **above process results in duplication of same code 3 times**. This increases the chances of error and data redundancy. **To avoid** this type of situation, **inheritance is used**.

- If we **create a class Vehicle** and **write these three functions in it** and **inherit the rest of the classes from the vehicle class**, then we can simply **avoid the duplication of data and increase re-usability**.

Look at the below diagram in which the three classes are inherited from vehicle class:



**Using inheritance**, we have to write the **functions only one time instead of three times** as we have inherited rest of the three classes from base class(Vehicle).

**Implementing inheritance in C++:** For creating a sub-class which is inherited from the base class we have to follow the below syntax.

```
class subclass_name : access_mode base_class_name{
    //body of subclass
};
```

Here, **subclass\_name** is the name of the sub class, **access\_mode** is the mode in which you want to inherit this sub class for example: public, private etc. and **base\_class\_name** is the name of the base class from which you want to inherit the sub class.

### Example: LAB 1:

```
#include<iostream>
#include<conio.h>
using namespace std;
//Base class
class Parent{
    public:
        int pid;
};
//Sub class inheriting from Base Class(Parent)
class Child: public Parent{
    public:
        int cid; };

//main function
int main(){
    Child c;    //an object of class child has all data members and member functions of class parent
    c.pid=10;
    c.cid=20;
    cout<<"Parent id="<<c.pid<<endl;
    cout<<"Child id="<<c.cid<<endl;
    getch();
    return 0; }
```

### OUTPUT:

```
Parent id=10
Child id=20
```

In the above program the 'Child' class is publicly inherited from the 'Parent' class so the public data members of the class 'Parent' will also be inherited by the class 'Child'.

## 5.3 Access Specifiers (Private, Protected and Public),

In C++, there are three access specifiers:

- **public** - members are accessible from outside the class.
- **protected** – members can't be accessed from outside the class, however, they can be accessed in inherited classes.
- **private** – members can't be accessed from outside the class.

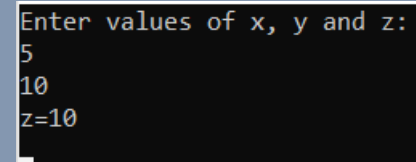
Inheritance and Accessibility			
Access Specifier	Accessible from Own Class	Accessible from Derived Class	Accessible from Objects Outside Class
<i>public</i>	yes	yes	yes
<i>protected</i>	yes	yes	no
<i>private</i>	yes	no	no



### Example: Lab 2:

```
#include<iostream>
#include<conio.h>
using namespace std;
class Parent{
    private:
        int x;
    protected:
        int y;
    public:
        int z;
};
class Child: public Parent{
    public:
    void setData(){
        cout<<"Enter values of x, y and z:"<<endl;
        //cin>>x;    //invalid, generates error because x is private
        cin>>y>>z;
    }
};
int main(){
    Child c;
    c.setData();
    //cout<<"x="<<c.x<<endl;    //invalid, generates error because x is private
    //cout<<"y="<<c.y<<endl;    //invalid, generates error because y is protected
    cout<<"z="<<c.z<<endl;    //valid
    getch();
    return 0;}
```

### OUTPUT:

A screenshot of a terminal window with a black background and white text. It shows the output of the program: "Enter values of x, y and z:" followed by three lines of input: "5", "10", and "z=10". A white cursor is visible at the end of the last line.

```
Enter values of x, y and z:
5
10
z=10
_
```

## 5.4 Overriding member functions

- We can define the functions in derived class having same name and **signature** as that of **base class** which is called **function overriding**.
- In such situations **base class** have **two versions of same function** one **derived from base** and **another defined in the base class itself**.
- If we call the overridden function by using the object of derived class version, method defined in derived class is invoked.
- We can call the version of method derived from base class as below:  
**obj.classname::method\_name;**

### Example: LAB 3:

```
#include<iostream>
#include<conio.h>
using namespace std;
class Parent{
    public:
        void show(){
            cout<<"This is class Parent."<<endl;
        }
};
class Child: public Parent{
    public:
        void show(){
            cout<<"This is class Child."<<endl;
        }
};
int main(){
    Child c;
    c.show();
    c.Parent::show();
    getch();
    return 0;
}
```

### OUTPUT:

```
This is class Child.
This is class Parent.
```

## 5.5 Types of inheritance

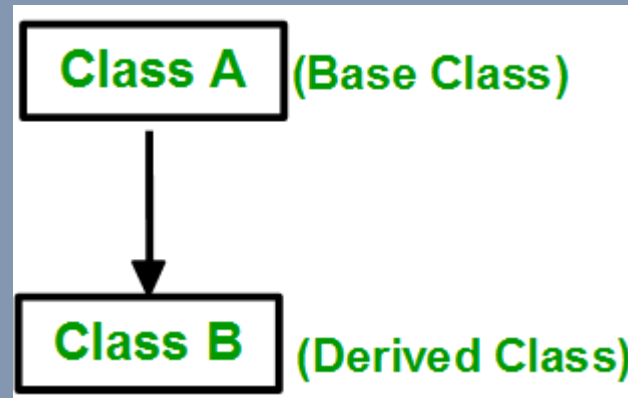
A class can inherit properties from one or more classes and from one or more levels.

On the basis of this concept, there are five forms of inheritance:

1. Single inheritance
2. Multiple inheritance
3. Hierarchical inheritance
4. Multilevel inheritance
5. Hybrid inheritance

## 1. Single inheritance:

- In single inheritance, a class is allowed to inherit from only one class.  
i.e. one sub class is inherited by one base class only.



Syntax:

```
class subclass_name : access_mode base_class{  
    //body of subclass  
};
```

#### Example: LAB 4

```
// C++ program to explain
// Single inheritance
#include <iostream>
using namespace std;
// base class
class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a vehicle." << endl;
    }
};
// sub class derived from a single base classes
class Car: public Vehicle{
};
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0; }
```

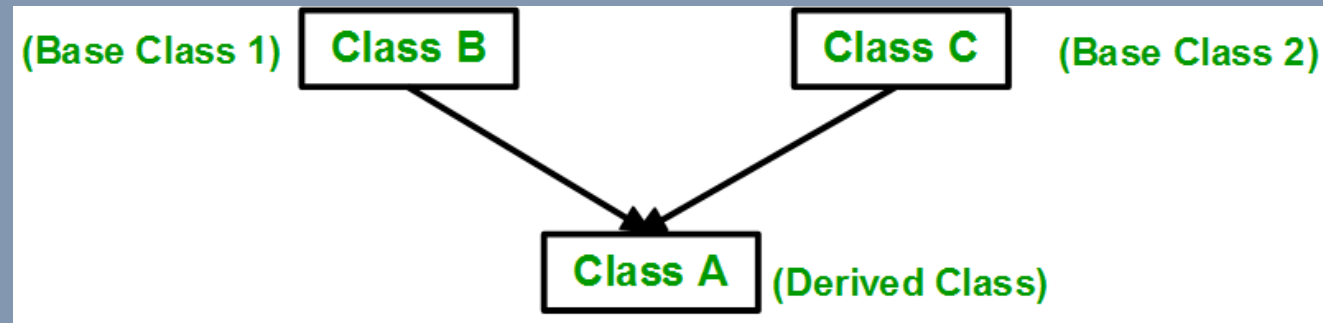
#### OUTPUT:

A terminal window with a black background and white text. The text "This is vehicle." is displayed on the first line, followed by a cursor (a small white vertical bar) on the second line.

```
This is vehicle.
_
```

## 2. 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**.



```
class subclass_name : access_mode base_class1, access_mode  
base_class 2, ....{  
    //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.

**LAB 5:** // C++ program to explain multiple inheritance

```
#include <iostream>
using namespace std;
// first base class
class Vehicle {
public:
    Vehicle(){
        cout << "This is a Vehicle." << endl;
    }
};
// second base class
class FourWheeler {
public:
    FourWheeler(){
        cout << "This is a 4 wheeler Vehicle." << endl;
    }
};
// sub class derived from two base classes
class Car: public Vehicle, public FourWheeler {
};
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0; }
```

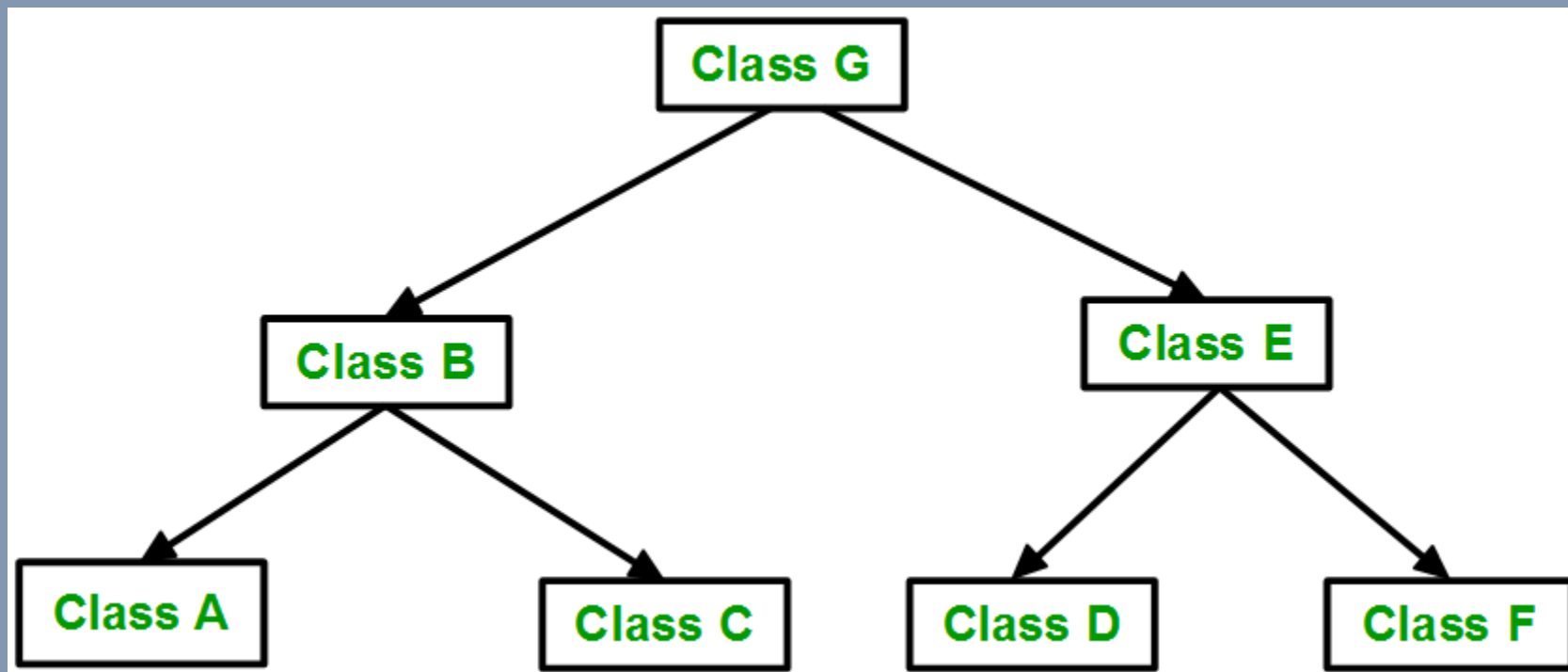
**OUTPUT:**

```
This is a Vehicle.
This is a 4 wheeler Vehicle.
```



### 3. 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.**



## LAB 6: // C++ program to implement Hierarchical Inheritance

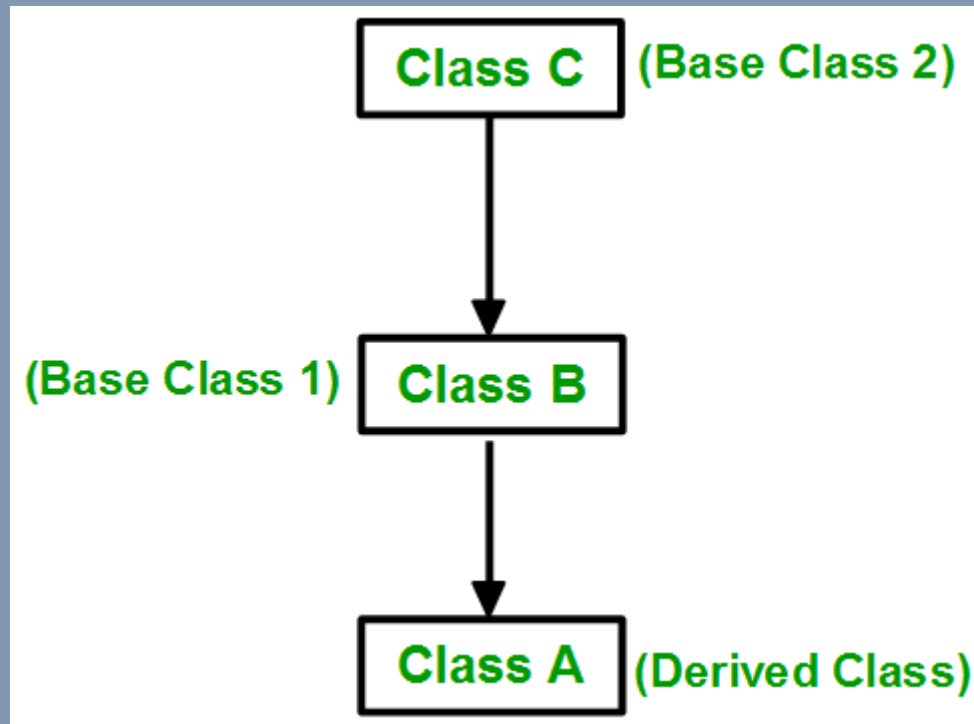
```
#include <iostream>
using namespace std;
// base class
class Vehicle
{
public:
    Vehicle(){
        cout << "This is a Vehicle." << endl;
    }
};
// first sub class
class Car: public Vehicle
{
};
// second sub class
class Bus: public Vehicle
{
};
int main()
{
    // creating object of sub class will
    // invoke the constructor of base class
    Car obj1;
    Bus obj2;
    return 0; }
```

### OUTPUT:

```
This is a Vehicle.
This is a Vehicle.
```

#### 4. Multilevel inheritance:

- In this type of inheritance, a **derived class** is created from another **derived class**.



**LAB 7:** // C++ program to implement Multilevel Inheritance

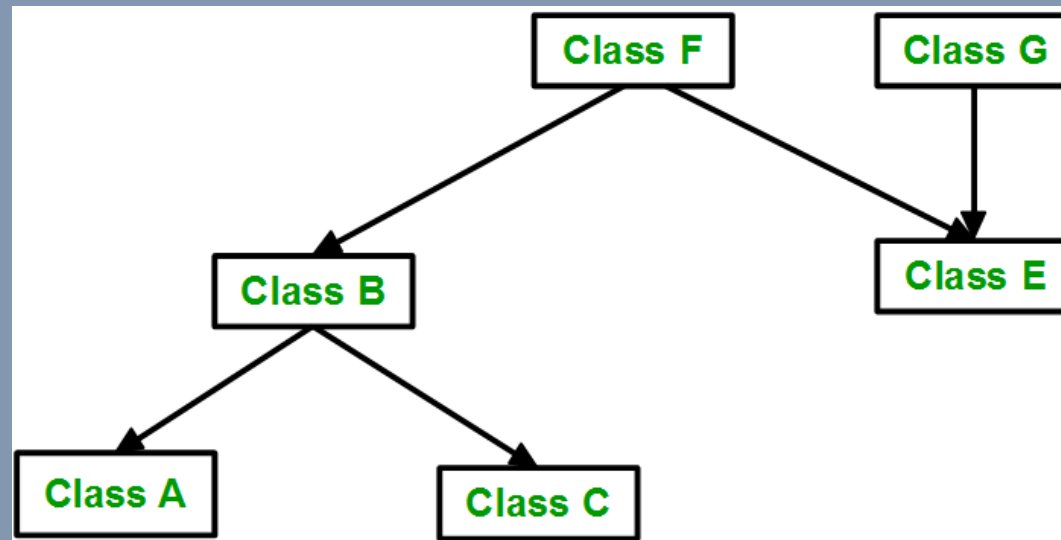
```
#include <iostream>
using namespace std;
// base class
class Vehicle{
public:
    Vehicle(){
        cout << "This is a Vehicle." << endl;
    }
};
// first sub_class derived from class vehicle
class FourWheeler: public Vehicle
{ public:
    FourWheeler(){
        cout<<"This is 4 Wheeler."<<endl;
    }
};
// sub class derived from the derived base class fourWheeler
class Car: public FourWheeler{
public:
    Car(){
        cout<<"This is a Car."<<endl;
    }
};
// main function
int main()
{
    //creating object of sub class will invoke the constructor of base classes
    Car obj;
    return 0; }
```

**OUTPUT:**

```
This is a Vehicle.
This is 4 Wheeler.
This is a Car.
_
```

## 5. Hybrid inheritance:

- Hybrid Inheritance is implemented by combining more than one type of inheritance.
- For example: Combining Hierarchical inheritance and Multiple Inheritance.
- Below image shows the combination of hierarchical and multiple inheritance:



**Lab 8:** // C++ program for Hybrid Inheritance

```
#include <iostream>
using namespace std;
// base class
class Vehicle{
public:
    Vehicle(){
        cout << "This is a Vehicle." << endl;
    }
};
//base class
class Fare{
public:
    Fare(){
        cout<<"Fare of vehicle."<<endl;
    }
};
// first sub class
class Car: public Vehicle{           //hierarchical due to both Car and Bus inherits Vehicle class
};
// second sub class
class Bus: public Vehicle, public Fare{           //multiple since Bus inherits more than one class
public:
    Bus(){
        cout<<"This is a Bus."<<endl;
    }
};
// main function
int main(){
    // creating object of sub class will
    // invoke the constructor of base class
    Bus obj2;
    return 0; }
```

**OUTPUT:**

```
This is a Vehicle.
Fare of vehicle.
This is a Bus.
```

## 5.6 Ambiguity in multiple inheritance and in multipath hybrid inheritance

An ambiguity in inheritance arises in two situations: in case of multiple inheritances and in case of hybrid multipath inheritance.

### 1. Ambiguity in Multiple Inheritance:

Suppose two base classes have an exactly similar member. Also, suppose a class derived from both of these base classes has not this member. Then, if we try to access this member from the objects of the derived class, it will be ambiguous. We can remove this ambiguity by using the syntax:

`obj.classname::variablename or methodname`

## LAB 9: Avoiding ambiguity in multiple inheritance using scope resolution operator:

```
#include<iostream>
#include<conio.h>
using namespace std;
class A{
    public:
        int x;
};
class B{
    public:
        int x;
};
class C: public A, public B{
};
int main(){
    C c;
    //c.x=10;    - ambiguous, will not compile
    c.A::x=10;   C le A ma x ko value 10 halyo
    c.B::x=20;
    cout<<"Value of x in class A="<<c.A::x<<endl;
    cout<<"Value of x in class B="<<c.B::x<<endl;
    getch();
    return 0; }
```

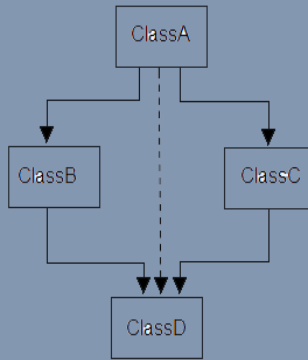
### OUTPUT:

```
Value of x in class A=10
Value of x in class B=20
```



## 2. Ambiguity in Multipath Inheritance:

- Another kind of ambiguity arises if we derive a class from two classes that are each derived from the same class, which is called multipath hybrid inheritance.
- In this case, public or protected member of grandparent is derived in the child class twice which creates confusion to the compiler.
- We can remove this kind of ambiguity by using the concept of **virtual base classes**.
- Consider a figure in right side, this creates a **diamond-shaped inheritance tree** and **all the public and protected members from class A inherited into class D twice** once through the path **A->B->D** and again through the path **A->B->C**. This causes ambiguity and should be avoided. For this, we make direct **base classes (B and C) virtual base classes**. When this happens, compiler creates direct path from A to D.



## LAB 10: Avoiding ambiguity in multipath hybrid inheritance using virtual base class:

```
#include<iostream>
#include<conio.h>
using namespace std;
class A{
    public:
        int a;
};
class B: virtual public A{// virtual keyword is used so that the grandchild class will get only one copy of parent class data members and functions.
    public:
        int b;
};
class C: virtual public A{
    public:
        int c; };
class D: public B, public C{
    public:
        int d; };

int main(){
    D obj;
    obj.a=100;
    obj.b=20;
    obj.c=30;
    obj.d=40;
    cout<<"Value of a="<<obj.a<<endl;
    cout<<"Value of b="<<obj.b<<endl;
    cout<<"Value of c="<<obj.c<<endl;
    cout<<"Value of d="<<obj.d<<endl;
    getch();
    return 0; }
```

### OUTPUT:

```
Value of a=100
Value of b=20
Value of c=30
Value of d=40
```

# 5.7 Public and Private Inheritance

## Modes of Inheritance

- 1.Public mode:** If we derive a sub class from a **Public base class**. Then the **public member of the base class** will become **public in the derived class** and **protected members** of the base class will **become protected in derived class**.
- 2.Protected mode:** If we derive a sub class from a **Protected base class**. Then **both public member and protected members** of the base class will **become protected in derived class**.
- 3.Private mode:** If we derive a sub class from a **Private base class**. Then **both public member and protected members** of the base class will **become Private in derived class**.

**Note :** The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed. For example, Classes B, C and D all contain the variables x, y and z in below example. It is just question of access.

Base class member access specifier	Type of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

// C++ Implementation to show that a derived class doesn't inherit access to private data members. However, it does inherit a full parent object

class A

{

public:

int x;

protected:

int y;

private:

int z; };

class B : public A

{

// x is public

// y is protected

// z is not accessible from B };

class C : protected A

{

// x is protected

// y is protected

// z is not accessible from C };

class D : private A // 'private' is default for classes

{

// x is private

// y is private

// z is not accessible from D };

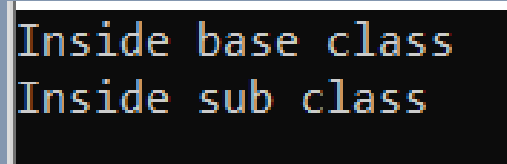
## 5.8 Constructor and Destructor in derived classes

pardaina

- Whenever we **create an object of a class**, the **default constructor of that class is invoked automatically** to initialize the members of the class.
- If we inherit a class from another class and create an object of the derived class, it is clear that the default constructor of the derived class will be invoked but before that the default constructor of all of the base classes will be invoke, i.e. the **order of invocation is that the base class's default constructor will be invoked first and then the derived class's default constructor will be invoked**.
- The constructor of **base class is called first to initialize all the inherited members**.

```
// C++ program to show the order of constructor call in single inheritance
#include <iostream>
using namespace std;
// base class
class Parent
{
    public:
    // base class constructor
    Parent()
    {
        cout << "Inside base class" << endl;
    };
}
// sub class
class Child : public Parent
{
    public:
    //sub class constructor
    Child()
    {
        cout << "Inside sub class" << endl;
    };
}
// main function
int main() {
    // creating object of sub class
    Child obj;
    return 0; }
```

OUTPUT:



```
Inside base class
Inside sub class
```

```
// C++ program to show the order of constructor calls in Multiple Inheritance

#include <iostream>

using namespace std;

// first base class

class Parent1
{
public:
    // first base class's Constructor
    Parent1()
    {
        cout << "Inside first base class" << endl; } };

// second base class

class Parent2
{
public:
    // second base class's Constructor
    Parent2()
    {
        cout << "Inside second base class" << endl; } };

// child class inherits Parent1 and Parent2

class Child : public Parent1, public Parent2
{
public:
    // child class's Constructor
    Child()
    {
        cout << "Inside child class" << endl; } };

// main function

int main() {

    // creating object of class Child

    Child obj1;

    return 0;

}
```

OUTPUT:

```
Inside first base class
Inside second base class
Inside child class
```

## Destructors:

- They are called automatically in the reverse order of constructors.
- The derived class destructor is executed first and then the destructor in the base class is executed.
- In case of multiple inheritances, the derived class destructor are executed in the order in which they appear in the definition of the derived class.
- Similarly, in a multilevel inheritance, the destructors will be executed in the order of inheritance.



```
//Destructors under Inheritance
```

```
#include<iostream>
```

```
#include<conio.h>
```

```
using namespace std;
```

```
class A{
```

```
    public:
```

```
        ~A(){
```

```
            cout<<"Class A Destructor"<<endl;
```

```
        }
```

```
};
```

```
class B: public A{
```

```
    public:
```

```
        ~B(){
```

```
            cout<<"Class B Destructor"<<endl;
```

```
        }
```

```
};
```

```
class C: public B{
```

```
    public:
```

```
        ~C(){
```

```
            cout<<"Class C Destructor"<<endl;
```

```
        }
```

```
};
```

```
int main(){
```

```
    {
```

```
        C x;
```

```
        //destructor is called at this point
```

```
    }
```

```
    getch();
```

```
    return 0; }
```

### OUTPUT:

```
Class C Destructor  
Class B Destructor  
Class A Destructor
```

## 5.9 Aggregation

- Inheritance is often called a “kind of” or “is a” relationship.
- In inheritance, if a class B is derived from a class A, we can say “B is a kind of A”. This is because B has all the characteristics of A, and in addition some of its own.
- There is another type of relationship, called a “has a” relationship or containership.
- In object oriented programming, **has a relationship** occurs when **one object is contained in another**.

**LAB 11:**

```
#include<iostream>
#include<conio.h>
using namespace std;
class Employee{
    int eid,salary;
    public:
        void setData(){
            cout<<"Enter employee id and salary:"<<endl;
            cin>>eid>>salary;}
        void getData(){
            cout<<"Employee ID="<<eid<<endl<<"Employee salary="<<salary<<endl;}
};
class Company{ //company has an employee relationship
    int cid;
    string cname;
    Employee e;
    public:
        void setData(){
            cout<<"Enter company id and name:"<<endl;
            cin>>cid>>cname;
            e.setData();
        }
        void getData(){
            cout<<"Company ID="<<cid<<endl<<"Company name="<<cname<<endl;
            e.getData();
        }
};
int main(){
    Company c;
    c.setData();
    c.getData();
    getch();
    return 0; }
```

**OUTPUT:**

```
Enter company id and name:
1
Prime
Enter employee id and salary:
2
30000
Company ID=1
Company name=Prime
Employee ID=2
Employee salary=30000
```

THANK YOU FOR YOUR ATTENTION