Name of Teacher: Miss Radhika M. Patil

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

Course Title: Object Oriented Programming using C++

- ## Polymorphism (Binding):

1. Polymorphism is one of the most important feature of OOP.
2. Poly means many and morph means forms.
3. It simply means one name multiple forms.
4. Typically polymorphism occurs when there is hierarchy of classes and they are related with inheritance.

- ## Types of Polymorphism:

There are two types of polymorphism:

1. Compile time
2. Run Time

1. Compile Time polymorphism (Static Binding):

- It is also called as static binding or early binding
- It can be achieved through function overloading and operator overloading.
- Overloaded member functions are selected for invoking by matching the arguments both type and number.
- This information is known to compiler at compile time.
- Thus compiler is able to select appropriate function for particular call at compile time itself.
- Hence, it is called compile time polymorphism i.e. object is bound to its function called at compile time.

2. Run Time polymorphism (Dynamic Binding):

1. It is also called as dynamic or late binding.
2. Appropriate function would be selected while program is running
3. Run time polymorphism is implemented with inheritance and virtual function.

• Virtual function:

1. Polymorphism refers to the property by which object belonging to different classes are able to respond to same message .
2. A base pointer even when it contains object of derived class , it will always execute function in base class.

```
Base *b;
Derived d;
b=&d;
b→show();
```

4. In the case compiler simply ignore the contents of pointer chooses member function that matches type of pointer .
5. So statement b→show(); will invoke show() function of base class.
6. When we want to invoke show() function in derived class using base class pointer then virtual function is used.
7. When we use **same function name in base and derived class then function in base class should be declared as virtual**.
8. When function is made virtual, compiler determines which function to use or invoke at run time based on contents of object pointed by base pointer rather than type of pointer.
9. Thus by making base pointer to point different derived class object, we can execute different versions of virtual function.

Program:

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

class Derived: public Base
{
        public:
                 void show()
                {
                    cout<<"\n Derived class";
                }
};

 void main()
{
            Base *b;     // Base class pointer
            Derived d;  // Derived class object
            b=&d;
            b→show();   // late binding occurs
    getch();
}
```

Output:

Derived class

b→show() this will execute version of function show() in derived class because base class pointer points to derived class object.

- ## Pure Virtual Function:

1. Pure virtual Functions are virtual functions with no definition. They start with **virtual** keyword and ends with = 0
2. Pure virtual function is a type of function which has only one function declaration .
3. It doesn't have function definition or implementation.
4. A pure virtual function can be declared in two ways – either function doesn't contain a body or virtual function may be equals to 0 in function declaration itself.
5. When the function has no definition, such function is known as "**do-nothing**" function.
6. The **pure virtual function** is known as "**do-nothing**" function.
7. A pure virtual function is a function declared in the base class that has no definition relative to the base class.
8. A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.
9. The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.

Program:

```
class Base
{
        public:
                virtual void show()=0; //pure virtual function
};

class Derived: public Base
{
                public:
                        void show()
                        {
                                cout<<"\n Derived class";
                        }
};

void main()
{
                Base *b;     // Base class pointer
                Derived d;  // Derived class object
                b=&d;
                b→show();   // late binding occurs
        getch();
}
```

Output:

Derived class

In above program Base is abstract Base class with pure virtual function show(). Hence we can't create object of class Base.

- **Abstract Base Class:**

- An **abstract class in C++** is a class that has at least one pure virtual function (i.e., a function that has no definition).
- The classes inheriting the abstract class must provide a definition for the pure virtual function; otherwise, the subclass would become an abstract class itself.
- Abstract classes are essential to providing an abstraction to the code to make it reusable and extendable.

- **Characteristics of Abstract Class:**

1. Abstract class contains at least one pure virtual function.
2. Abstract class can't be instantiated (i.e. object of abstract class can't be created), only pointer or references of abstract class can be created.
3. Abstract class can have normal functions and variables along with pure virtual function.
4. Derived classes of abstract class must implement its pure virtual function, otherwise they become abstract classes.

Program:

```
class Sample
{
      public:
            virtual void getdata() {}
            virtual void display()=0;
};

class Derived: public Sample
{
    public:
          void getdata()
          {
              cout<<"\n This is Derived class";
          }

          void display()
          {
                cout<<"\n Statements in derived class";
          }
};

void main()
{
            Sample *s;
            Derived d;
            s=&d;
            s→getdata();
            s→display();
   getch();
}
```

Output:

This is Derived class
Statements in derived class

Here, the two pure virtual functions getdata() and display() are defined in base class in two different possible ways. Since pure virtual functions are defined in class Sample so object of class Sample can't be created.

Assignment:

1. Differentiate between constructor and destructor
2. Differentiate between classes and objects

THANK YOU…