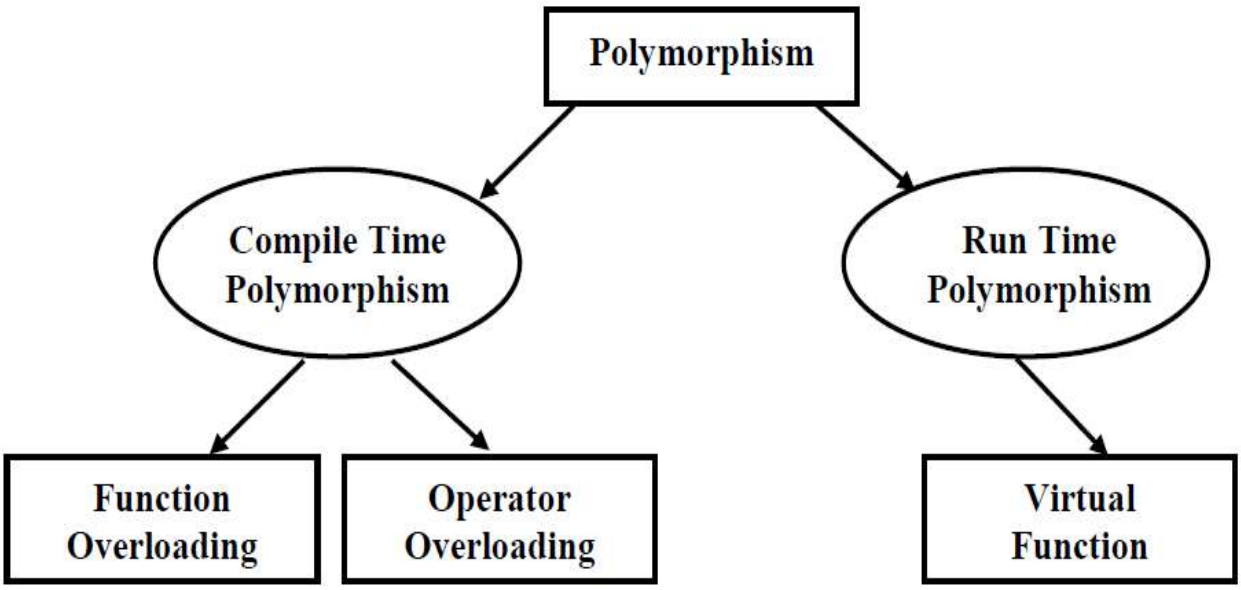
Polymorphism

Polymorphism:-**Polymorphism comes from two greek words, namely Poly and Morphos. Poly indicates many and Morphos indicates forms. Therefore, it is the existence of the same thing in different forms. Polymorphism enables one common interface for many implementations and for objects to act differently under different circumstances.**



Static or Compile time polymorphism:-**It is the polymorphism that occurs at compile time. IN static polymorphism the binding or the association of an object of a class and its member occurs at compile time .Overloaded constructors of a class and polymorphic non-virtual methods are classic examples of static or compile time polymorphism. Static polymorphism can be achieved by function overloading operator overloading.**

Function Overloading:- **When there are multiple functions with same name but different parameters then these functions are said to**

be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments.

Example:-

```
#include <iostream>
using namespace std;
class Ajay
{
   public:
   void fun(int a)
   {
      cout << "value of a is " << a <<”\n”;
   }
   void fun(double b)
   {
      cout << "value of b is " << b<<”\n”;
   }
   void fun(int a, int b)
   {
      cout << "value of a and b is " << a<< ", " << b;
   }
};
int main() {

   Ajay a;
   a.fun(7);
   a.fun(9.132);
   a.fun(85,64);
   return 0;
}
```

Output:

value of x is 7
value of x is 9.132
value of x and y is 85, 64

Operator Overloading:- **The process of making an operator to represent different behaviours in different circumstances is called operator overloading.**

We can overload all the operators in c++ except the following.

- **class member access operators(.,->)**
- **scope resolution operator(::)**
- **size operator(sizeof)**
- **condition operator(?:).**

Example:-

```
#include<iostream>
using namespace std;
class Complex
 {
private:
   int real, imag;
public:
   Complex(int r = 0, int i =0)  {real = r;   imag = i;}

   // This is automatically called when '+' is used with
   // between two Complex objects
   Complex operator + (Complex const &obj)
 {
       Complex res;
       res.real = real + obj.real;
```

```
        res.imag = imag + obj.imag;
        return res;
    }
    void print()
{
 cout << real << " + i" << imag << endl;
}
};
int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
```

Output:-

```
12 + i9
```

Dynamic or Run time Polymorphism:-**Dynamic polymorphism refers to the polymorphism that occurs at runtime. In dynamic polymorphism the binding or the association of an object of a class and its members occurs at runtime. In other words, when the appropriate member function is selected while the program is running.it is called runtime polymorphism. Runtime polymorphism is achieved through virtual function. Which is descried below.**

Function overridden or virtual member functions:-**on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.**

Example:-

```
#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;
    bptr->print();
    bptr->show();
    return 0;
}
```

Output:-

print derived class

show base class

Created by:-Ajay Kumar Verma