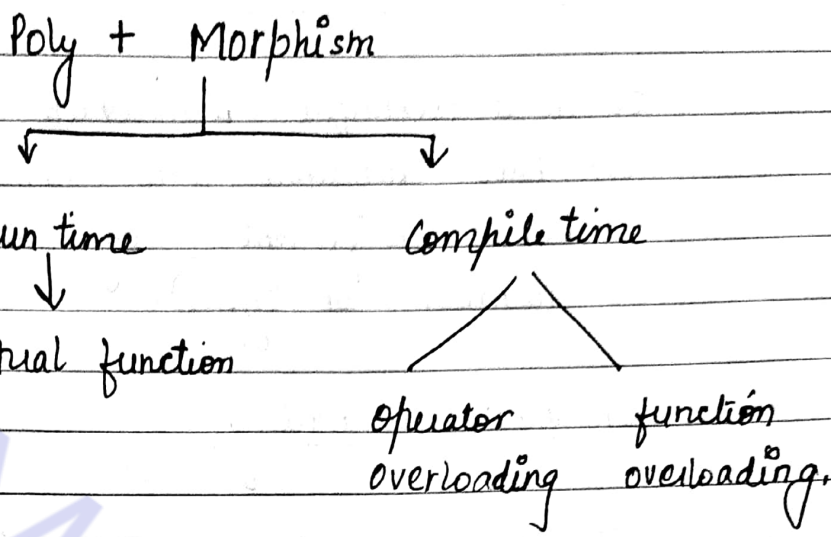
# UNIT-3

# POLYMORPHISM




It is made up of two words Poly and morphism where poly means many and morphism means forms Polymorphism is a crucial technique of ~~oops~~ (OOPS)

Polymorphism occurs when there is a hierarchy of classes and they are related There are two types of Polymorphism

1. Compile time
2. Run time

* Compile time polymorphism is executed and implemented using overloading (Function and operator) where as run time polymorphism is implemented using virtual function.

Page No.
Date

## COMPILE TIME POLYMORPHISM

(•) **Function overloading**

It is a technique in which a program can have functions with same name but different arguments (in terms of number or types).

(•) **Operator overloading**

It is a technique where an operator can be represented and reused as many times as a program demands

To define the task of an operator we must specify what it means in relation to the class to which the operator is applied.
This is done with the help of a special function called operator function which describes the task to it.

## RUN TIME POLYMORPHISM

(•) **Virtual function**

It is a technique of polymorphism used to implement run time polymorphism where a member function of a class is

represented as a function which does not occupies any memory but exist only at the run time.

Virtual function can be invoked using arrow operator. C++ supports a mechanism in which, at run time the class objects are under consideration and the appropriate version of function is invoked. Since the function is linked with a particular class much later after the compilation. This process is termed as **LATE BINDING** and it is also known as **Dynamic binding** because the selection of the appropriate function is done dynamically at run time.

Page No.
Date

# OPERATOR OVERLOADING

```
#include <iostream.h>
#include <conio.h>
class abc
{
  public:
    int rollno;
    char grade;
    float marks;

abc (int m, char n, float g)
{
      rollno = m;
      grade = n;
      marks = g;
}
 void operator++(int)
{
    rollno = rollno + 1;
    grade = grade + 1;
    marks = marks + 1;
}
};
 int main()
{
  abc s (22, 'a', 20.1)
cout << " Before overloading " << s.rollno
    << s.grade << s.marks;
s++;
cout << " After overloading " << s.rollno << s.grade <<
```

s.marks;
getch();
}

# VIRTUAL FUNCTION

* Rules for virtual function :

1) It a member fn which is redefined in the derived class and the fn can be overridden.

2) Virtual function can-not be static & also cannot be friend of another class.

3) Virtual function defined in base class should be accessed using pointer or reference to achieve run-time polymorphism.

4) If we have virtual function & it is overridden in derived class then we do not need virtual keyword in the derived class.

5) A base class pointer can point to the object of base class as well as to the object of derived class.

6) For invoking a virtual function, we do not need dot operator instead we will use arrow operator (→)

```
#
# Class base
{
    public:
    virtual void show ()
    {
    cout << " We are in derived class";
    }
};
    int main ()
    {
    base * b;
    derived d;
    b = &d;
    b → show ();
     getch();
    return 0;
}
```

```
class derived : public base
{ public:
  void show ()
  { cout << "we are in derived class" }
```

Page No.
Date

# INHERITANCE

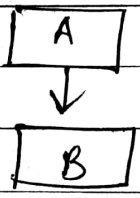
Inheritance is a feature of OOPs (Object oriented programming system) which allows the creation of a new class (base) with derivation of its own features and allows child class to have its own unique features

This is basically done by creating new classes and reusing the properties of the existing one.

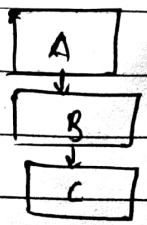
The mechanism of deriving a new class from the old one is known as Inheritance.

## Types of Inheritance

1) Single level Inheritance

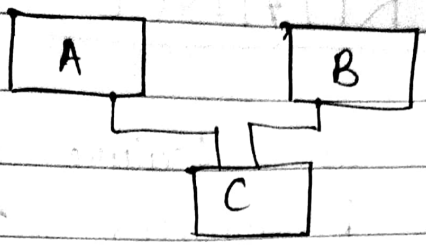



2) Multi-level Inheritance

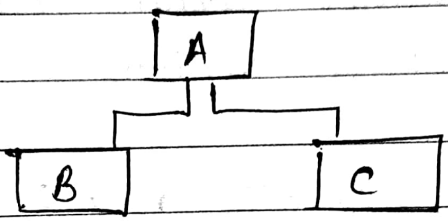


Page No.
Date

3) Multiple Inheritance




(4) Hierarchical inheritance.




(5) Hybrid Inheritance