Subject Name: **Object Oriented Programming Using C++**

Subject Code: **BCA-301 N**

Subject Topic: **Polymorphism**

Abhishek Dwivedi

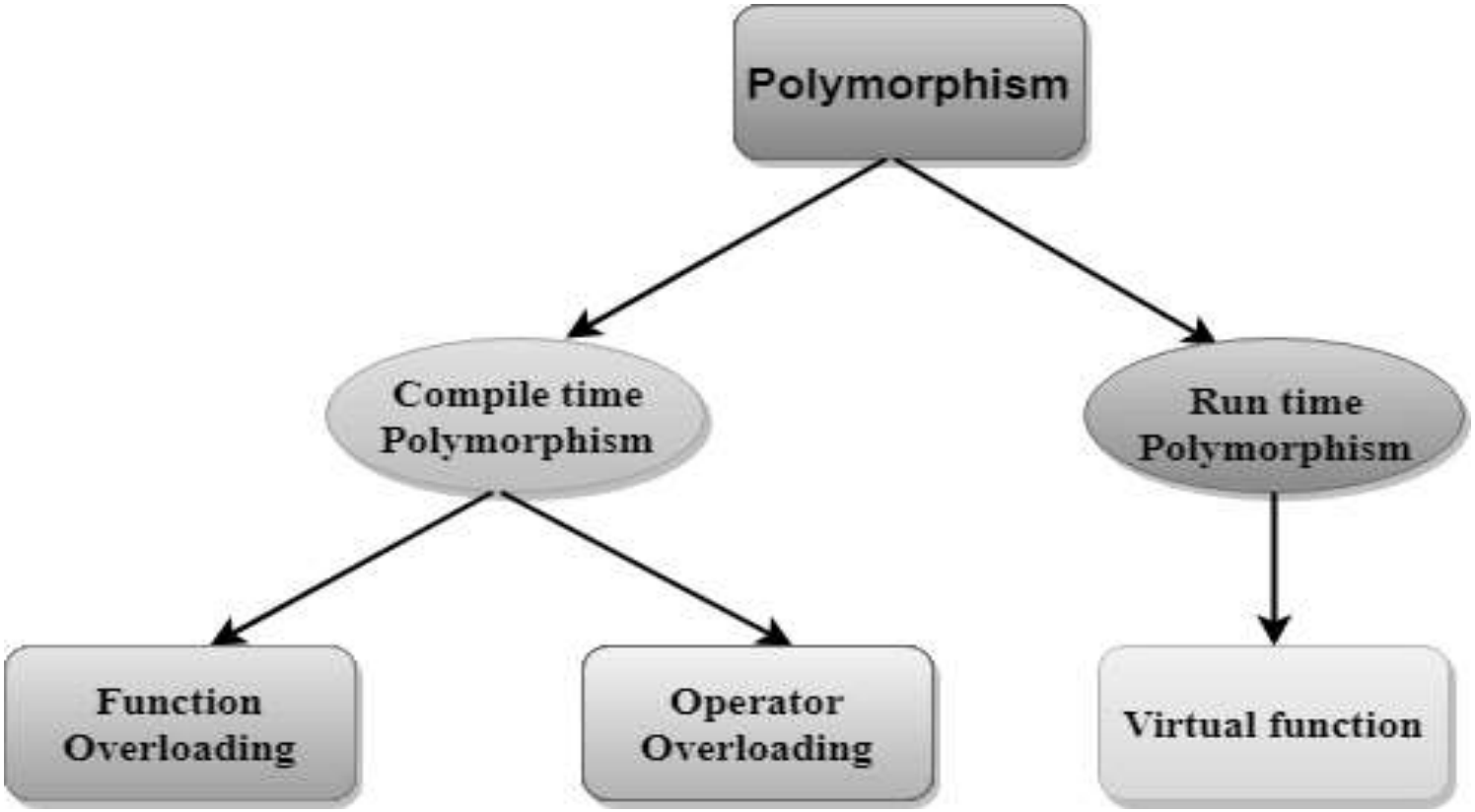
Assistant Professor

Department of Computer Application

UIET, CSJM University, Kanpur

Polymorphism

- The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms.



Compile time polymorphism

- The overloaded functions are invoked by matching the type and number of arguments. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading which is also known as static binding or early binding.

Run time polymorphism

- Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

Requirements for Overriding a Function

- Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class.

- Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

Example

```
class Base
{
public:
void show()
{ cout << "Base class";
 }
};
class Derived : public Base
 {
public:
void show()
 {
cout << "Derived Class";
}
}
void main()
{
    Derived d;
    d.show();
    getch();
}
```

Function Call Binding using Base class Pointer

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

```
void main()
{
    Base *b;            //Base class pointer
    Derived  d;         //Derived class object
     b = &d;
    b->show();          //Early Binding Occurs
    getch();
 }
```

- Although, the object is of Derived class, still Base class's method is called. This happens due to Early Binding.
- Compiler on seeing **Base class's pointer**, set call to Base class's show() function, without knowing the actual object type.

Virtual Functions

- Virtual Function is a function in base class, which is overridden in the derived class, and which tells the compiler to perform **Late Binding** on this function.

- Virtual Keyword is used to make a member function of the base class Virtual.

- In Late Binding function call is resolved at runtime. Hence, now compiler determines the type of object at runtime, and then binds the function call. Late Binding is also called **Dynamic** Binding or **Runtime** Binding.

Using Virtual Keyword

- Making base class's methods virtual by using virtual keyword while declaring them. Virtual keyword will lead to Late Binding of that method.

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

```
void main()
{
   Base *b;                    //Base class pointer
   Derived  d;                 //Derived class object
   b = &d;
   b->show();                  //Late Binding Occurs
   getch()
}
```

- On using Virtual keyword with Base class's function, Late Binding takes place and the derived version of function will be called, because base class pointer pointes to Derived class object.

References:

- www.studytonight.com
- www.tutorialpoint.com
- www.geeksforgeeks.org
- “Object oriented programming in C++” Robert Lafore
- “Object oriented programming with C++”, E.Balagurusamy