Inheritance

Inheritance is a process in which one class acquires all the properties and behaviors of its parent object automatically.

In such way, you can reuse, extend or modify attributes and behaviors which are defined in other class.

The class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

The *syntax* of derived class:

Class derived_class_name::visibility-mode base_class_name{

//body of the derived class.

}

Visibility mode : It specifies whether the features of base class are publicly inherited or privately inherited. It can be public or private.

Public, Protected and Private inheritance:

Public inheritance makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.

Protected inheritance makes the public and protected members of the base class protected in the derived class.

Private inheritance makes the public and protected members of the base class private in derived class.

	Derived class visibility		
Base class visibility	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not inherited	Not inherited	Not inherited

Types of inheritance:

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

1) **Single inheritance:** It is defined as the inheritance in which a derived classis inherited from the only one base class.

Example 01:

```
Class Base{
Public: float salary = 80000;
};
Class derived : public Base{
Public: float bonus = 2000;
};
Int main(){
Derived p1;
```

```
Cout<<”Salary:”<<p1.salary<<endl;
Cout<<”Bonus”<<p1.bonus<<end;
Return 0;
}
```

Output:

Salary: 80000

Bonus: 2000

Example 2:

```
Class Base{
Public: int a, int b;
Int mul(){
Return a*b;
  }
   };
 Class derived : private Base{
    public: int a, int b;
       void f(){
           return mul(a,b);
}
};
Int main(void){
Derived b;
Cout<<b.f(2,3)<<endl;// we can’t do b.mul()
```

Return 0;

}

Output: 6

2) **Multilevel inheritance:** It is a process of inheriting a class from another derived class. Inheritance is transitive so the last derived class acquires all the members of all it's base classes.

Eg:

```
Class Base{
Public: float salary = 50000;
};
Class derived1: public Base{
};
Class derived2:public derived1{
Public: float bonus = 2000;
};
Int main(void){
Derived2 p1;
Cout<<"Salary"<<p1.salary<<endl;
Cout<<"Bonus"<<p1.bonus<<endl;
Return 0;
}
```

Output: Salary: 5000

Bonus: 2000

3) **Multiple Inheritance:** Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

Syntax: class D:visibility B1, visibility B-2,….{
//Body of the class;
}

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

```
Class A{
Public: void fun(){
Cout<<”Class A”<<endl;
  }
};
Class B{
Public: void func(){
Cout<<”Class B”<<endl;
   }
};
Class C:public A,public B{
Public: void fun2(){
Fun();
  }
};
```

Error: reference to ‘fun’ is ambiguous fun();

To resolve:

```
Class C: public A,public B{
Public: void fun(){
A::fun();
B::fun();
```

```
  }
};
```

Even in the main function we can’t write obj.fun() because there are 3 fun() in class C. To resolve this we need to use scope resolution operator, A::obj.fun().

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