Polymorphism in C++

(These notes are adapted from https://www.geeksforgeeks.org/polymorphism-in-c/)

Polymorphism in C++

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. Real life example of polymorphism, a person at a same time can have different characteristic. Like a man at a same time is a father, a husband, a employee. So a same person posses have different behavior in different situations. This is called polymorphism. Polymorphism is considered as one of the important features of Object Oriented Programming.

In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

1. **Compile time polymorphism**: This type of polymorphism is achieved by function overloading or 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**.
     Rules of Function Overloading

```
// C++ program for function overloading

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:

    // function with 1 int parameter
    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name but 1 double parameter
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name and 2 int parameters
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y << endl;
    }
};

int main() {

    Geeks obj1;

    // Which function is called will depend on the parameters passed
    // The first 'func' is called
    obj1.func(7);

    // The second 'func' is called
    obj1.func(9.132);
```

```
    // The third 'func' is called
    obj1.func(85,64);
    return 0;
}
```

- Run on IDE
- Output:

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

- In the above example, a single function named *func* acts differently in three different situations which is the property of polymorphism.
- **Operator Overloading**: C++ also provide option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add to operands. So a single operator '+' when placed between integer operands , adds them and when placed between string operands, concatenates them. **Example**:

```
// CPP program to illustrate
// Operator Overloading
#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();
}
```

Run on IDE
Output:

```
12 + i9
```

In the above example the operator '+' is overloaded. The operator '+' is an addition operator and can add two numbers(integers or floating point) but here

the operator is made to perform addition of two imaginary or complex numbers. To learn operator overloading in details visit this link.

2. **Runtime polymorphism**: This type of polymorphism is achieved by Function Overriding.
   - **Function overriding** 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**.

```
// C++ program for function overriding

#include <bits/stdc++.h>
using namespace std;

// Base class
class Parent
{
    public:
    void print()
    {
        cout << "The Parent print function was called" << endl;
    }
};

// Derived class
class Child : public Parent
{
    public:

    // definition of a member function already present in Parent
    void print()
    {
        cout << "The child print function was called" << endl;
    }

};

//main function
int main()
{
    //object of parent class
    Parent obj1;

    //object of child class
    Child obj2 = Child();


    // obj1 will call the print function in Parent
    obj1.print();

    // obj2 will override the print function in Parent
    // and call the print function in Child
    obj2.print();
    return 0;
}
```

   - Run on IDE
   - Output:

```
The Parent print function was called
The child print function was called
```