

# UNIT 3

## Inheritance

- In OOPs, the mechanism of deriving new class from old one is called inheritance.
- The old class is referred to as (base class /parent class/super class) and new class is called as (derived class/child class /sub class).
- The derived class inherits some or all of the data and functions from base class so that we can reuse the already written and tested code in our program.

### Example:

The properties of class named Person like name, age, address, phone number can be reusable in Class Employee. So, inheritance supports reusability and minimizes coding redundancy.

### Visibility Modifier (Access Specifier):

Visibility Modifier (Access Specifier)	Accessible from own class	Accessible from derived class	Accessible from objects outside class
private	Yes	No	No
protected	Yes	Yes	No
public	Yes	Yes	Yes

### Defining Derived Class (Specifying Derived Class):

#### General syntax :

```
class derived_class_name : visibility_mode base_class_name
{
//members of derived class
};
```

- The colon (:) indicates that the **derived\_class\_name** is derived from the **base\_class\_name**.
- The **visibility\_mode** is optional, if present, may be either private, protected or public. The default visibility mode is private.
- Visibility mode specifies whether the features of the base class are privately derived or publicly derived or derived on protected.

Base Class Visibility	Derived Class Visibility		
	private derivation	protected derivation	public derivation
Private	Not inherited	Not inherited	Not inherited
Protected	private	protected	protected
Public	private	protected	Public

```

class A
{
private:
    int x;
protected:
    int y;
public:
    int z;
};

class B : private A // 'private' is default for classes
{
    // x is not accessible from B
    // y is private
    // z is private
};

class C : protected A
{
    // x is not accessible form C
    // y is protected
    // z is protected
};

class D : public A
{
    // x is not accessible from D
    // y is protected
    // z is public
};

```

While any derived\_class is inherited from a base\_class, following things should be understood:

1. When a base class is privately inherited by a derived class, only the public and protected members of base class can be accessed by the member functions of derived class. This means no private member of the base class can be accessed by the objects of the derived class. Public and protected member of base class becomes private in derived class.
2. When a base class is publicly inherited by a derived class the private members are not inherited, the public and protected are inherited. The public members of base class becomes public in derived class whereas protected members of base class becomes protected in derived class.
3. When a base class is protectedly inherited by a derived class, then public members of base class becomes protected in derived class ; protected members of base class becomes protected in the derived class, the private members of the base class are not inherited to derived class but note that we can access private member through inherited member function of the base class.

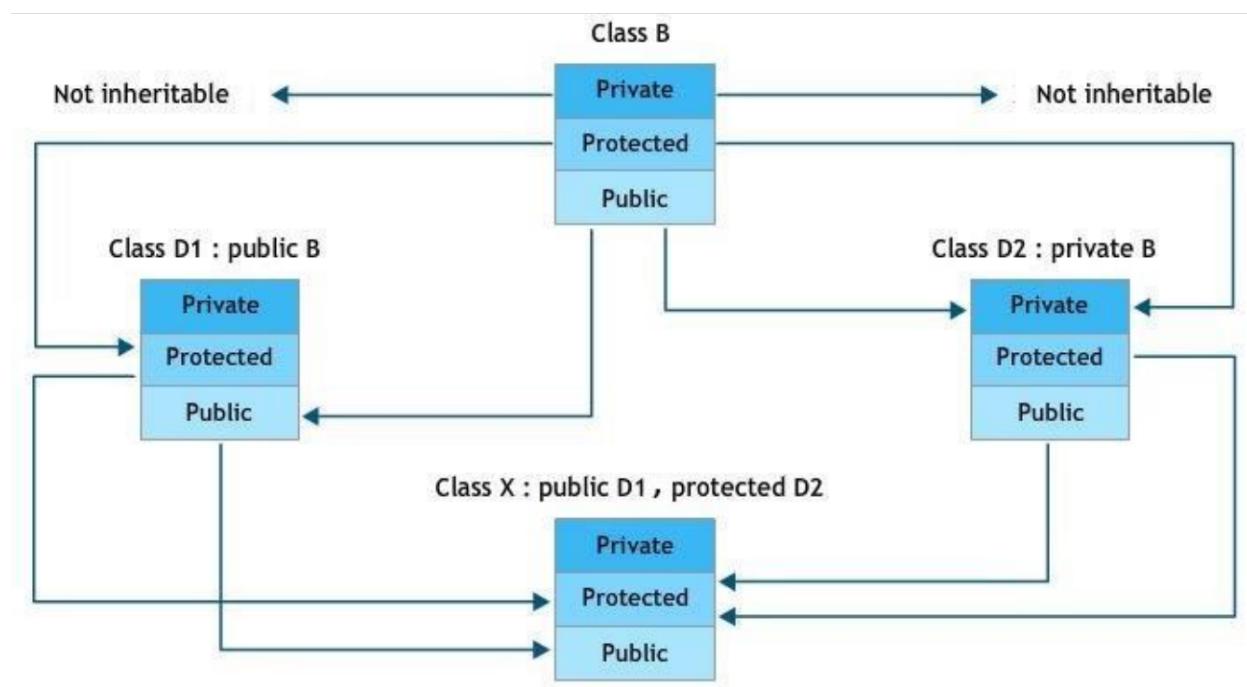


Fig: Effect of inheritance on visibility of members

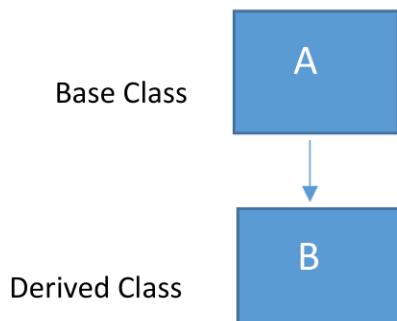
## Types of Inheritance:

There are five types of inheritance:

- 1) Single Inheritance
- 2) Multiple inheritance
- 3) Hierarchical Inheritance
- 4) Multilevel Inheritance
- 5) Hybrid Inheritance

### 1) Single Inheritance

When a class is derived from only one base class, then it is called single inheritance.



**General form:**

```
class A           // base class
{
    .....
};

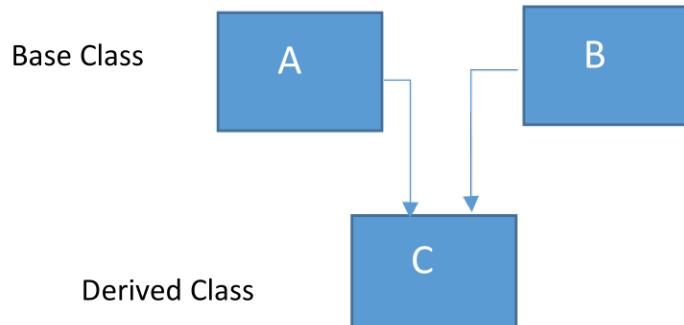
class B : public A // derived class B
{
    .....
};
```

**Here in, Single Inheritance**

- One base class and one derived class only involved
- Class A is base class.
- B is derived class.
- B is not used as base class again
- No further extension of derived class
- All data members in protected and public of class A can be accessed using object of class B.

## 2) Multiple Inheritance

When a class is derived from two or more base classes, it is called multiple inheritance.



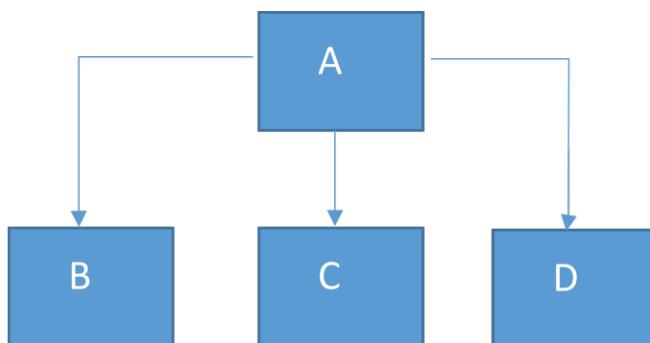
### General form

```
class A  
{  
};  
Class B  
{  
};  
class C:public A, public B  
{  
};
```

In General form there are 3 classes A, B and C. Class A and B are base classes and C is derived class.

All data members in protected and public of class A & B can be accessed using object of class C. Here two base classes A and B have been created and a derived class C is created from both base classes.

3) **Hierarchical Inheritance:** When two or more than two classes are derived from one base class, it is called hierarchical inheritance.



**General form**

```
class A  
{
```

```
};
```

```
Class B: public A  
{
```

```
};
```

```
Class C: public A  
{
```

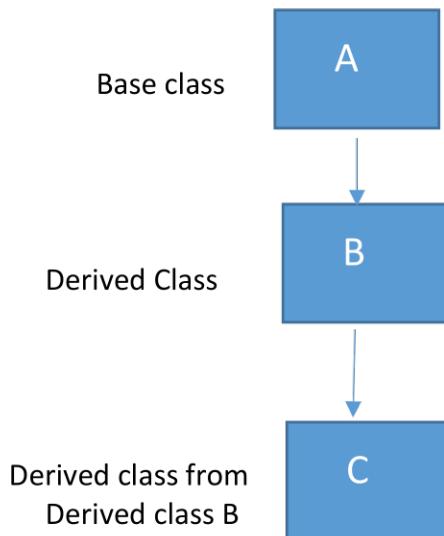
```
};
```

```
Class D: public A  
{
```

```
};
```

In General form three classes B and C and D are derived from same base class A. All data members in protected and public of class A can be accessed using object of classes B, C, D.

- 4) **Multilevel Inheritance:** The mechanism for deriving a class from another derived class is known as multilevel inheritance.



### **General form**

```
Class A
{
    .....
};

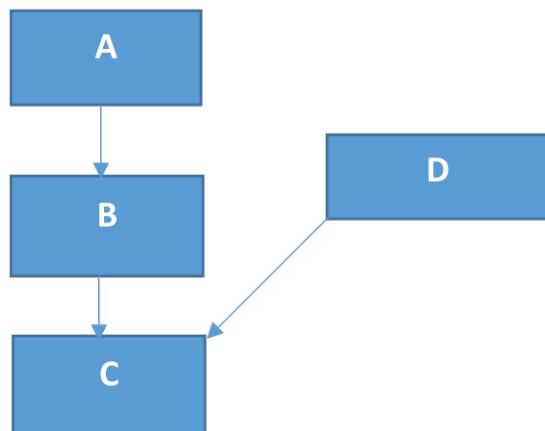
Class B: public A
{
    .....
};

Class C: public B
{
    .....
};
```

In General form there are 3 classes A,B and C. Class A is base class and Class B is derived class from Class A and class C is derived class from class B. All data members in protected and public of class A can be accessed using object of classes B, C and of class B can be accessed by class C.

#### **5) Hybrid Inheritance:**

Combination of two or more than two types of inheritance is known as hybrid inheritance.



**General form:**

```
Class A
{
};

Class D
{

};

Class B: public A
{
};

Class C: public B, Public D
{
};

};
```

**Description:** In General form there are 4 classes A, B, C and D. Classes A & D are base classes and Class B is derived class from Class A and class C is derived class from classes B & D. All data members in protected and public of class A can be accessed using object of classes B, C and of classes B&D can be accessed by class C.

**Program:**

**Single Inheritance: Public**

```
#include<iostream>
using namespace std;
class A
{
private:
int x;
protected:
int y;
public:
int z;
void get_xyz()
{
    cout<<"Enter the value of x,y,z"<<endl;
    cin>>x>>y>>z;
}
```

```

void show_xyz()
{
    cout<<"x="<<x<<"y="<<y<<"z="<<z<<endl;
}
};

class B:public A
{
private:
int k,sum;
public:
void get_k()
{
cout<<"Enter the value of k"<<endl;
cin>>k;
}
void show_k()
{
cout<<"k="<<k<<endl;
}
void addition()
{
    sum=y+z+k;
}

void display()
{
cout<<"Sum="<<sum<<endl;
}
};

int main()
{
B b1;
b1.get_xyz();
b1.get_k();
b1.show_xyz();
b1.show_k();
b1.addition();
b1.display();
}

```

## Single Inheritance: Private

```
#include<iostream>
using namespace std;
class A
{
private:
int x;
protected:
int y;
public:
int z;
void get_xyz()
{
    cout<<"Enter the value of x,y,z"<<endl;
    cin>>x>>y>>z;
}
void show_xyz()
{
    cout<<"x="<<x<<"y="<<y<<"z="<<z<<endl;
}
};

class B:private A
{
private:
int k,sum;
public:
void getdata()
{
get_xyz();
cout<<"Enter the value of k"<<endl;
cin>>k;
}
void showdata()
{
show_xyz();
cout<<"k="<<k<<endl;
}
void addition()
{
    sum=y+z+k;
}
void display()
{
cout<<"y+z+k="<<sum<<endl;
}
};
```

```

int main()
{
B b1;
b1.getdata();
b1.showdata();
b1.addition();
b1.display();
}

```

### **Example of Multiple Inheritance:**

```

#include<iostream>
using namespace std;
class M
{
protected:
    int m;
public:
    void get_m()
    {
        cout<<"Enter the value of m"<<endl;
        cin>>m;
    }
};

class N
{
protected:
int n;
public:
    void get_n()
    {
        cout<<"Enter the value of n"<<endl;
        cin>>n;
    }
};

class P:public M,public N
{
private:
    int product;
public:
    void calculate()
    {
        product=m*n;
    }
}

```

```

        void display()
        {
            cout<<"m*n="<<product<<endl;
        }
    };
int main()
{
P p;
p.get_m();
p.get_n();
p.calculate();
p.display();
return 0;
}

```

### **Example of multilevel Inheritance**

```

#include<iostream>
using namespace std;
class student
{
protected:
int roll;
public:
void getnum(int x)
{
    roll=x;
}

void putnum()
{
cout<<"Roll number="<<roll<<endl;
}
};

class test:public student
{
protected:
float sub1,sub2;
public:
void getmarks(float x,float y)
{
    sub1=x;
    sub2=y;
}

```

```

void putmarks()
{
    cout<<"Marks in sub1="<<sub1<<endl;
    cout<<"Marks in sub2="<<sub2<<endl;
}
};

class result:public test
{
    private:
        float total;
    public:
        void display()
        {
            total=sub1+sub2;
            putnum();
            putmarks();
            cout<<"Total="<<total<<endl;
        }
};

int main()
{
    result r;
    r.getnum(10);
    r.getmarks(75.5,80.2);
    r.display();
    return 0;
}

```

### **Example of Hierarchical Inheritance**

```

#include<iostream>
using namespace std;
class A
{
protected:
    int x,y;
public:
    void get_xy()
    {
        cout<<"Enter the value of x and y"<<endl;
        cin>>x>>y;
    }
};

```

```

class B:public A
{
private:
int sum ;
public:
void add( )
{
sum=x+y ;
cout<< "x+y=" <<sum<<endl;
}
};

class C: public A
{
private:
int diff;
public:
void sub()
{
diff=x-y;
cout<< "x-y=" <<diff<<endl;
}
};

class D: public A
{
private:
int product;
public:
void mul( )
{
product=x*y ;
cout<<"x*y=" <<product<<endl;
}
};

int main()
{
B b1;
C c1;
D d1;
b1.get_xy();
b1.add();
c1.get_xy();
c1.sub() ;
d1.get_xy();
d1.mul();
return 0;
}

```

## Example of Hybrid Inheritance

```
#include<iostream>
using namespace std;
class A
{
protected:
int x;
public:
void assignx( )
{
x=20;
}
};
class B: public A
{
protected:
int y ;
public:
void assigny()
{
y=40;
}
};
class C :public B
{
protected:
int z;
public:
void assignz( )
{
z=60;
}
};
class D
{
protected :
int k ;
public:
void assignk( )
{
k=80;
}
};
```

```

class E :public D, public C
{
private:
int total;
public:
void output()
{
total=x+y+z+k;
cout<< "x+y+z+k="<<total<<endl;
}
};

int main( )
{
E e1;
e1.assignx();
e1.assigny();
e1.assignz();
e1.assignk();
e1.output();
return 0;
}

```

## Ambiguity in Multiple inheritance

When two or more than two base classes have a function of identical name and when class inherits from multiple base classes then ambiguity occurs.

Let us consider the following case:

```

class M
{
public:
void display()
{
cout<<"Class M"<<endl;
}
};

class N
{
public:
void display()
{
cout<<"Class N"<<endl;
}
};

```

Which `display()` function is used by the derived class when we inherit these two classes ? We can solve this problem by redefining the member function in the derived class named `display()` using **resolution operator** with the function as shown below.

```
class P:public M,public N
{
public:
void display()
{
    M::display();
}
};
```

Now we can use the derived class as follows:

```
int main()
{
P p;
p.display();
return 0;
}
```

**Output:**

**Class M**

**How can you solve this problem due to the same function name that occurs in both base and derived classes? Explain with examples.**

### **Ambiguity in single Inheritance**

Ambiguity can occurs in single inheritance also. Suppose the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed.

This is known as function overriding in C++. The function in derived class overrides the function in base class.

#### **C++ program to demonstrate function overriding.**

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

class Base
{
public:
void display()
{
    cout << "Base Function" << endl;
}
};
```

```

class Derived : public Base
{
public:
    void display()
    {
        cout << "Derived Function" << endl;
    }
};

int main()
{
    Derived d1;
    d1.display();
    return 0;
}

```

### **Output**

Derived Function

- ✓ Here, the same function display() is defined in both Base and Derived classes.
- ✓ So, when we call display() from the Derived object d1, the display() from Derived is executed by overriding the function in Base.
- ✓ To access the overridden function of the base class, we use the scope resolution operator :: .

Now modifying the program we get output as follows.

```

int main()
{
    Derived d1;
    d1.display();
    d1.Base::display();
    return 0;
}

```

Here, this statement

**d1.Base::display();**

accesses the display() function of the Base class.

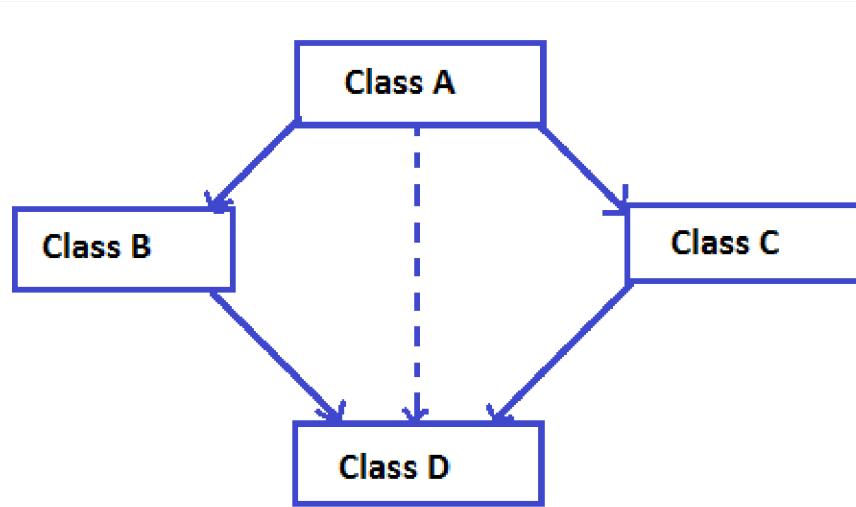
### **Output**

Derived Function  
Base Function

## Multipath inheritance and Virtual Base class

- During the time of hybrid inheritance when there is a hierarchical inheritance at the upper level and multiple inheritance at lower level, ambiguity occurs due to the duplication of data from multiple path at the grand child class. How this kind of ambiguity is resolved? Explain with suitable example.[PU:2013 fall]
- Does ambiguity occurs in hybrid inheritance .If yes? How can you remove this? Explain with example.[PU:2018 fall]
- Under what condition virtual base class is created? Explain it with suitable example .[PU:2017 fall]

Let us consider the following situation:



*figure: multipath inheritance*

In the above example, both Class B and Class C inherit properties of Class A, they both have single copy of Class A. However Class D inherit both Class B and Class C, therefore Class D have two copies of Class A, one from Class B and another from Class C. This introduces ambiguity and should be avoided..

The duplication of inherited members due to these multiple paths can be avoided by making common base class as virtual class.

By making a class virtual only one copy of that class is inherited though we may have many inheritance paths between the virtual base class and a derived class .we can specify a base class inheritance by using the keyword `virtual`.

In above situation to remove multiple copies of Class A in Class D, so we must inherit Class A in Class B and Class C as virtual class.

```
class A
{
-----
};

class B:virtual public A
{
//parent1
-----
};

class C:public virtual A
{
//parent2
-----
};

class D:public B, public C
{
//child
-----
};
```

## Program

```
#include<iostream>
using namespace std;
class A
{
public:
    int a;
};
class B : virtual public A
{
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 = 10;    //value of obj.a is replaced by 100 due to only one copy of data member of class A
    obj.a = 100;   // is available in class D
    obj.b = 20;
    obj.c = 30;
    obj.d = 40;
    cout<< "A=" << obj.a << endl;
    cout<< "B=" << obj.b << endl;
    cout<< "C=" << obj.c << endl;
    cout<< "D=" << obj.d << endl;
    return 0;
}
```

## Constructor and Destructor in Inheritance

- Compiler automatically calls constructor of base class and derived class when derived class object is created.
- If we declare derived class object in inheritance constructor of base class is executed first and then constructor of derived class.
- If derived class object goes out of scope the derived class destructor is executed first and then base class destructor

**Program to Illustrate when base and derived class constructor and destructor functions are executed.**

```
#include <iostream>
using namespace std;
class A
{
public:
A()
{
cout<< "Constructor in base class" << endl ;
}
~A()
{
cout<< "Destructor in base class" << endl ;
}
};

class B:public A
{
public:
B()
{
cout<< "Constructor in derived class" << endl ;
}
~B()
{
cout<< "Destructor in derived class" << endl ;
}
};

int main( )
{
    B obj;
    return 0;
}
```

**In Above Program** Class A is base class with one constructor and destructor, Class B is derived from class A having a constructor and destructor. In **main()** Object of derived class i.e. class B is declared. When class B's object is declared constructor of base class is executed followed by derived class constructor. At end of program destructor of derived class is executed first followed by base class destructor.

## **Output:**

```
Constructor in base Class  
Constructor in derived class  
Destructor in derived Class  
Destructor in base class
```

### **Constructor in derived class**

- If the **base class does not have any constructors taking arguments** (parameterized constructors), the **derived class need not have a constructor** function.
- If the **base class contains a constructor with one or more arguments** then it is **mandatory for the derived class to have a constructor** and pass the arguments to the base class constructors.
- When both the **derived and base classes contain constructors** then the **base class constructors is execute first** and then the constructor in the derived class is executed.
- In case of **Multiple Inheritance**
  - The **base classes are constructed in the order** in which they appear in the declaration of the derived class
- In case of **Multilevel Inheritance**
  - The constructors will be executed **in the order of inheritance**
- The constructors for virtual base classes are invoked before any non-virtual base classes.

Method of inheritance	Order of Execution
Class B:public A{ };	A(); base constructor (first) B(); derived constructor(second)
Class A:public B, public C{ }	B(); base constructor (first) C(); base constructor(second) A();derived (last)
Class A: public B, virtual public C{ }	C(); virtual base (first) B(); ordinary base constructor(second) A();derived (last)

### **Argument passing mechanism for supplying initial values to the bases classes constructors:**

- The **constructor of the derived class receives the entire list of values as its arguments and passed them to the base class constructors** in the order in which they are declared in the base class.
- The **base class constructors are called and executed first** before executing the statements in the body of the derived constructor.

***General form of defining a derived constructor:***

```
Derived-constructor(Arglist1, Arglist2,.....ArglistN, ArglistD): base1(Arglist1), base2(arglist2)....  
baseN(arglistN)  
{  
//Body of the derived constructor  
}
```

The header line of derived constructor function contains two parts separated by a colon (:)  
The first part provide the declaration of arguments that are passed to the derived constructor  
and the second part lists the function calls to the base constructors

Here, base1(Arglist1), base2(Arglist2)..... baseN (ArglistN) are function calls to base class  
constructors and Arglist1, Arglist2..... ArglistN are the actual parameters that are passed to the  
base constructors . Here, ArglistD provides the parameters that are necessary to initialize the  
members of the derived class itself.

**Example:**

```
#include<iostream>  
using namespace std;  
class alpha  
{  
int x;  
public:  
alpha(int a)  
{  
x=a;  
cout<<"Alpha is initialized"<<endl;  
}  
void showa()  
{  
cout<<"x="<<x<<endl;  
} };  
class beta  
{  
int y;  
public:  
beta(int b)  
{  
y=b;  
cout<<"Beta is initialized"<<endl;  
}
```

```

void showb()
{
cout<<"y="<<y<<endl;
}
};

class gamma:public beta,public alpha
{
int z;
public:
gamma(int a,int b,int c):alpha(a),beta(b)
{
    z=c;
    cout<<"Gamma is initialized"<<endl;
}

void showg()
{
cout<<"z="<<z<<endl;
}
};

int main()
{
gamma g(5,10,15);
g.showa();
g.showb();
g.showg();
return 0;
}

Output:
Beta is initialized
Alpha is initialized
Gamma is initialized

```

Here, **beta** is initialized first although it appears second in the derived constructor *as it has been declared first in the derived class header line*

```

class gamma: public beta, public alpha
{
```

```
}
```

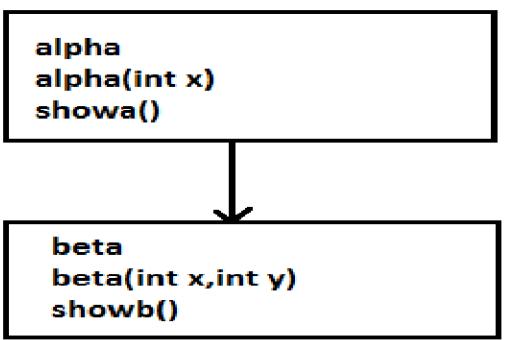
If we change the order *to*

```

class gamma: public alpha, public beta
{
```

}  
then alpha will be initialized first

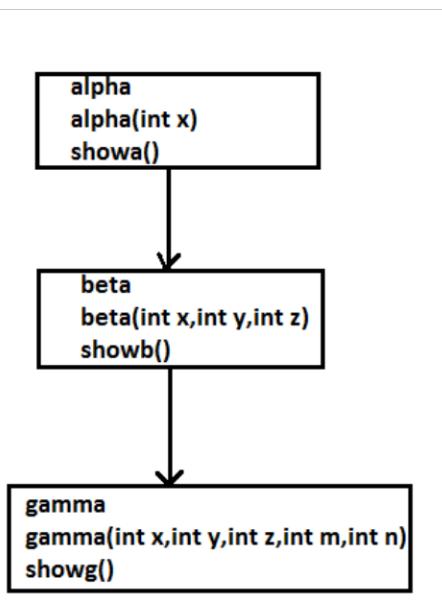
1. Write a complete program with reference to given below.



```
#include<iostream>
using namespace std;
class alpha
{
private:
int a;
public:
alpha(int x)
{
a=x;
}
void showa()
{
cout<<"value of a="<<a<<endl;
}
};
class beta:public alpha
{
private:
int b;
public:
beta(int x,int y):alpha(x)
{
b=y;
}
void showb()
{
cout<<"value of b="<<b<<endl;
}
};
```

```
int main()
{
beta b1(3,4);
b1.showa();
b1.showb();
return 0;
}
```

2. Write a complete program with reference to given below.



```
#include<iostream>
using namespace std;
class alpha
{
private:
int a;
public:
alpha(int x)
{
a=x;
}
void showa()
{
cout<<"value of a="<<a<<endl;
}
};
```

```

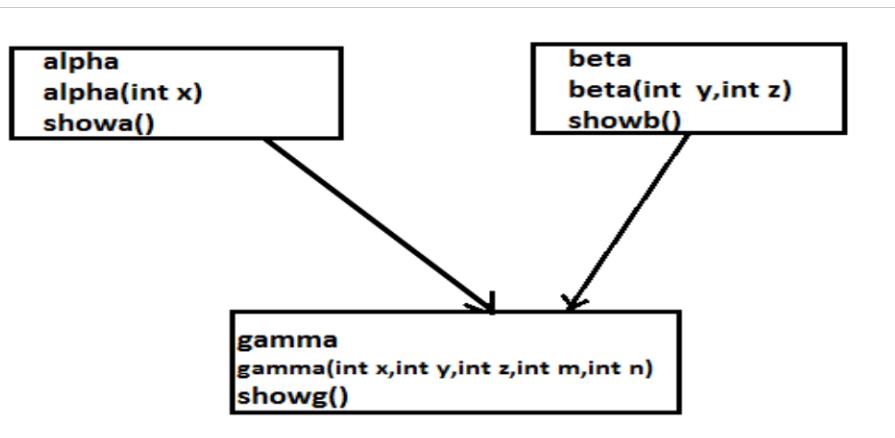
class beta:public alpha
{
private:
int b,c;
public:
beta(int x,int y,int z):alpha(x)
{
b=y;
c=z;
}
void showb()
{
cout<<"value of b="<<b<<endl;
cout<<"value of c="<<c<<endl;
}
};

class gamma:public beta
{
private:
int d,e;
public:
gamma(int x,int y,int z,int m,int n):beta(x,y,z)
{
d=m;
e=n    ;
}
void showg()
{
cout<<"value of d="<<d<<endl;
cout<<"value of e="<<e<<endl;
}
};

int main()
{
gamma g1(5,6,7,10,20);
g1.showa();
g1.showb();
g1.showg();
return 0;
}

```

3. Write a complete program with reference to given below.



```
#include<iostream>
using namespace std;
class alpha
{
private:
int a;
public:
alpha(int x)
{
a=x;
}
void showa()
{
cout<<"value of a="<<a<<endl;
}
};

class beta
{
private:
int b,c ;
public:
beta(int y,int z)
{
b=y;
c=z;
}

void showb()
{
cout<<"value of b="<<b<<endl;
cout<<"value of c="<<c<<endl;
}
};
```

```
class gamma:public alpha,public beta
{
private:
int d,e;
public:
gamma(int x,int y,int z,int m,int n):alpha(x),beta(y,z)
{
d=m;
e=n    ;
}

void showg()
{
cout<<"value of d="<<d<<endl;
cout<<"value of e="<<e<<endl;
}
};

int main()
{
gamma g1(5,6,7,10,20);
g1.showa();
g1.showb();
g1.showg();
return 0;
}
```

## **Subclass, Subtype and Principle of Substitutability**

If we consider the relationship of a data type associated with a parent class to the data type associated with a child class, the following observations can be made.

- Instances of the child class must possess all data members associated with the parent class.
- Instances of the child class must implement, through inheritance at least (if not explicitly overridden), all functionality defined for parent class. (*They can also define new functionality*).
- The instance of a child class can mimic the behavior of parent class and should be indistinguishable from an instance of parent class if substituted in similar situation.

### **The Principal of Substitutability**

It states that “If we have two classes A and B such that class B is a subclass of A, it should be possible to substitute instances of class B for instances of class A in any situation with no observable effect”

The term subtype is used to refer to a subclass relationship in which the principle of substitutability is maintained to distinguish such forms from the general subclass relationship, which may or may not satisfy this principle.

In the example below the class A object is replaced by the object of class B without any error. This is achieved when a child class is inherited from a base class publicly. In the same example, if the mode of inheritance is made private the base class object ‘a’ can’t be replaced by the derived class object ‘b’.

#### **EXAMPLE:**

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

class A
{
public:
void display()
{
cout << "class A";
}
};
```

```

class B : public A
{
public:
void display()
{
cout << "class B";
}
};

void test(A a)
{
a.display();
}

int main()
{
B b;
test(b);      // b substituted for object of A.
return 0;
}

```

## Composition and its implementation

- ✓ Composition is a relationship between two classes in which one class, known as the composite class, contains an object of another class, known as the component class, as a member variable.
- ✓ The composite class owns the component class, and the component class cannot exist independently of the composite class.
- ✓ In other words, the lifetime of the component class is controlled by the composite class.

```

class Engine
{

};

class Car
{
.....
Engine e;
.....
};

```

In this example, we have two classes: Engine and Car. The Car class is composed of an Engine object. The Engine object is an essential part of the Car object, and if the Car object is destroyed, the Engine object will also be destroyed.

### Program to illustrate the concept of composition

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

class Employee
{
int eid;
float salary;
public:
void getdata()
{
cout<<"Enter id and salary of employee"<<endl;
cin>>eid>>salary;
}
void display()
{
cout<<"Emp ID:"<<eid<<endl;
cout<<"Salary:"<<salary<<endl;
}
};

class Company
{
char cname[20];
char department[20];
Employee e;
public:
void getdata()
{
e.getdata();
cout<<"Enter company name and Department:"<<endl;
cin>>cname>>department;
}
void display()
{
e.display();
cout<<"Company name:"<<cname<<endl;
cout<<"Department:"<<department<<endl;
}
};
```

```

int main()
{
Company c;
c.getdata();
c.display();
return 0;
}

```

In above example class company contains the object of another class employee. As we know “company **has a** employee” sounds logical .so there exists a relationship between company and employee.

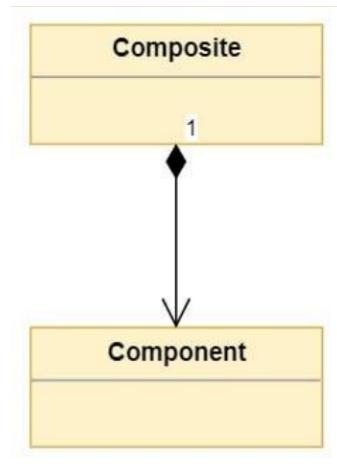
## Composition Relationship Diagram

Composition is a concept that models a has a relationship. It means that a class Composite can contain an object of another class Component. Has a relationship, here means that a Composite has a Component.

It is a "strong" form of aggregation with the following characteristics:

- ✓ it is binary association,
- ✓ it is a whole/part relationship,
- ✓ a part could be included in at most one composite (whole) at a time, and if a composite (whole) is deleted, all of its composite parts are "normally" deleted with it.

UML represents composition as follows:



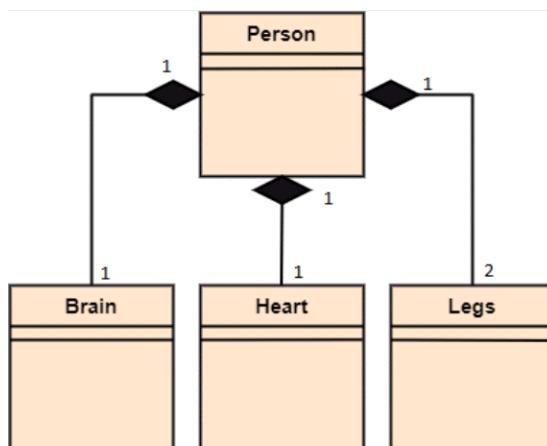
Composition is represented through a line with a diamond at the composite class pointing to the component class. The composite side can express the cardinality of the relationship. The cardinality indicates the number or valid range of Component instances the Composite class will contain.

In the diagram above, the 1 represents that the Composite class contains one object of type Component.

Cardinality can be expressed in the following ways:

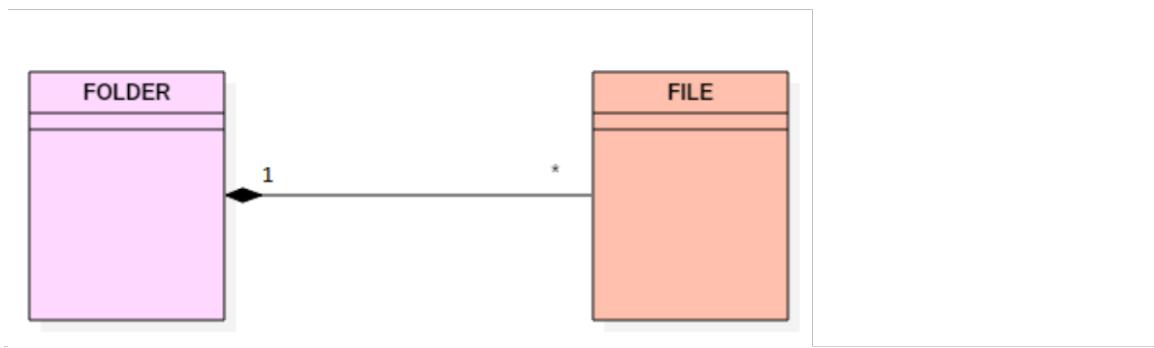
- ✓ A number indicates the number of Component instances that are contained in the Composite.
- ✓ The \* symbol indicates that the Composite class can contain a variable number of Component instances.
- ✓ A range 1..4 indicates that the Composite class can contain a range of Component instances. The range is indicated with the minimum and maximum number of instances, or minimum and many instances like in 1..\*.

### Example 1



- ✓ 1 Person contains 1 Brain.
- ✓ 1 person contains 1 heart.
- ✓ 1 person contains 2 legs.
- ✓ If the person is destroyed, the brain, heart, and legs will also get discarded.

### Example 2:



- ✓ Folder could contain many files, while each File has exactly one Folder parent.
- ✓ If Folder is deleted, all contained Files are deleted as well.

## IS A RULE/HAS A RULE ( IS A /HAS A RELATIONSHIP)

As we know, One of the advantages of an Object Oriented programming language is code reuse, however, There are two ways we can do code reuse either by the implementation of inheritance (IS-A relationship), or object composition (HAS-A relationship).

IS A Relationship	HAS A relationship
1. If the first concept is a specialized instance of the second one then there exists an “is-a” relationship between them.	1. If the second concept is a component of the first and the two are not in any sense the same thing then there exists a “has a” relationship between them.
2. Example: Car is a vehicle	2. Example: car has a engine.
3. “IS A” Relationship refers to Inheritance.	3. “HAS A” Relationship refers to composition.
4. ‘Is-a’ relationship asserts the instance of the subclass must be more specialized form of the superclass.	4. In ‘has-a’ relationship class contains object of another class as its data member.
5. class Vehicle {  }; class Car: Public Vehicle { };	5. class engine{ }; class car { engine e; // a car is composed of an engine };

## Inheritance VS Composition

Inheritance	Composition
<p>1. In Inheritance derived classes inherit attributes and methods from their parent class.</p> <p>2. It exhibits “is-a” relationship.</p> <p>3. Example: Car <b>is a</b> vehicle</p> <pre>4. class Vehicle { }; class Car: Public Vehicle { }; class Bus: Public Vehicle {    };</pre>	<p>1. In composition a class contains object of another class as its member data.</p> <p>2. It exhibits “has-a” relationship.</p> <p>3. Example: Car <b>has a</b> engine</p> <pre>4. class engine { }; class car { engine e; // a car is composed of an engine };</pre>
<p>5. Inheritance leads tight coupling between superclass and subclass. So it is harder to maintain in future.</p>	<p>5. In Composition relationship between class can be decoupled easily .so it is can be easily maintained in future.</p>
<p>6. Inheritance permits polymorphism</p>	<p>6. Composition does not permit polymorphism.</p>
<p><b>Program to illustrate concept of inheritance</b></p> <pre>#include&lt;iostream&gt; using namespace std; class first { private: int a; public: void geta() { cout&lt;&lt;"Enter value of a"&lt;&lt;endl; cin&gt;&gt;a; } void puta() { cout&lt;&lt;"a="&lt;&lt;a&lt;&lt;endl; } };</pre>	<p><b>Program to illustrate concept of composition</b></p> <pre>#include&lt;iostream&gt; using namespace std; class first { private: int a; public: void geta() { cout&lt;&lt;"Enter value of a"&lt;&lt;endl; cin&gt;&gt;a; } void puta() { cout&lt;&lt;"a="&lt;&lt;a&lt;&lt;endl; } };</pre>

<pre> class second: public first {     private:         int b;     public:         void getb()         {             cout&lt;&lt;"Enter value of b"&lt;&lt;endl;             cin&gt;&gt;b;         }         void putb()         {             cout&lt;&lt;"b="&lt;&lt;b&lt;&lt;endl;         } };  int main() {     second s;     s.getb();     s.putb();     return 0; } </pre>	<pre> class second {     private:         int b;         first f;     public:         void getb()         {             f.getb();             cout&lt;&lt;"Enter value of b"&lt;&lt;endl;             cin&gt;&gt;b;         }         void putb()         {             f.putb();             cout&lt;&lt;"b="&lt;&lt;b&lt;&lt;endl;         } };  int main() {     second s;     s.getb();     s.putb();     return 0; } </pre>
---	--

## Software reusability

Software reusability is the practice of using existing code for a new software. But in order to reuse code, that code needs to be high-quality. And that means it should be safe, secure, reliable, efficient and maintainable.

In OOP, the concept of inheritance provide the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

Reusability can be provided through the concept of composition also. In composition one class contains the object of another classes to make it as a whole.

### Advantages of reusability.

- Re usability enhanced reliability.
- Re usability saves the programmer time and effort.
- As the existing code is reused, it leads to less development and maintenance costs.
- Extensibility as we can extend the already made classes by adding some new features.
- Inheritance makes the sub classes follow a standard interface.

## **Difficulty in software reusability**

- As the number of projects and developers increases, it becomes harder to reuse software. It's a challenge to effectively communicate the details and requirements for code reuse.
- As the number of projects and developers grows, it's difficult to share libraries of reusable code.

## **Advantages and Disadvantages of Inheritance**

### **Advantages (Pros /Merits)**

- 1) Inheritance strongly supports reusability. The data members and member functions that are defined at parent class can be reused in base class. So, there is no need to define the member again.
- 2) Reusability enhanced reliability. The base class code will be already tested and debugged.
- 3) Base class logic can be extend as per business logic of the derived class.
- 4) Eliminates duplication of code.
- 5) Reduces development and maintenance costs as well saves time and efforts.
- 6) Program structure is short and concise which is more reliable.
- 7) Codes are easy to debug. Inheritance allows the program to capture the bugs easily
- 8) With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.
- 9) It is easy to partition the work in a project based on objects.
- 10) Object oriented systems can be easily upgraded from small to large system.
- 11) Software complexity can be easily managed.

### **Disadvantages (Cons/Demerits)**

- 1) Main disadvantage of using inheritance is that the two classes (base and inherited class) get tightly coupled. This means one cannot be used independent of each other.
- 2) A change in base class will affect all the child classes.
- 3) It increases the time and efforts take to jump through different levels of the inheritance.so it reduces execution speed.
- 4) Often, data members in the base class are left unused which may lead to memory wastage.
- 5) Inappropriate usage of inheritance will cause complexity in program
- 6) If the methods in the super class are deleted then it is very difficult to maintain the functionality of the child class which has implemented the super class's method.
- 7) Also with time, during maintenance adding new features both base as well as derived classes are required to be changed. If a method signature is changed then we will be affected in both cases (inheritance & composition)

- 1) WAP to concatenate two strings (name and address of a person) using the concept of containership. [2014 fall]**

```
#include<iostream>
#include<string.h>
using namespace std;
class first
{
char name[20];
public:
void setn(char n[])
{
strcpy(name,n);
}
char* getn()
{
return (name);
}
};

class second
{
char address[20];
public:
void seta(char a[])
{
strcpy(address,a);
}
char* geta()
{
return (address);
}
};

class concat
{
private:
first f;
second s;
public:
void getinfo(char n[],char a[])
{
f.setn(n);
s.seta(a);
}
```

```

void join()
{
strcat(f.getn(),s.getn());
cout<<"name+address:"<<f.getn();
}
};

int main()
{
concat c;
c.getinfo("Ram","kathmandu");
c.join();
return 0;
}

```

**Alternative solution:**

```

#include<iostream>
using namespace std;
class first
{
string name;
public:
void setn(string n)
{
name=n;
}
string getn()
{
return (name);
}
};

class second
{
string address;
public:
void seta(string a)
{
address=a;
}
string geta()
{
return (address);
}
};

```

```

class concat
{
private:
string tn,ta;
first f;
second s;
public:
void getinfo( string a,string b)
{
f.setn(a);
s.seta(b);
}
void join()
{
tn=f.getn();
ta=s.geta();
tn=tn+ta;
cout<<"name+address:<<tn;
}
};
int main()
{
concat c;
c.getinfo("Ram","Kathmandu");
c.join();
return 0;
}

```

- 2) Create a class person with data members name, age and address. Create another class teacher with data members Qualification and department .Also create another class Student with data members program and semester. Both class are inherited from class person. Every class has at least one constructor which uses base class constructor. Create member function showdata() in each to display the information of the class member.[PU:2018 fall,2014 spring]**

```

#include<iostream>
#include<string.h>
using namespace std;
class Person
{
private:
char name[20];
int age;
char address[20];

```

public:

```

Person(char n[],int a,char addr[])
{
    strcpy(name,n);
    age=a;
    strcpy(address,addr);
}
void showdata()
{
    cout<<"Name:"<<name<<endl;
    cout<<"Age:"<<age<<endl;
    cout<<"Address:"<<address<<endl;
}
};

class Teacher:public Person
{
char qualification[20];
char department[20] ;
public:
Teacher(char n[],int a,char addr[],char q[],char d[]):Person(n,a,addr)
{
strcpy(qualification,q);
strcpy(department,d);
}
void showdata()
{
cout<<"Qualification:"<<qualification<<endl;
cout<<"Department:"<<department<<endl;
}
};

class Student:public Person
{
private:
char program[20];
int sem;
public:
Student(char n[],int a,char addr[],char p[],int s):Person(n,a,addr)

{
strcpy(program,p);
sem=s;
}
}

```

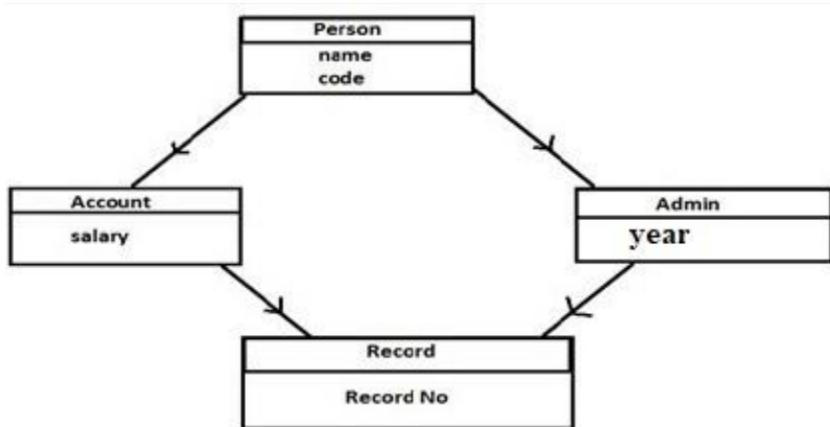
```

void showdata()
{
    cout<<"Program:"<<program<<endl;
    cout<<"Semester:"<<sem<<endl;
}
};

int main()
{
    Teacher t("Hari",32,"Kathmandu","Master","Civil");
    Student s("Ram",24,"Pokhara","Computer",2);
    cout<<"Information of Teacher is"<<endl;
    t.Person::showdata();
    t.showdata();
    cout<<"Information of Student is"<<endl;
    s.Person::showdata();
    s.showdata();
    return 0;
}

```

**3) Consider the class network of the following figure:**



The class **Record** derives the information from both **Account** and **Admin** classes and in turn derive information from the class **Person**. Define all the four classes with at least one parameterized constructor and 'void display' method in each class. In **main()** function create a object of class '**Record**' and initialize all data members and display them. [PU:2015 spring]

```

#include<iostream>
#include<string.h>
using namespace std;

```

```

class Person
{
private:
char name[20];
int code;
public:
Person(char n[],int c)
{
strcpy(name,n);
code=c;
}
void display()
{
cout<<"Name="<<name<<endl;
cout<<"Code="<<code<<endl;
}
};

class Account: virtual public Person
{
private:
float salary;
public:
Account(char n[],int c,float s):Person(n,c)
{
salary=s;
}
void display()
{
cout<<"salary="<<salary<<endl;
}
};

class Admin:virtual public Person
{
private:
int year;
public:
Admin(char n[],int c,int y):Person(n,c)
{
year=y;
}
void display()
{
cout<<"No of experience year="<<year<<endl;
}
};

```

```

class Record:public Account,public Admin
{
private:
int recno;
public:
Record(char n[],int c,float s,int y,int r):Account(n,c,s),Admin(n,c,y),Person(n,c)
{
recno=r;
}
void display()
{
cout<<"Record no ="<<recno<<endl;
}
};

int main()
{
char name[20];
int code,year,recno;
float salary;
cout<<"Enter person name and code "<<endl;
cin>>name>>code;
cout<<"Enter salary "<<endl;
cin>>salary;
cout<<"Enter number of year of experience "<<endl;
cin>>year;
cout<<"Enter Record no "<<endl;
cin>>recno;
Record r1(name,code,salary,year,recno);
r1.Person::display();
r1.Account::display();
r1.Admin::display();
r1.display();
return 0;
}

```

- 4) Write a base class that ask the user to enter time(hour,minute and second) and derived class adds the time of its own with the base.Finally make third class that is friend of derived and calculate the difference of base time and is own time.[PU:2017 fall]**

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

```

class time1
{
protected:
int hr1,min1,sec1;
public:
void getdata1()
{
cout<<"Enter hour,minute and second for base class"<<endl;
cin>>hr1>>min1>>sec1;
}
void display1()
{
cout<<hr1<<"hours"<<min1<<"minutes"<<sec1<<"seconds"<<endl;
}
};

class time2:public time1
{
private:
int hr2,min2,sec2;
int hr,min,sec;
public:
void getdata2()
{
cout<<"Enter hour,minute and second for derived class"<<endl;
cin>>hr2>>min2>>sec2;
}
void display2()
{
cout<<hr2<<"hours"<<min2<<"minutes"<<sec2<<"seconds"<<endl;
}
void addtime()
{
sec=sec1+sec2;
min=sec/60;
sec=sec%60;
min=min+min1+min2;
hr=min/60;
min=min%60;
hr=hr+hr1+hr2;
cout<<"sum of time:";
cout<<hr<<"hours"<<min<<"minutes"<<sec<<"seconds"<<endl;
}
friend class time3;
};

```

```

class time3
{
private:
int hr,min,sec;
int hr3,min3,sec3;

public:
void getdata3()
{
cout<<"Enter hour,minute and second for friend class"<<endl;
cin>>hr3>>min3>>sec3;
}
void timediff(time2 t)
{
if(sec3>t.sec1)
{
--t.min1;
t.sec1=t.sec1+ 60;
}
sec = t.sec1-sec3;
if(min3> t.min1)
{
--t.hr1;
t.min1=t.min1+ 60;
}
min = t.min1-min3;
hr = t.hr1-hr3;
if(hr<0)
{
hr=hr*(-1);
}
cout<<"Time Difference:";
cout<<hr<<"hours"<<min<<"minutes"<<sec<<"seconds"<<endl;
}
};

```

```

int main()
{
time2 t2;
time3 t3;
t2.getdata1();
t2.getdata2();
cout<<"Time in base class:";
t2.display1();
cout<<"Time in derived class:";
t2.display2();
t2.addtime();
t3.getdata3();
t3.timediff(t2);
return 0;
}

```

**5. Write a program to input two vector coordinates from the base class named “Base”. Class “Derived” inherits all the properties of class “Base”. Class “Derived” must contain a function named add\_vector() that add the two vectors input from the base class and finally display the result from the function display() that is friend to the base class.[PU:2019 spring]**

```

#include<iostream>
using namespace std;

class Base
{
protected:
    int x1,y1,z1,x2,y2,z2,x,y,z;
public:
    void getdata()
    {
        cout<<"Enter coordinate of Vector1:"<<endl;
        cin>>x1>>y1>>z1;
        cout<<"Enter coordinate of Vector2:"<<endl;
        cin>>x2>>y2>>z2;
    }
    friend void display (Base b) ;
};

```

```

class Derived: public Base
{
public:
void add_vector()
{
    x=x1+x2;
    y=y1+y2;
    z=z1+z2;
}

};

void display(Base b)
{
    cout<<"First vector (v1)="<<b.x1<<"i+"<<b.y1<<"j+"<<b.z1<<"k"<<endl;
    cout<<"Second vector (v2)="<<b.x2<<"i+"<<b.y2<<"j+"<<b.z2<<"k"<<endl;
    cout<<"Addition of vector(v1+v2)="<<b.x<<"i+"<<b.y<<"j+"<<b.z<<"k"<<endl;
}
int main()
{
    Derived d;
    d.getdata();
    d.add_vector();
    display(d); // Principle of substitutability is used here
    return 0;
}

```

**6)Create two classes Publication and Sales. The Publication class holds title and price and Sales class holds the total monthly sales. Derive a class called Book from both publication and sales. The book class stores author's name and page count. Each of these three classes should have function Display() to print the information of classes and should have at least one constructor that is inherited to derived class as well. Write a program to implement these classes.[PU:2022 fall]**

```

#include<iostream>
#include<string.h>
using namespace std;
class Publication
{
private:
char title[20];
float price;
public:
    Publication(char t[],float p)
    {
        strcpy(title,t);
        price=p;
    }
}

```

```

void display()
{
    cout<<"Title:"<<title<<endl;
    cout<<"Price:"<<price<<endl;
}
};

class Sales
{
private:
float total_monthly_sales;
public:
    Sales(float tms)
    {
        total_monthly_sales=tms;
    }
void display()
{
    cout<<"Total monthly sales:"<<total_monthly_sales<<endl;
}
};

class Book:public Publication,public Sales
{
private:
char author_name[20];
int pagecount;
public:
    Book(char t[],float p,float tms,char an[],int pcount):Publication(t,p),Sales(tms)
    {
        strcpy(author_name,an);
        pagecount=pcount;
    }
void display()
{
    cout<<"Author's name:"<<author_name<<endl;
    cout<<"Page count:"<<pagecount<<endl;
}
};

int main()
{
    Book b1("Physics",460.5,95000.5,"Ramesh",550);
    b1.Publication::display();
    b1.Sales::display();
    b1.Book::display();
    return 0;
}

```

## **Previous Board Exam Questions**

1. "Inheritance allows us to create a hierarchy of classes. Justify this statement. Discuss private and public inheritance.[PU:2016 spring]
2. How does visibility mode control the access of members in the derived class? Explain with an example.[PU 2017 spring]
3. Explain hybrid inheritance with example.[2009 spring]
4. What is hybrid Inheritance. Explain any three pros and three cons of inheritance. [PU: 2010 fall]
5. How inheritance support reusability features of OOP? Explain with example.[PU:2010 spring]
6. How does inheritance support reusability? What are the different forms of inheritance? Explain with example.[PU:2020 fall]
7. Why protected access specifier used in C++?Explain different types of inheritance with examples.[PU:2020 fall]
8. When base class and derived class have the same function name what happens when derived class object calls the function?[PU 2017 fall]
9. Explain how inheritance support Reusability? Describe the syntax of multiple and multilevel inheritance?[PU:2015 fall]
10. Inheritance supports characteristic of OOP. Justify your answer. Explain ambiguity that occurs in multiple inheritance.[PU:2017 spring]
11. "Ambiguity is essential evil" ",Explain by example how it can effectively solve in complex programming?[PU: 2015 spring]
12. Explain why multiple inheritance is dangerous?
13. During the time of hybrid inheritance when there is hierarchical inheritance at the upper level and multiple inheritance at lower level, ambiguity occurs due to the duplication of data from multiple path at the grand child class. How this kind of ambiguity is resolved? Explain with suitable example?
14. Does ambiguity occurs in hybrid inheritance? If yes, how can you remove this? Explain with an example.[PU 2018 fall]
15. Under what condition virtual base class is created? Explain with suitable example.[PU:2017 fall,2019 fall,2014 fall]
16. How are arguments are sent to base constructors in multiple inheritance ?Who is responsibility of it.[PU:2013 spring]
17. How does inheritance influence working of constructors and destructors? Class 'Y' has been derived from class 'X' .The class 'Y' does not contain any data members of its own. Does the class 'Y' require constructors? If yes why.[PU:2013 spring]
18. What is containership? How does it differ from inheritance, describe how an object of a class that contain object of another classes are created.[PU:2013 fall]
19. How composition differs from inheritance?
20. Explain how composition provide reusability?[PU:2018 fall]
21. Compare and contrast composition and inheritance?[PU:2015 fall]
22. Distinguish between subclass and subtype in light of principle of substitutability. Support your answer with suitable example.[PU:2006 spring,2016 spring]

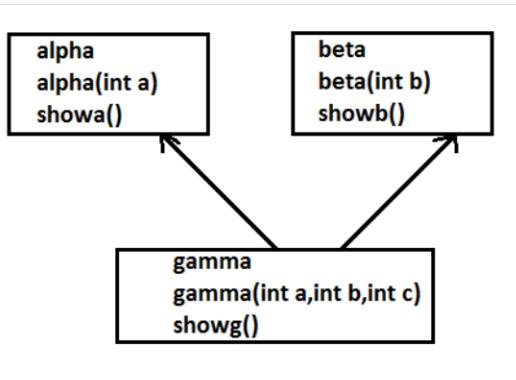
23. Differentiate between
- subclass and subtype.
  - Is a rule and has a rule
24. State principle of substitutability .Explain sub-classing for specialization, generalization.  
List out disadvantages of inheritance.[PU:2016 fall]
25. Differentiate between is a rule and has a rule with suitable example.  
[PU:2015 fall,2014 spring]

**26. Write a short notes on:**

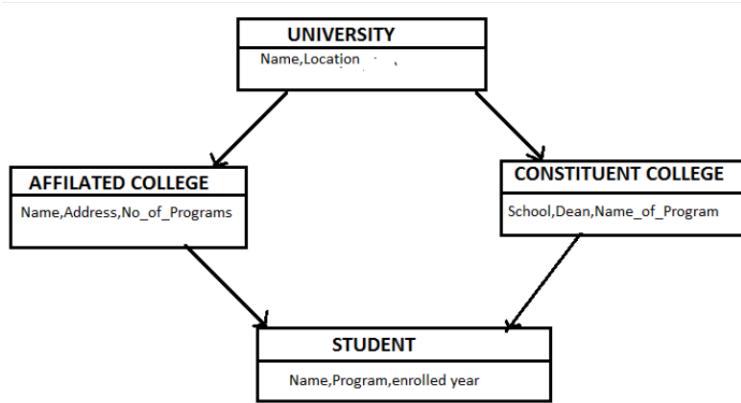
- Containership[PU:2010 fall]
- Subclass-subtype
- Software reusability[PU:2005 fall]
- Is a rule and has a rule[PU:2009 fall,2016 fall, 2016 spring,2015 spring]
- Hybrid inheritance[PU:2006 spring]ii
- Inheritance and substitutability
- Generalization[PU:2013 spring]

**Programming Questions:**

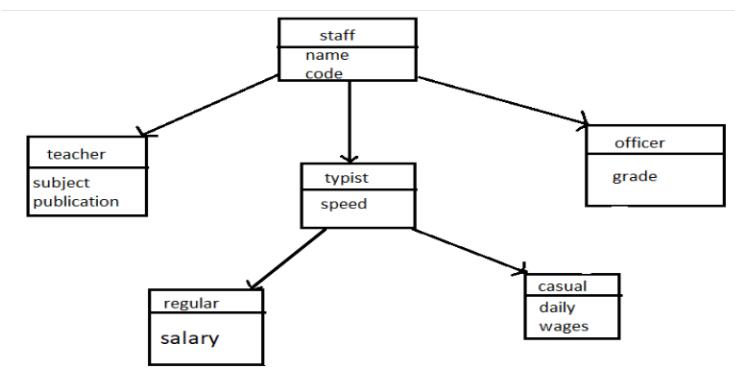
- 1) WAP to enter information of n students and then display is using the concept multiple inheritance,[PU: 2015 fall]
- 2) Create a class student with two data members represent name and age. Use appropriate member function to read and print these data members name and roll. Derive a class marks from student that has additional data member sessional1,sessional2 to store sessional marks. Derive another class result from marks and add the sessional marks. Use appropriate member function to read and display data in the class.
- 3) Create a class student with two data members to represent name and age. Use member function to read and print those data. From this derive a class called boarder with member data to represent room number. Derive another class called day-scholar from class student with member data to represent address of student. In both derived class use function to read and print respective data.
- 4) Write a complete program with reference to the given figure.



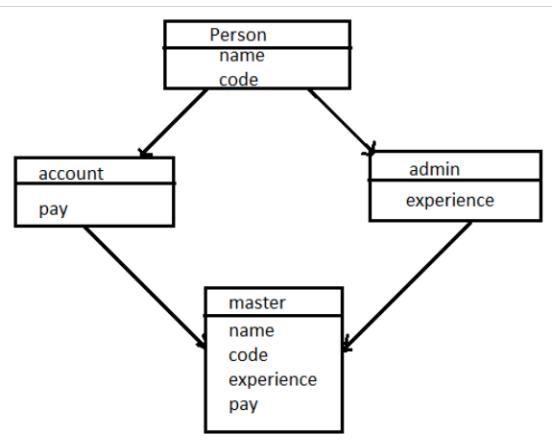
- 5) The following figure shows the minimum information required for each class. Write a program by realizing the necessary member functions to read and display information of individual object. Every class should contain at least one constructor and should be inherited from other classes as well.[PU:2019 fall]



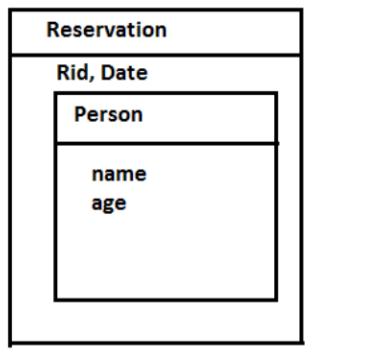
- 6) Create a class called **Person** with suitable data members to represent their name and age. Use member function to initialize and display these information. Derive **Student** and **Employee** from the **Person** class with their unique features. Initialize object of these class using constructor and display the information.[PU:2019 spring]
- 7) An educational institution wishes to maintain a database of its employees. The database is divided into a number of classes whose hierarchical relationship are shown below. The figure also shows minimum information requires for each class. Specify all the classes and define functions to create database and retrieve individual information when required.



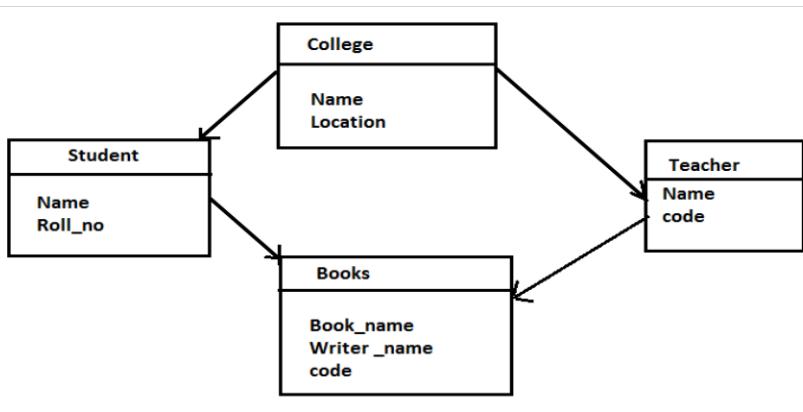
- 8) The following figure shows minimum information required for each class.
- Write a Program to realize the above program with necessary member functions to create the database and retrieve individual information



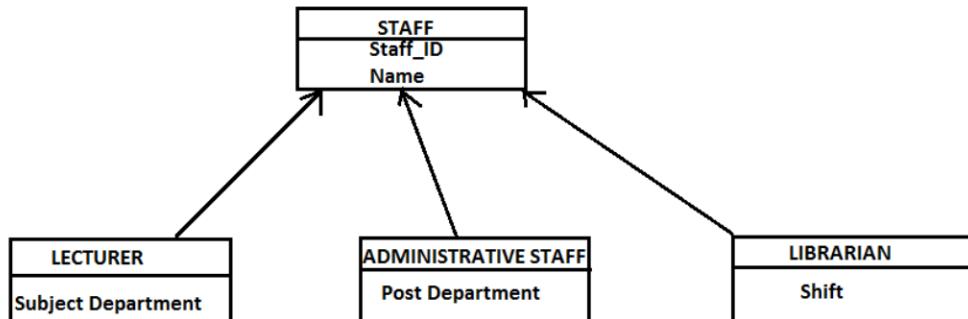
- ii) Rewrite the above program using constructor on each class to initialize the data members.
- 9) Write a program that allow you to book a ticket for person and use two classes Person, Reservation. Class Reservation is composite class/ container class.



- 10) The following figure shows the minimum information required for each class. Write a program to realize the above program with necessary member functions to create the database and retrieve individual information .Every class should contain at least one constructor and should be inherited to other classes as well.[PU:2010 spring][PU 2009 fall]



**11)** Develop a complete program for an institution, which wishes to maintain a database of its staff. The database is divided into number of classes whose hierarchical relationship is shown in the following diagram. specify all classes and define constructors and functions to create database and retrieve the individual information as per requirements.



**12)** Develop a complete program for an institution which wishes to maintain a database of its staff. Declare a base class **STAFF** which include **staff\_id** and **name**.Now develop a records for the following staffs with the given information below.

Lecturer(subject,department)

Administrative staff (Post,department)