Name of Teacher: Miss Radhika M. Patil

Class: B.Sc. Computer Science (Entire)- II **Semester : 3**

Course Title: Object Oriented Programming using C++

Unit 4 : Inheritance and Polymorphism

- **Inheritance:**

1. Inheritance is one of the most important characteristic of OOP.
2. Inheritance is the process in which a new class is created from existing class.
3. Inheritance is the process in which object of one class acquires the properties of object of another class.
4. The principal behind inheritance is that each derived class shares common characteristics with class from which it is derived.
5. In OOP, concept of inheritance provides **Code Reusability.**
6. This means that we can add additional features to an existing class without modifying it.
7. The new class or derived class will have combined features of both the classes.
8. The mechanism of creating new class from existing class is called inheritance.
9. **Existing class** is known as **Base class or Parent class** and **new class** is called as **Derived class or Child Class.**
10. Derived class inherits some or all of properties from base class.
11. Inheritance is also called as “Parent-Child relationship” or “Has-a relationship”.

- **Defining a base class:**

Base class is a class that has been extended or inherited by another class. It allows extending class (i.e. derived class) to inherit its properties and behaviour. It is also called as Parent class or Super class.

Example:

```
class Person
{
        //statements;
};
```

A new class Employee can be created by inheriting above class Person as

```
class Employee: Person
{
      //statements;
};
```

Here class Person is said to be Base class of Employee.

- **Defining a Derived class**

1. Derived class can be defined by its relationship with base class in addition to its own details.
2. Derived class is also known as Sub class or child class.
3. The general form of defining derived class is:

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

4. Colon indicates that derived_class_name is derived from Base_class_name.
5. Visibility mode is optional. But if present it may be either private or public or protected.
6. **Default** visibility mode is **private**.
7. Visibility mode specifies whether features of base class are privately derived or publically derived.

Example:

```
class A
{
      //members of class A
};

class B: public A                    //public derivation
 {
       //members of B
 };
```

8. When Base class is privately inherited by derived class then public members of base class become private members of derived class.
9. Therefore, private members of base class can only be accessed by function of derived class. They are inaccessible to objects of derived class.
10. When base class is publically inherited then public members of a base class remain public members of derived class. So the are accessible to objects of derived class.
11. In both the cases private members of base class are not inherited.
12. When class is derived in protected mode it changes mode of inherited members in derived class.
13. The public and protected members become protected members for derived class in protected derivation.
14. So inherited members can only be accessed through member function of derived class.

Access Specifiers in Base Class	**Access Specifiers in Derived Class Derivations (Inheritance)**		
	Private	Protected	Public
Public	Private	Protected	Public
Protected	Private	Protected	Protected
Private	Can’t be inherited	Can’t be inherited	Can’t be inherited

- **Types of Inheritance:**

In C++, we have 5 different types of Inheritance. Namely,

1.Single Inheritance

2.Multiple Inheritance

3.Hierarchical Inheritance

4.Multilevel Inheritance

5.Hybrid Inheritance (also known as Virtual Inheritance)

1. Single/ Simple inheritance:

1. Single inheritance is defined as the inheritance in which **only one derived class** is inherited from the **only one base** class.

2. It is the most simplest form of inheritance.

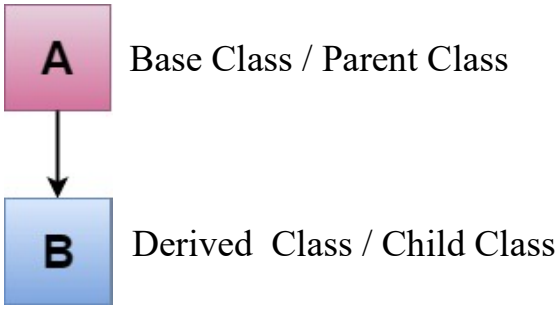

Fig. Single Inheritance

3. As shown in fig. one class B is derived from only one base class A.

Syntax:

```
class Derived_class_name : visibility_mode Base_class_name
{
        private:
              // derived class data members and member functions
        public:
              // derived class data members and member functions
         protected:
              // derived class data members and member functions
};
```

Program:

```
class A
{
                int a;
        public:
                int b;
                void get_ab();
                int get_a();
                void show_a();
};

class B: public A
{
                int c;
       public:
                void add();
                void display();
};

void A:: get_ab()
{
     a=5;
     b=10;
}

int A::get_a()
{
    return a;
}

void A :: show_a()
{
    cout<<"\n a="<<a;
}

void B:: add()
{
       c= get_a()+ b;
}

void B :: display()
{
       int i= get_a();
       cout<<"\n a="<<i;
       cout<<"\n b="<<b;
       cout<<"\n Addition="<<c;
}
```

```
void main()
{
        B d;
        d.get_ab();
        d.add();
        d.show_a();
        d.display();
        d.b=20;
        d.add();
        d.display();
        getch();
}
```

Output:

2. Multilevel Inheritance:

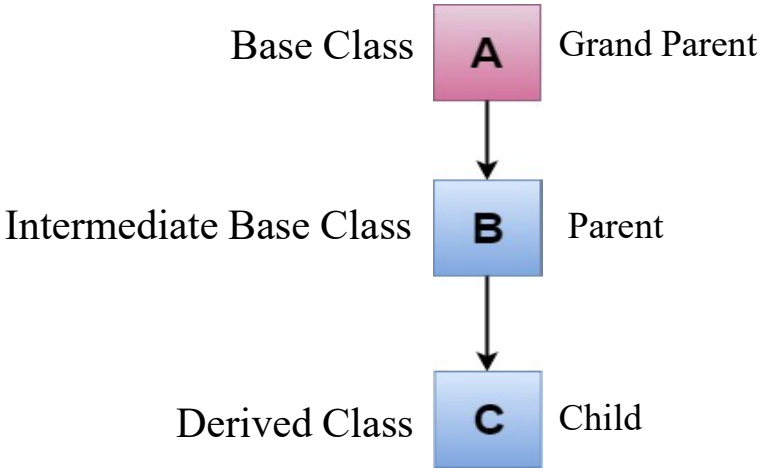

Fig. Multilevel Inheritance

1. The mechanism of deriving a class from another derived class is called multilevel inheritance.
2. Here derived class acts as intermediate base class for its derived class.
3. When one class inherits another class which is further inherited by another class, it is known as multi level inheritance.
4. Inheritance is transitive so the last derived class acquires all the members of all its base classes.
5. In above fig. class A acts as Base class for derived class B which in turn serve as a base class for derived class C.
6. The class B is called intermediate Base class because it provides a link for inheritance between A and C.
7. The chain A→ B→ C is called inheritance path.

Program:

```
class Student
{
        protected:
                int rno;
        public:
                void get (int);
                void put();
};

void Student:: get (int a)
{
        rno=a;
}

void Student :: put()
{
    cout<<"\n Roll No.="<<rno;
}
```

```
class  Test : public Student
{
        protected:
                int m1, m2;
        public:
                void get_marks( int a, int b)
                {
                    m1=a;
                    m2=b;
                }

                void put_marks()
                {
                   cout<<"\n Marks 1="<<m1;
                   cout<<"\n Marks 2="<<m2;
                }
};
```

```
class  Result : public Test
{
        int tot;
    public:
        void display()
        {

         tot=m1+m2;

            put();
            put_marks();
            cout<<"\n Total="<<tot;
        }
};
```

```
void main()
{
      Result r;
      r.get(100);
      r.get_marks(70,85);
      r.display();
      getch();
}
```

Output:

Roll No = 100
Mark1=70
Mark 2=85
Total=155

3. Multiple Inheritance:

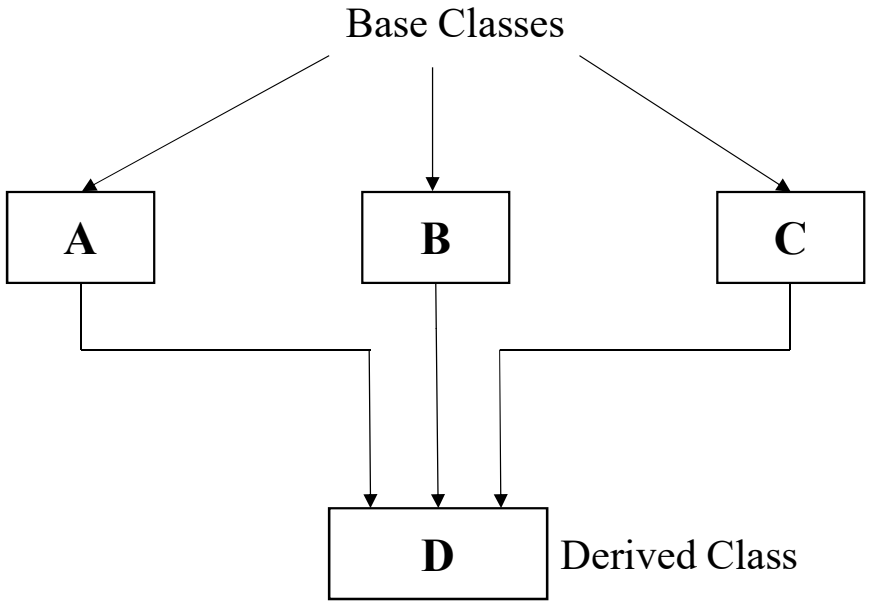

Fig. Multiple Inheritance

1. Inheritance in which one class is derived from two or more base classes is known as Multiple inheritance.
2. Multiple inheritance allow to combine features of several existing classes as a starting point for defining new classes.
3. Its like a child inheriting physical properties of one parent and intelligence from another.

Syntax:

```
class Derived: visibility_mode Base1, visibility_mode Base2,…….., visibility_mode Base n
{
        // Data members of Derived Class
};
```

Example:

```
class D: public A, public B, public C
{
      // Data members of Class D;
};
```

Program:

```
class M
{
    protected:
                int m;
    public:
                void get_m(int x)
                 {
                       m=x;
                 }
};

class N
{
      protected:
                 int n;
      public:
                void get_n(int x)
                   {
                       n=x;
                    }

};
```

```
class P: public M, public N
{
          public:
                    void display()
                    {
                          cout<<"\n m="<<m;
                          cout<<"\n n="<<n;
                          cout<<"\n Sum="<<m+n;
                     }
};


void main()
{
          P   p;
           p.get_m(10);
           p.get_n(20);
           p.display();
    getch();
}
```

Output:

m=10
n=20
Sum=30

- **Ambiguity in Multiple Inheritance:**

Multiple inheritance may face a problem when function with same name appears in more than one base classes. There is a confusion about which function is used by derived class when we inherit these two base classes.

Program:

```
class M
{
        public:
                void show()
                {
                        cout<<"\n Class M";
                 }
};

class N
{
        public:
                void show()
                {
                        cout<<"\n Class N";
                 }
};
class P: public M, public N
{
        public:
                void show()
                {
                        M:: show(); // Invokes show() function of class M
                         N:: show(); // Invokes show() function of class N

                }
};
  void main()
 {
                P p;
                p.show();
        getch();
 }
```

In above program there are two base classes M and N. When these two base classes are inherited by derived class P then there will arrive a question about which show() is to be called. This can be solved by using scope resolution operator with class name.

- **Ambiguity in Single Inheritance:**

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

```
class B: public A
{
        public:
                void display()
                {
                    cout<<"\n Derived class";
                }
};
```

1. In this case the function display() in derived class overrides the inherited function in base class so a simple call to display() function by 'B' type object will invoke function defined in class B only.
2. We may invoke function defined in class A by using scope resolution operator to specify a class.

```
void main()
{
        B b;
        b.display();  //invokes display() of class B
        b.A:: display();        //  invokes display() of class A
       getch();
}
```

4. Hierarchical Inheritance

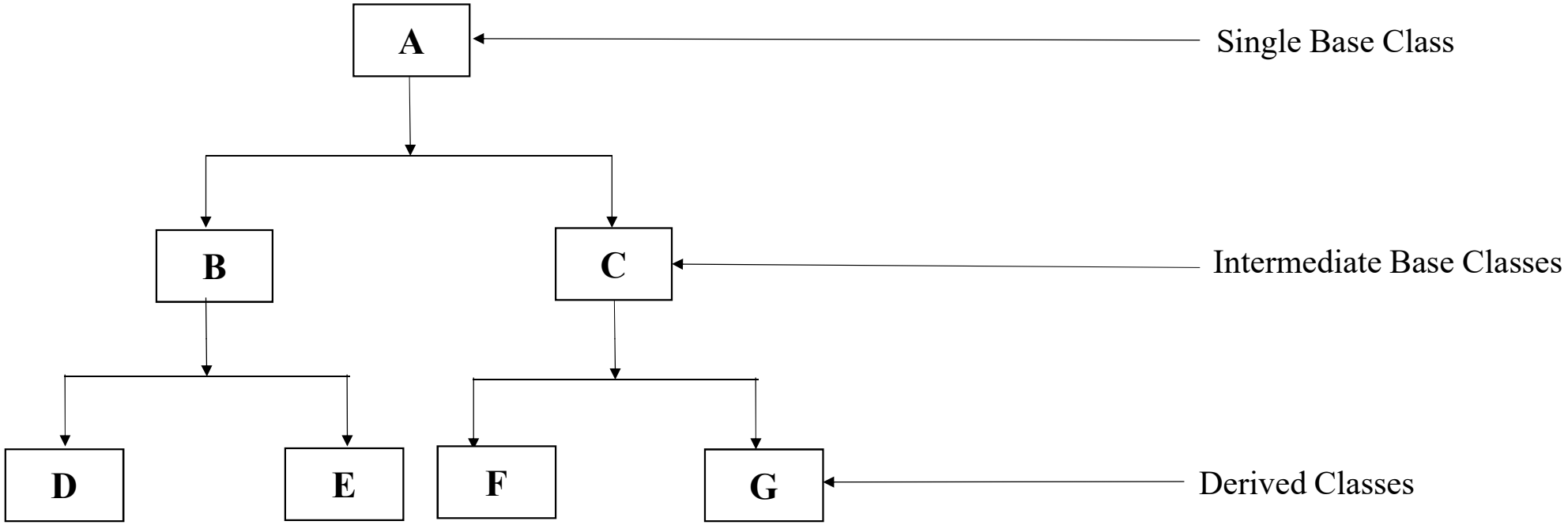

Fig. Hierarchical Inheritance

1. When more than one classes are derived from single base class such inheritance is called hierarchical inheritance.
2. In this inheritance the features that are common in levels are included in base class.
3. Derived classes can also further be inherited in same way , thus it finds hierarchy of classes or tree of classes which rooted at single base class.
4. In above diagram class A is base class from which class B and class C are derived, from class B we derive two classes D and E.and from class C we derive two classes F and G.
5. The whole arrangement from hierarchy rooted at single base class A.

Syntax:

```
class Base
{
        // Base class statements;
};

class Derived1: visibility_mode  Base
{
        // derived1 class statements;
};
class Derived2: visibility_mode  Base
{
        // derived2 class statements;
};
```

Program:

```
class Number
{
        public:
                int a;
                void get_num()
                {
                    cout<<"\n Enter a number:";
                    cin>>a;
                }
};

class Square: public Number
{
        public:
                void sqr()
                {
                        get_num();
                        cout<<"\n Square of number="<<a*a;
                }
};

class Cube: public Number
{
        public:
                void cub()
                {
                        get_num();
                        cout<<"\n Square of number="<<a*a*a;
                }
};

void main()
{
        Square s;
        s.sqr();
        Cube c;
        c.cub();
    getch();
}
```

5. Hybrid Inheritance:

In this inheritance one or more types of inheritance are combined together and used.

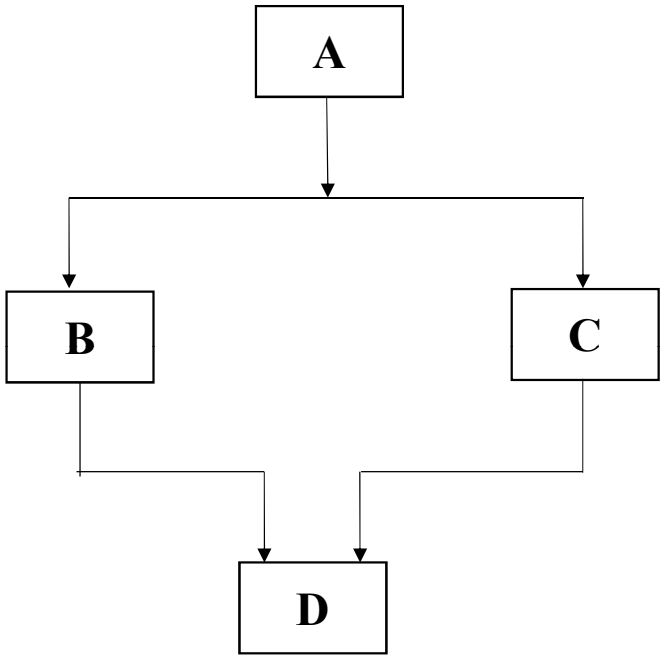

Fig. Hybrid Inheritance:

Program:

Write an OO C++ program to display student result using Hybrid Inheritance.

```
class Student
{
        protected:
                int roll_no, char name[20];
        public:
                void get()
                {
                    cin>>roll_no>>name;
                }
};

class Test: public Student
{
        protected:
                int m1,m2;
        public:
                void get_marks()
                {
                    cin>>m1>>m2;
                }
};

class Sports: public Student
{
        protected:
                int score;
        public:
                void get_score()
                {
                    cin>>score;
                }
};
```

```
class Result: public Test, public Sports
{
        int tot;
        public:
                void display()
                {
                    tot= m1+m2+score;
                   cout<<"\n"<<roll_no<<"\t"<<name<<"\t"<<m1<<"\t"<<m2<<"\t"<<score<<"\t"<<tot;
                }
};
void main()
{
        int n;
        Result r[5];
         cout<<"\nEnter number of students:";
         cin>>n;
        for(int i=0;i<n;i++)
        {
             cout<<"\n Enter details of student "<<i+1;
             r[i].get_marks();
             r[i].get_score();
        }
         cout<< "\n Roll No.\tName\tSub1\tSub2\tSports mark\tTotal";
         for (int j=0;j<n;j++)
         {
                  r[j].display();
         }
        getch();
        }
```

THANK YOU…