Type of Inheritance

When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance.

We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied −

Public Inheritance − When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class but can be accessed through calls to the public and protected members of the base class.

Protected Inheritance − When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

Private Inheritance − When deriving from a private base class, public and protected members of the base

class become private members of the derived class.

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 the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person possesses 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

Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.

Function Overloading: When there are multiple functions with the 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.h>
using namespace std;
class CDGI
```

```
{
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() {

CDGI

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;
}
```

Operator Overloading: C++ also provide an 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();
}
```