**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
6. **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;

return 0;

}

**Output:** 6

**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

**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.

A

C

B

D

**HIERARCHIAL INHERITANCE**

Inheritance is a feature of Object-Oriented-programming in which a derived class (child class) inherits the property (data member and member functions) of the Base class (parent class). For example, a child inherits the traits of their parents.

In Hierarchical inheritance, more than one sub-class inherits the property of a single base class. There is one base class and multiple derived classes. Several other classes inherit the derived classes as well. Hierarchical structures thus form a tree-like structure. It is similar to that, mango and apple both are fruits; both inherit the property of fruit. Fruit will be the Base class, and mango and apple are sub-classes

EXAMPLE:

|  |
| --- |
| // C++ program for Hierarchical Inheritance  #include<iostream>  using namespace std;    class A   //superclass A  {    public:    void show\_A() {      cout<<"class A"<<endl;    }  };  class B : public A   //subclass B  {    public:    void show\_B() {      cout<<"class B"<<endl;    }  };    class C : public A   //subclass C  {    public:    void show\_C() {      cout<<"class C"<<endl;    }  };    int main() {    B b;  // b is object of class B    cout<<"calling from B: "<<endl;    b.show\_B();    b.show\_A();      C c;  // c is object of class C    cout<<"calling from C: "<<endl;    c.show\_C();    c.show\_A();    return 0;  } |

OUTPUT:

calling from B:

class B

class A

calling from C:

class C

class A