SRM UNIVERSITY AP
Amaravati

Object Oriented Programming using C++

CSE-III

CSE206-OOP

Dr. Priyanka Singh

UNIT IV: Inheritance

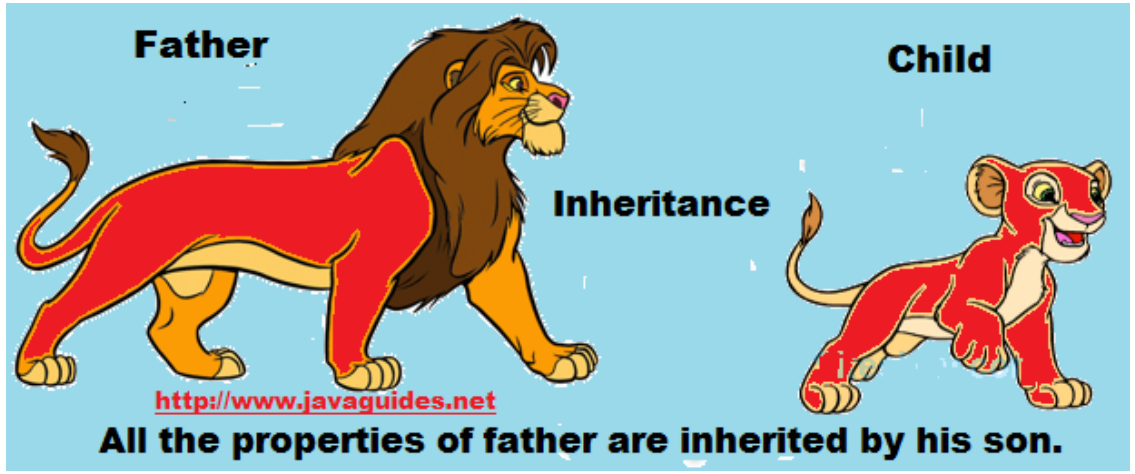
- Concept of inheritance. Derived class and based class.
- Derived class constructors, member function, inheritance in the English distance class, class hierarchies, inheritance and graphics shapes, public and private inheritance
- Aggregation: Classes within classes, inheritance, and program.

Introduction

- 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.

Introduction

- A class inherits the characteristics of another class.
- Builds class types from other class types.
- Increases reusability as no need to write the same piece of code again and again.

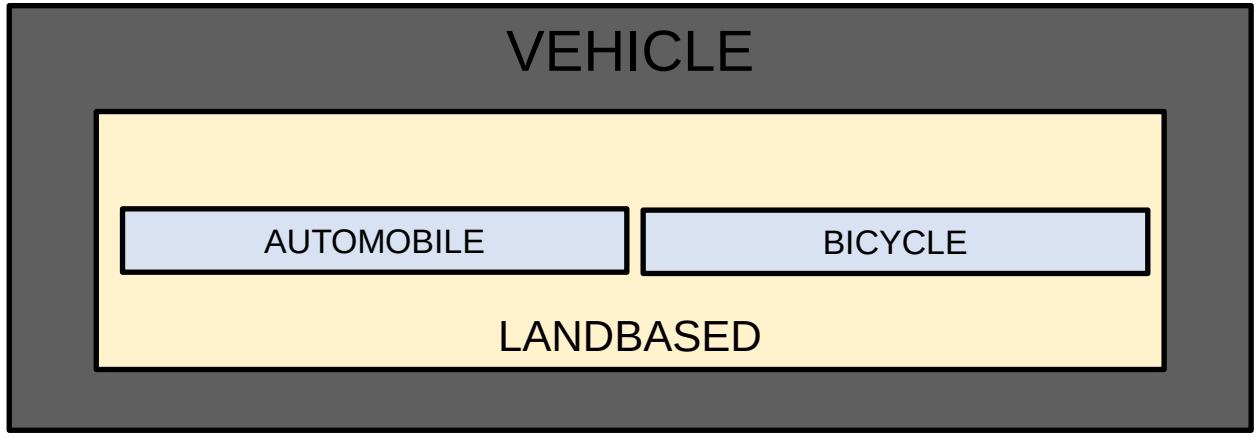


Inheritance

- Subgroupings with respect to a parent are called
  - Subclass
  - Derived Class
  - Children
- The derived class inherits from parent
- Characteristics
- Properties
- Capabilities

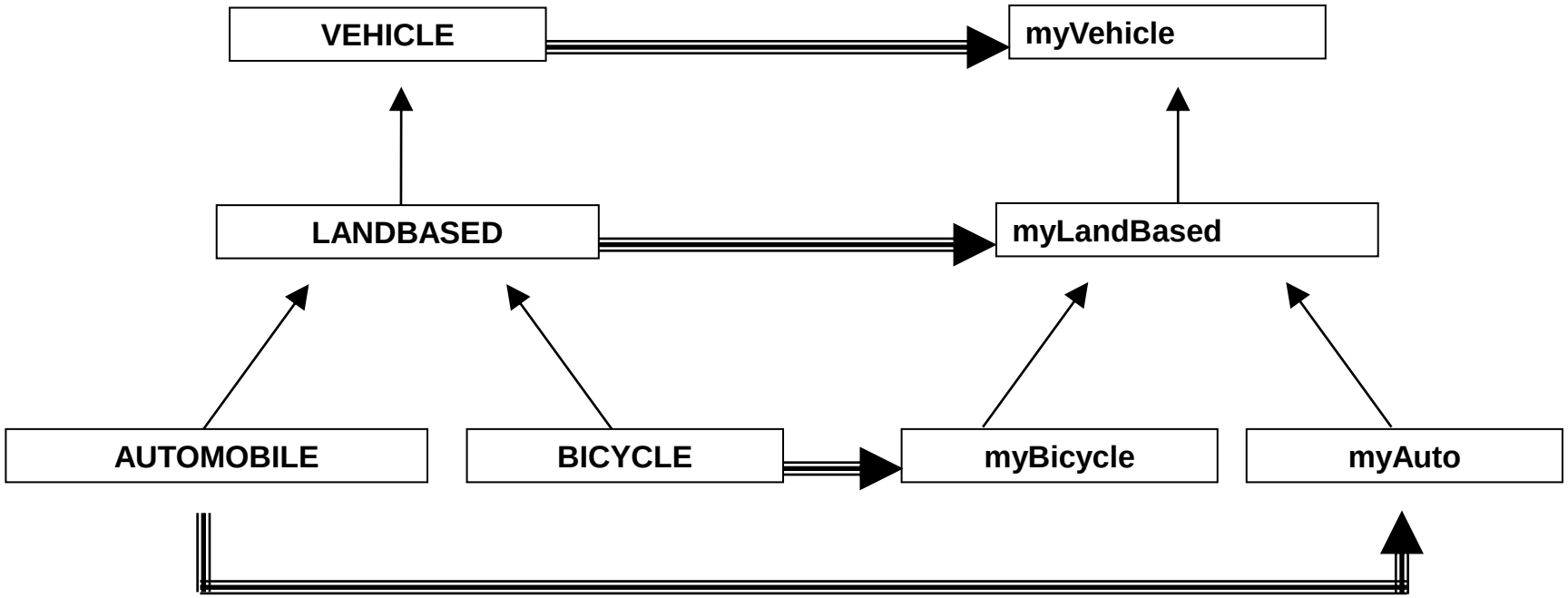
Inheritance

- Consider the vehicle hierarchy:



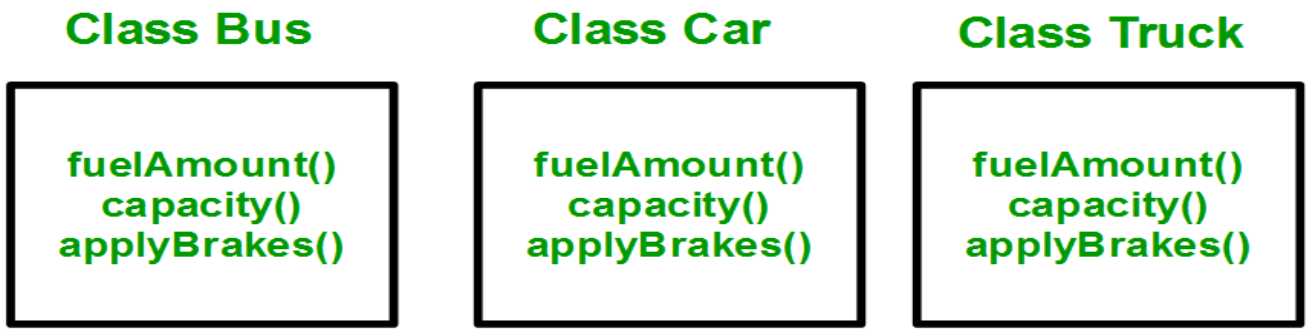
- Note that each entity is also a class

Class and Instance Hierarchy

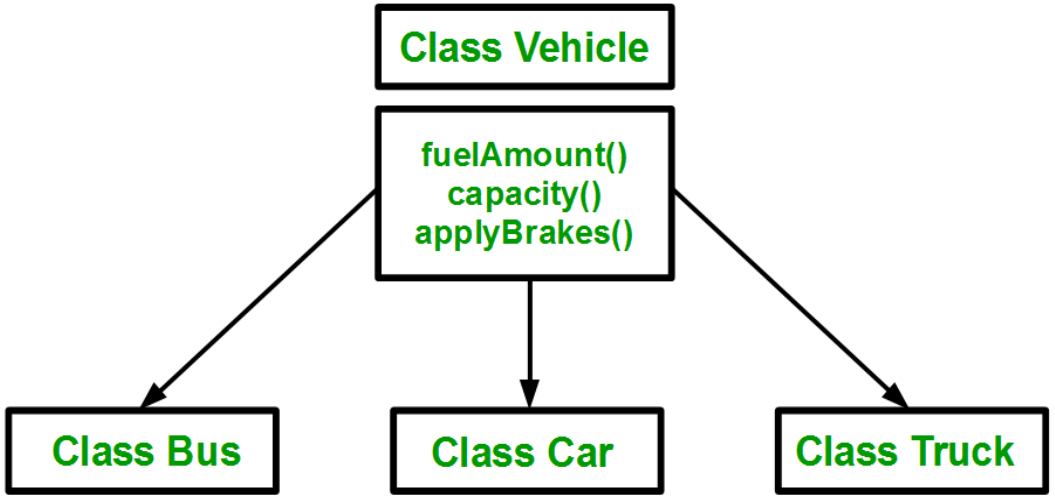


Example

- Consider a group of vehicles. You 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:



- You 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.



Implementing inheritance in C++:

- A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es).
- A class derivation list names one or more base classes and has the form −

- **class derived-class: access-specifier base-class**

- Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

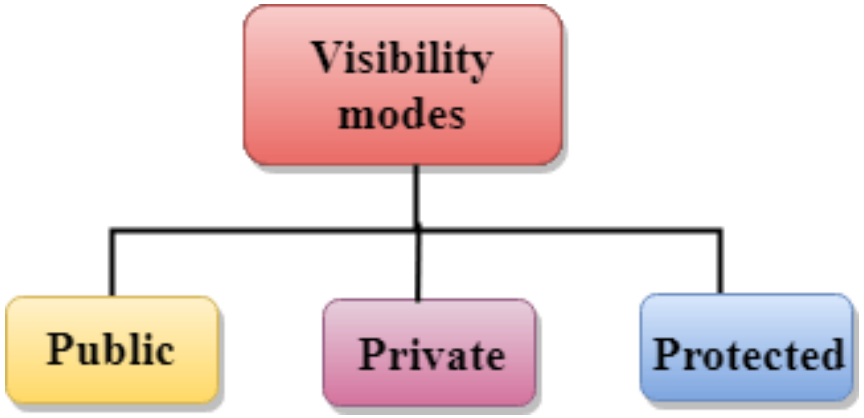
```cpp
#include <iostream>
using namespace std;
 class Shape {// Base class
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }
   protected:
      int width;
      int height;
};
class Rectangle: public Shape {// Derived class
   public:
      int getArea() {
         return (width * height);
      }
};
```

```cpp
int main(void) {
   Rectangle Rect;
   Rect.setWidth(5);
   Rect.setHeight(7);
   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;
   return 0;
}
```

Access Control and Inheritance

- A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class. A derived class inherits all base class methods with the following exceptions −
  - Constructors, destructors and copy constructors of the base class.
  - Overloaded operators of the base class.
  - The friend functions of the base class.

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no



Class Derivation

- Any class can serve as a base class…
  - Thus a derived class can also be a base class
  - Worth spending time at the outset of a design to develop sound definitions
  - syntax

    class DerivedClassName:specification BaseClassName

    DerivedClassName - the class being derived

    specification - specifies access to the base class members

    public

    protected

    private

    - private by default

Class Derivation

Class C be derived from base class A - PUBLIC

class C : public A

In class C

The inherited *public* members of A

Appear as *public* members of C

If myValue is a public data member of A

myValue can be accessed publicly through instances of C

```
C myC;
myC.myValue; // ok
```

Class Derivation

Class C be derived from base class B - PRIVATE

class C : private B

In class C

The inherited *public* members of B

Appear as *private* members of C

if myValue is a public data member of B

myValue cannot be accessed publicly and directly through instances of C

```
C myC;
myC.myValue; // compile error
```

Function members of C can still access public members of B as public

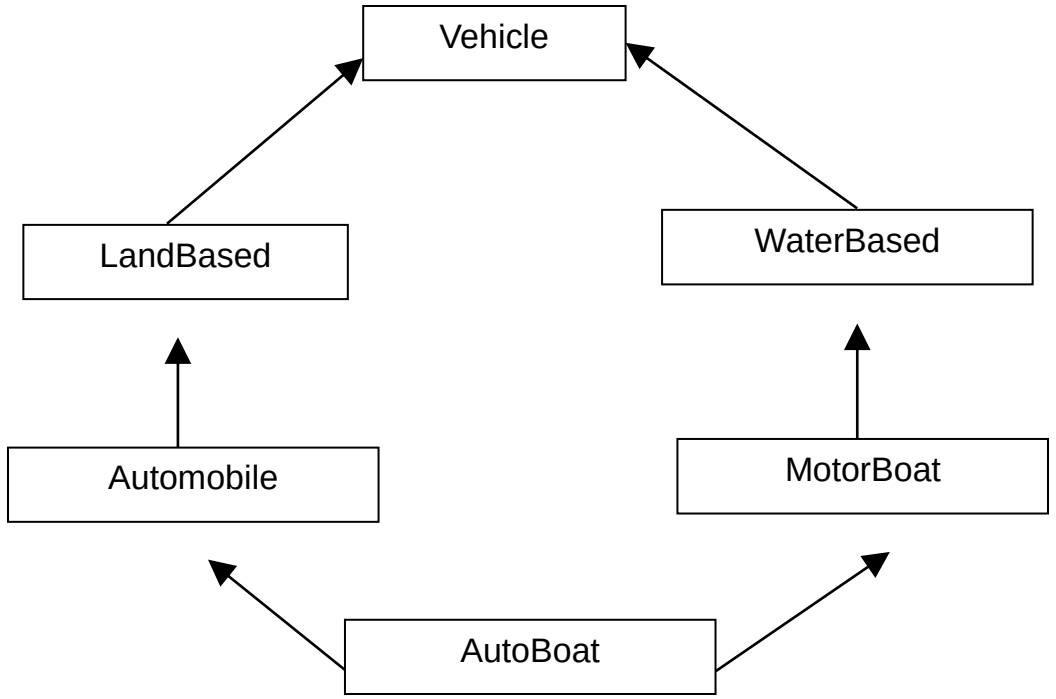
Type of Inheritance

- When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access-specifier as explained above.
- We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied −

- **Public Inheritance** − When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
- **Protected Inheritance** − When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.
- **Private Inheritance** − When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

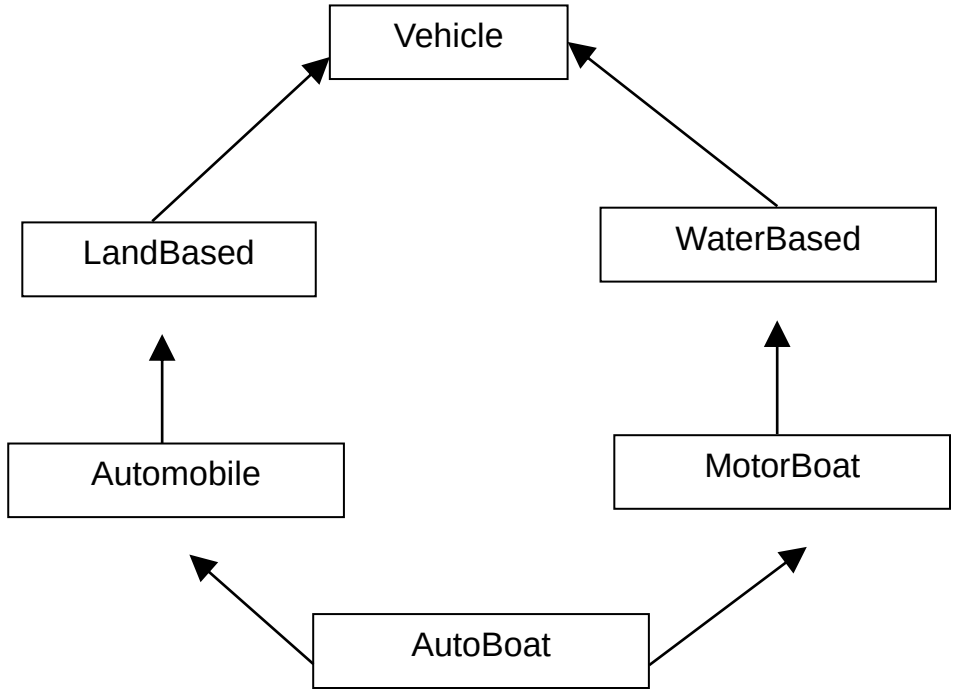
Single/Multiple Inheritance

- Single Inheritance
- Each class or instance object has a single parent
- Multiple Inheritance
  - Classes inherit from multiple base classes (might not have same ancestors as shown in the example below)
  - Defines a relationship:Between several (independent) class types

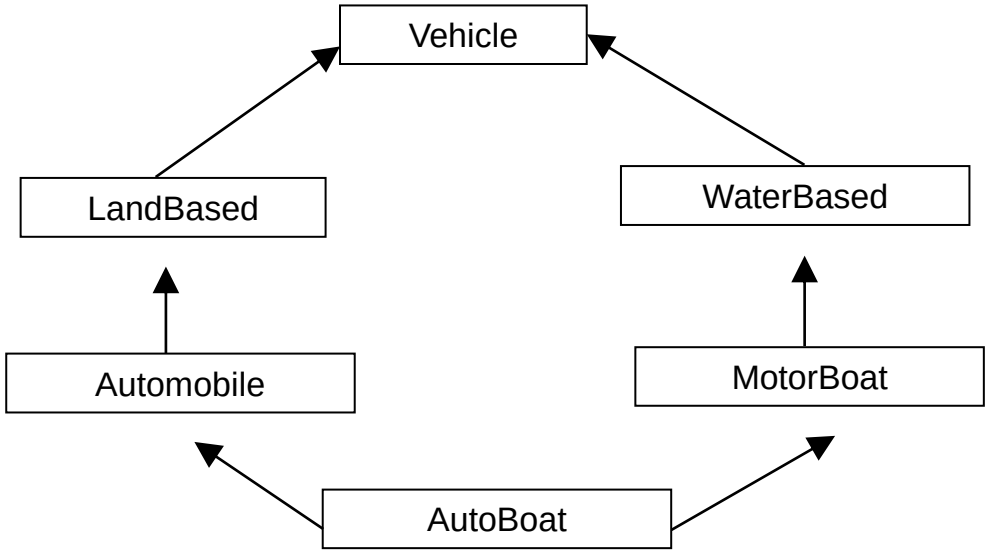


Single/Multiple Inheritance

- The derived class AutoBoat inherits attributes and properties from multiple classes, namely, Automobile, MotorBoat, …

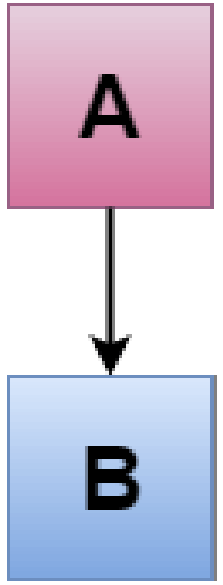


Single/Multiple Inheritance



Single Inheritance

- Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.
- Where 'A' is the base class, and 'B' is the derived class.



Employee is the base class and Programmer is the derived class.

```
#include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
   };
int main(void) {
     Programmer p1;
     cout<<"Salary: "<<p1.salary<<endl;
     cout<<"Bonus: "<<p1.bonus<<endl;
    return 0;
}
```

Class work

Create a constructor to class A and pass two parameters (l and b), create a member function to multiply them. Create another class B with data member display. Inherit class A to B and use display function to show the multiplication of two numbers passed in Class A

```
#include <iostream>
using namespace std;
class A  {
    int a = 4;
    int b = 5;
    public:
    int mul()     {
        int c = a*b;
        return c;
    }
};
  class B : private A  {
    public:
    void display()     {
        int result = mul();
        std::cout <<"Multiplication of a and b is : "<<result<< std::endl;
    }
};
int main()  {
    B b;
    b.display();
     return 0;
}
```

In the example, class A is privately inherited. Therefore, the mul() function of class 'A' cannot be accessed by the object of class B. It can only be accessed by the member function of class B.

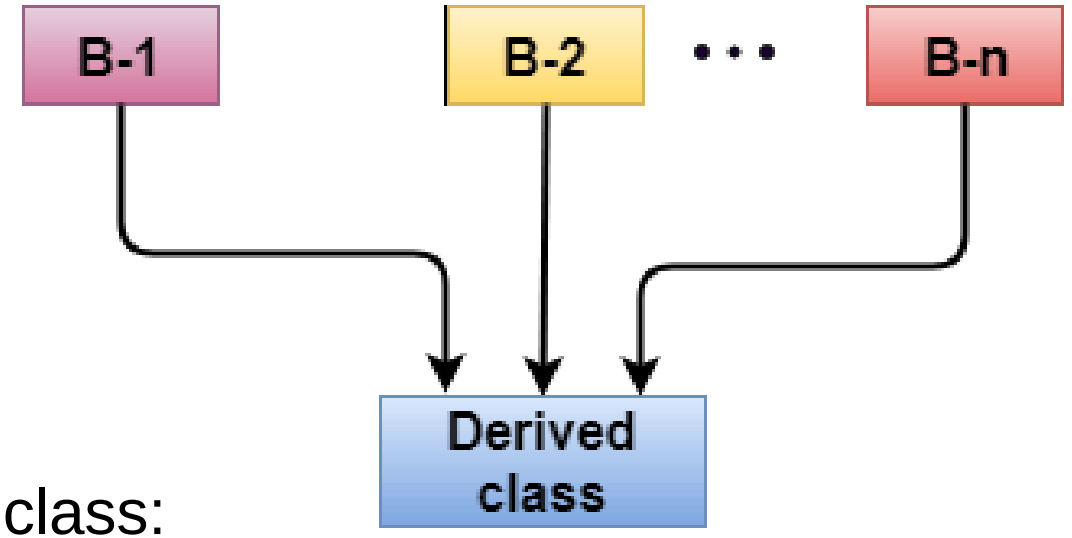
Multilevel Inheritance

- Multilevel inheritance is a process of deriving a class from another derived class.
- When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++.
- Inheritance is transitive so the last derived class acquires all the members of all its base classes.

```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
 }
  };
  class Dog: public Animal
  {
      public:
    void bark(){
   cout<<"Barking..."<<endl;
     }
  };
  class BabyDog: public Dog
  {
      public:
    void weep() {
   cout<<"Weeping...";
     }
  };
int main(void) {
    BabyDog d1;
    d1.eat();
    d1.bark();
     d1.weep();
     return 0;
}
```

C++ Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



Syntax of the Derived class:

```
class D : visibility B-1, visibility B-2, ?
{
    // Body of the class;
}
```

Multiple Inheritance - Example

```
#include<iostream>
using namespace std;
class base {
    int arr[10];
};

class b1: public base { };
class b2: public base { };
class derived: public b1, public b2 {};
int main(void)
{
  cout << sizeof(derived);
  return 0;
}
```

Class 'C' inherits two base classes 'A' and 'B' in a public mode.

```
#include <iostream>
using namespace std;
class A  {
   protected:
    int a;
   public:
   void get_a(int n)     {
      a = n;
   }
};
class B {
   protected:
   int b;
   public:
   void get_b(int n)     {
      b = n;
   }
};
class C : public A, public B
{
  public:
   void display()     {
      std::cout << "The value of a is : " <<a<< std::endl;
      std::cout << "The value of b is : " <<b<< std::endl;
      cout<<"Addition of a and b is : "<<a+b;
   }
};
int main()  {
  C c;
  c.get_a(10);
  c.get_b(20);
  c.display();

   return 0;
}
```

Practice question-1

- Create two classes named Mammals and MarineAnimals. Create another class named BlueWhale which inherits both the above classes. Now, create a function in each of these classes which prints "I am mammal", "I am a marine animal" and "I belong to both the categories: Mammals as well as Marine Animals" respectively. Now, create an object for each of the above class and try calling
  1 - function of Mammals by the object of Mammal
  2 - function of MarineAnimal by the object of MarineAnimal
  3 - function of BlueWhale by the object of BlueWhale
  4 - function of each of its parent by the object of BlueWhale

Practice question-2

- Make a class named Fruit with a data member to calculate the number of fruits in a basket. Create two other class named Apples and Mangoes to calculate the number of apples and mangoes in the basket. Print the number of fruits of each type and the total number of fruits in the basket.

Ambiguity Resolution in Inheritance

```
#include <iostream>
using namespace std;
class A {
   public:
   void display()     {
      std::cout << "Class A" << std::endl;
   }
};
class B  {
   public:
   void display()
   {
      std::cout << "Class B" << std::endl;
   }
};
class C : public A, public B  {
   void view()
   {
      display();
   }
};
```

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

```
int main()  {
   C c;
   c.display();
   return 0;
}
```

error: reference to 'display' is ambiguous display();

Ambiguity Resolution in Inheritance

- The issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

```
class C : public A, public B
{
    void view()
    {
        A :: display();         // Calling the display() function of class A.
        B :: display();         // Calling the display() function of class B.

    }
};
```

Ambiguity Resolution in single Inheritance

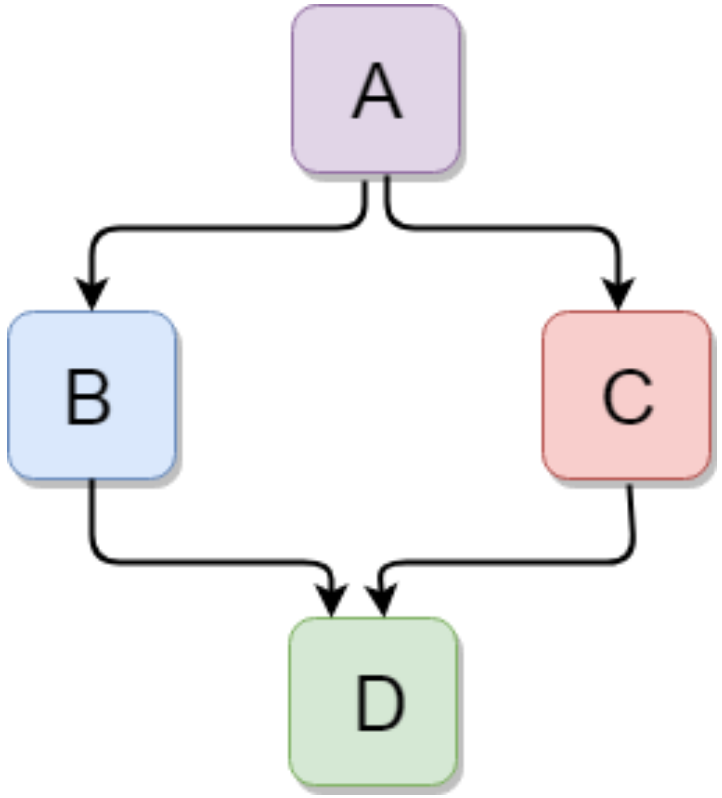
```
class A
{
  public:
void display()
{
  cout<<“Class A”;
}
} ;
class B: public A
{
 public:
 void display()
{
 cout<<“Class B”;
}
} ;
```

In this case, the function of the derived class overrides the method of the base class. Therefore, call to the display() function will simply call the function defined in the derived class. If we want to invoke the base class function, we can use the class resolution operator.

```
int main()
{
  B b;
  b.display();     // Calling the display() function of B class.
  b.B :: display();  // Calling the display() function defined in B class.
}
```

Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.



```
#include <iostream>
using namespace std;
class A
{
   protected:
   int a;
   public:
   void get_a()
   {
     std::cout << "Enter the value of 'a' : " << std::endl;
     cin>>a;
   }
};
class B : public A
{
   protected:
   int b;
   public:
   void get_b()
   {
      std::cout << "Enter the value of 'b' : " << std::endl;
     cin>>b;
   }
};
```

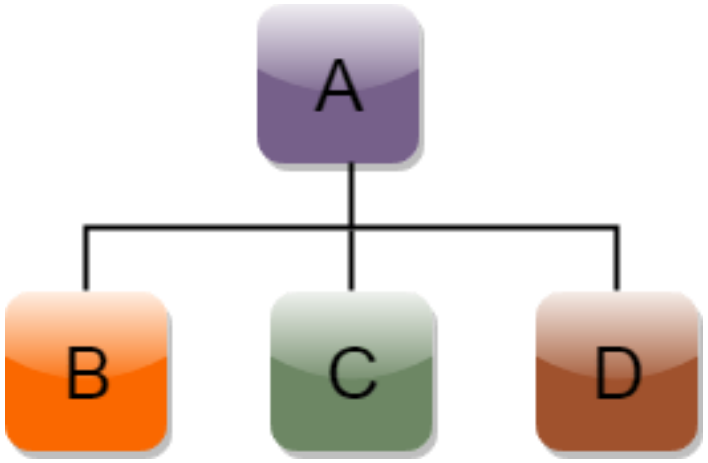
```
class C  {
   protected:
   int c;
   public:
   void get_c()
   {
      std::cout << "Enter the value of c is : " << std::endl;
      cin>>c;
   }
};
class D : public B, public C  {
   protected:
   int d;
   public:
   void mul()     {
       get_a();
       get_b();
       get_c();
       std::cout << "Multiplication of a,b,c is : " <<a*b*c<< std::endl;
   }
};
```

```
int main()  {
   D d;
   d.mul();
   return 0;
}
```

```
Enter the value of 'a' : 10
Enter the value of 'b' : 20
Enter the value of c is : 30
Multiplication of a,b,c is : 6000
```

Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



Syntax of Hierarchical inheritance:

```
class A
{
   // body of the class A.
}
class B : public A
{
   // body of class B.
}
class C : public A
{
   // body of class C.
}
class D : public A
{
   // body of class D.
}
```

```
#include <iostream>
using namespace std;
class Shape              // Declaration of base class.
{
   public:
   int a;
   int b;
   void get_data(int n,int m)
   {
      a= n;
      b = m;
   }
};
class Rectangle : public Shape  // inheriting Shape class
{
   public:
   int rect_area()
   {
      int result = a*b;
      return result;
   }
};
```

```
class Triangle : public Shape    // inheriting Shape class
{   public:
    int triangle_area()
    {
        float result = 0.5*a*b;
        return result;
    }
};
```

```
int main()
{   Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    int m = r.rect_area();
    std::cout << "Area of the rectangle is : " <<m<< std::endl;
    std::cout << "Enter the base and height of the triangle: " << std::endl;
    cin>>base>>height;
    t.get_data(base,height);
    float n = t.triangle_area();
    std::cout <<"Area of the triangle is : "  << n<<std::endl;
    return 0;
}
```

C++ Aggregation (HAS-A Relationship)

Aggregation is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents HAS-A relationship.

```
#include <iostream>
using namespace std;
class Address {
    public:
   string addressLine, city, state;
     Address(string addressLine, string city, string state)
    {
       this->addressLine = addressLine;
       this->city = city;
       this->state = state;
    }
};
```

Example of aggregation where Employee class has the reference of Address class as data member. In such way, it can reuse the members of Address class.

```
class Employee
   {
      private:
      Address* address;  //Employee HAS-A Address
      public:
      int id;
      string name;
      Employee(int id, string name, Address* address)
     {   this->id = id;
         this->name = name;
         this->address = address;
      }
     void display()
      {
         cout<<id <<" "<<name<< " "<<
           address->addressLine<< " "<< address->city<< " "<<address->state<<endl;
      }
   };
int main(void) {
    Address a1= Address("C-146, Sec-15","Noida","UP");
    Employee e1 = Employee(101,"Nakul",&a1);
    e1.display();
   return 0;
}
```