Experiment 08

OBJECTIVE

Implement different types of inheritance, function overriding and virtual Function.

PROGRAM

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.

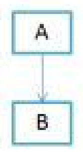
Syntax:

```
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.

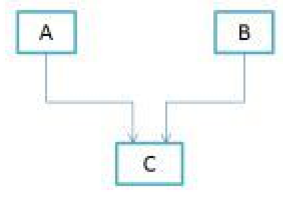
Note: A derived class doesn't inherit access to private data members. However, it does inherit a full parent object, which contains any private members which that class declares.

1) Single Inheritance: When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.



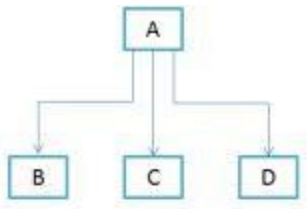
(a) Single Inheritance

2) Multiple Inheritance: "**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



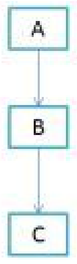
(b) Multiple Inheritance

3) Hierarchical Inheritance: In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.



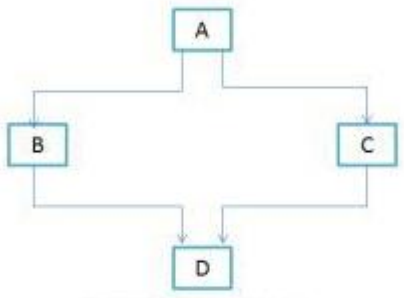
(c) Hierarchical Inheritance

4) Multilevel Inheritance: **Multilevel inheritance** refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



(d) Multilevel Inheritance

5) Hybrid Inheritance: In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple** inheritances. A typical flow diagram would look like.



(e) Hybrid Inheritance

Program:

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

//Base class
class Parent
{
   public:
     int id_p;
};

// Sub class inheriting from Base Class (Parent)
class Child : public Parent
{
   public:
     int id_c;
};

//main function
int main()
  {

       Child obj1;

       // an object of class child has all data members
       // and member functions of class parent
       obj1.id_c = 7;
       obj1.id_p = 91;
       cout << "Child id is " <<  obj1.id_c << endl;
       cout << "Parent id is " <<  obj1.id_p << endl;

       return 0;
  }
```

Output:

```
Child id is 7

Parent id is 91
```

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’.
 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.

FUNCTION OVERRIDING: As we know, inheritance is a feature of OOP that allows us to create derived classes from a base class. The derived classes inherit features of the base class. 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.

Program:

```
#include <iostream>

using namespace std;

class Base {

  public:
    void print() {

        cout << "Base Function" << endl;

    }

};

class Derived : public Base {

   public:

    void print() {

        cout << "Derived Function" << endl;

    }

};

int main() {

    Derived derived1;

    derived1.print();

    return 0;

}
```

Output: Derived Function

Explanation: Here, the same function print () is defined in both Base and Derived classes.

So, when we call print () from the Derived object derived1, the print () from Derived is executed by overriding the function in Base.

VIRTUAL FUNCTION:
A virtual function is a member function which is declared within a base class and is re-defined (Overriden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at Run-time.

Rules for Virtual Functions:

1. Virtual functions cannot be static.
2. A virtual function can be a friend function of another class.
3. Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.
4. The prototype of virtual functions should be the same in the base as well as derived class.
5. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
6. A class may have virtual destructor but it cannot have a virtual constructor.

Consider the following simple program showing run-time behavior of virtual functions.

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

class base {
public:
   virtual void print()
   {
      cout << "print base class" << endl;
   }

   void show()
   {
      cout << "show base class" << endl;
   }
};

class derived : public base {
public:
   void print()
   {
      cout << "print derived class" << endl;
   }
```

```
    void show()
    {
        cout << "show derived class" << endl;
    }
};

int main()
{
    base* bptr;
    derived d;
    bptr = &d;

    // virtual function, binded at runtime
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();
}
```

Output: print derived class
show base class

Explanation: Runtime polymorphism is achieved only through a pointer (or reference) of base class type. Also, a base class pointer can point to the objects of base class as well as to the objects of derived class. In above code, base class pointer 'bptr' contains the address of object'd'ofderivedclass.
Late binding(Runtime) is done in accordance with the content of pointer (i.e. location pointed to by pointer) and Early binding(Compile time) is done according to the type of pointer, since print() function is declared with virtual keyword so it will be bound at run-time (output is print derived class as pointer is pointing to object of derived class) and show() is non-virtual so it will be bound during compile time(output is show base class as pointer is of base type). **NOTE:** If we have created a virtual function in the base class and it is being overridden in the derived class then we don't need virtual keyword in the derived class, functions are automatically considered as virtual functions in the derived class.