OBJECT ORIENTED PROGRAMMING USING C++

UNIT-4

PREPARED BY,
MR.D.STANLEY,
AP/BCA,
THE AMERICAN COLLEGE,
MADURAI

Constructors in C++

A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) is created. It is special member function of the class because it does not have any return type.

Constructors in C++

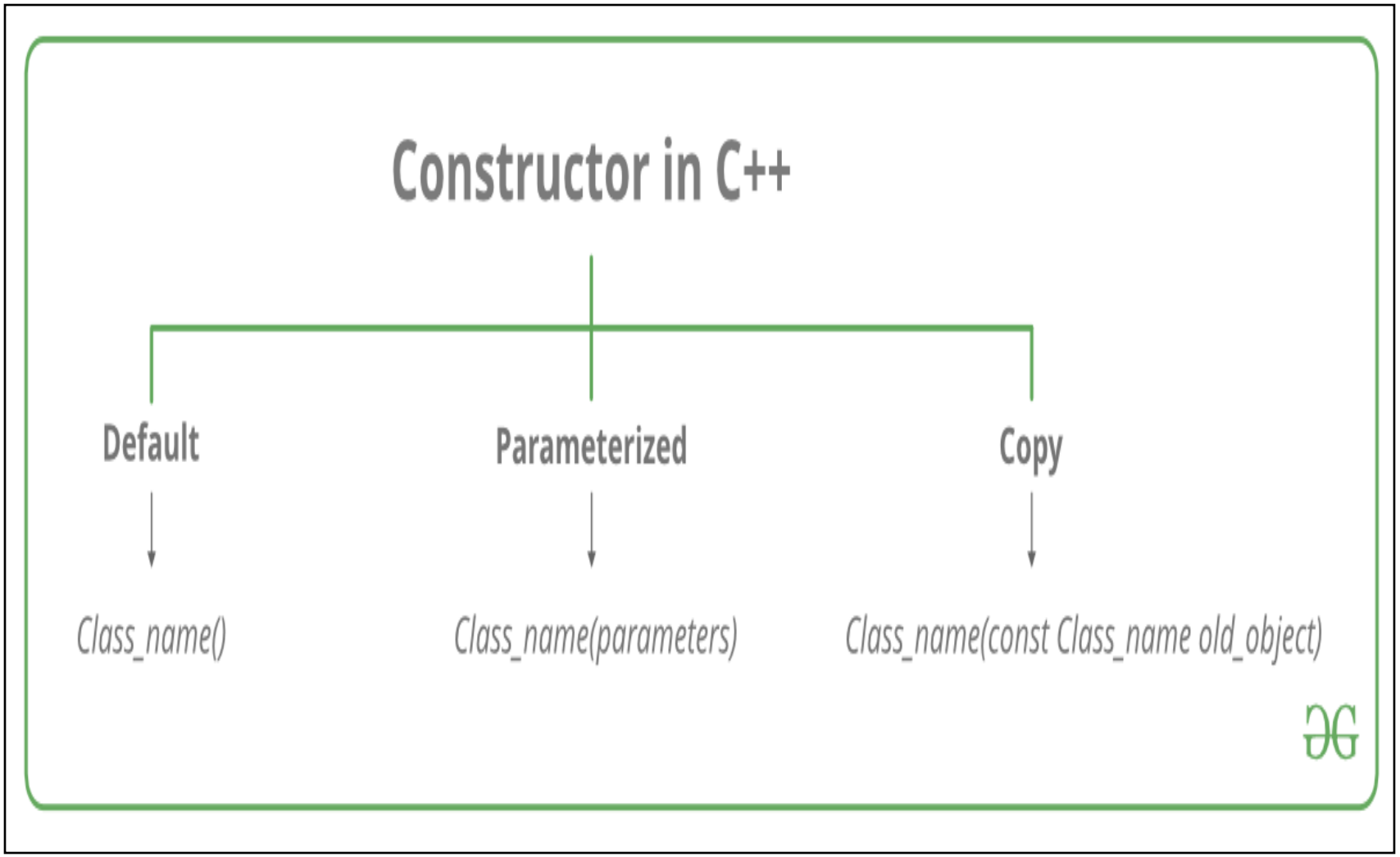
```
class MyClass
{     // The class
      public:           // Access specifier
        MyClass()
 {     // Constructor
          cout << "The American college";
        }
    };

    int main()
{
      MyClass myObj;     // Create an object of MyClass (this will call the
    constructor)
      return 0;
    }
```

Characteristics of Constructors

- They should be declared in the public section
- They do not have any return type, not even void
- They get automatically invoked when the objects are created
- They cannot be inherited though derived class can call the base class constructor
- Like other functions, they can have default arguments
- You cannot refer to their address
- Constructors cannot be virtual

Types of Constructors



Types of Constructors

- **Default Constructors:** Default constructor is the constructor which doesn't take any argument. It has no parameters.

Default constructor

```
#include <iostream>
using namespace std;

class construct
{
public:
    int a, b;

    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};

int main()
{

    construct c;
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```

Parameterized Constructors

- It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object.

Parameterized Constructors

```
#include <iostream>
using namespace std;

class Point
{
private:
   int x, y;

public:
   // Parameterized Constructor
   Point(int x1, int y1)
   {
      x = x1;
      y = y1;
   }

   int getX()
   {
      return x;
   }
   int getY()
   {
      return y;
   }
};
```

Parameterized Constructors

```
int main()
{
     Point p1(10, 15);
    cout << "p1.x = " << p1.getX() << ", p1.y = "
   << p1.getY();
    return 0;
}
```

Copy Constructors

C++ provides a special type of constructor which takes an object as an argument and is used to copy values of data members of one object into another object. In this case, copy constructors are used to declaring and initializing an object from another object.

Example:

Calc C2(C1);

Or

Calc C2 = C1;

C++ Destructor

- A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.
- A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).
- Note: C++ destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

C++ Destructor

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
           cout<<"Constructor Invoked"<<endl;
        }
        ~Employee()
        {
           cout<<"Destructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    return 0;
}
```

C++ Inheritance

- In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.
- In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

Advantage of C++ Inheritance

- **Code reusability:** Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

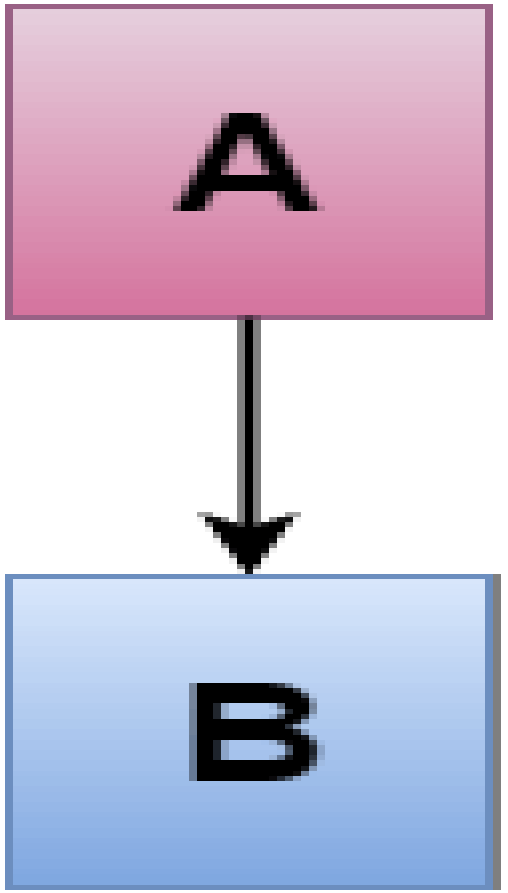
Types Of Inheritance

C++ supports five types of inheritance:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

Single Inheritance

- **Single inheritance** is defined as the inheritance in which a derived class is inherited from the only one base class.



Single Inheritance

```
#include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
   };
int main(void) {
     Programmer p1;
     cout<<"Salary: "<<p1.salary<<endl;
     cout<<"Bonus: "<<p1.bonus<<endl;
    return 0;
}
```

Multilevel Inheritance

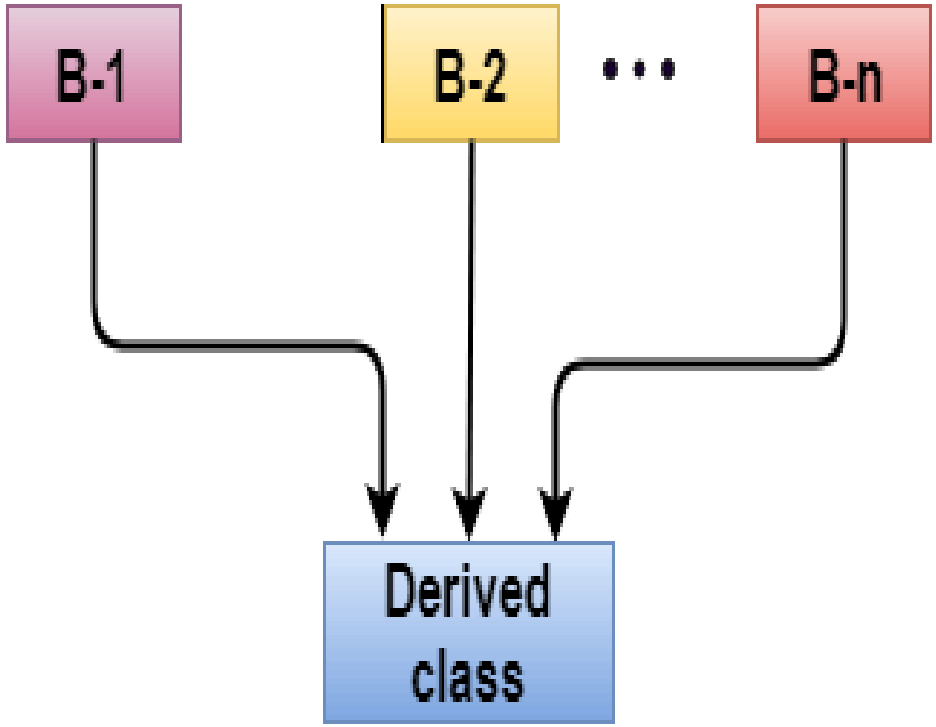
- **Multilevel inheritance** is a process of deriving a class from another derived class.

Multilevel Inheritance

```
#include <iostream>
using namespace std;
 class Animal {
   public:
 void eat() {
    cout<<"Eating..."<<endl;
 }
   };
   class Dog: public Animal
   {
      public:
     void bark(){
    cout<<"Barking..."<<endl;
     }
   };
   class BabyDog: public Dog
   {
      public:
     void weep() {
    cout<<"Weeping...";
     }
   };
int main(void) {
    BabyDog d1;
    d1.eat();
    d1.bark();
     d1.weep();
     return 0;
}
```

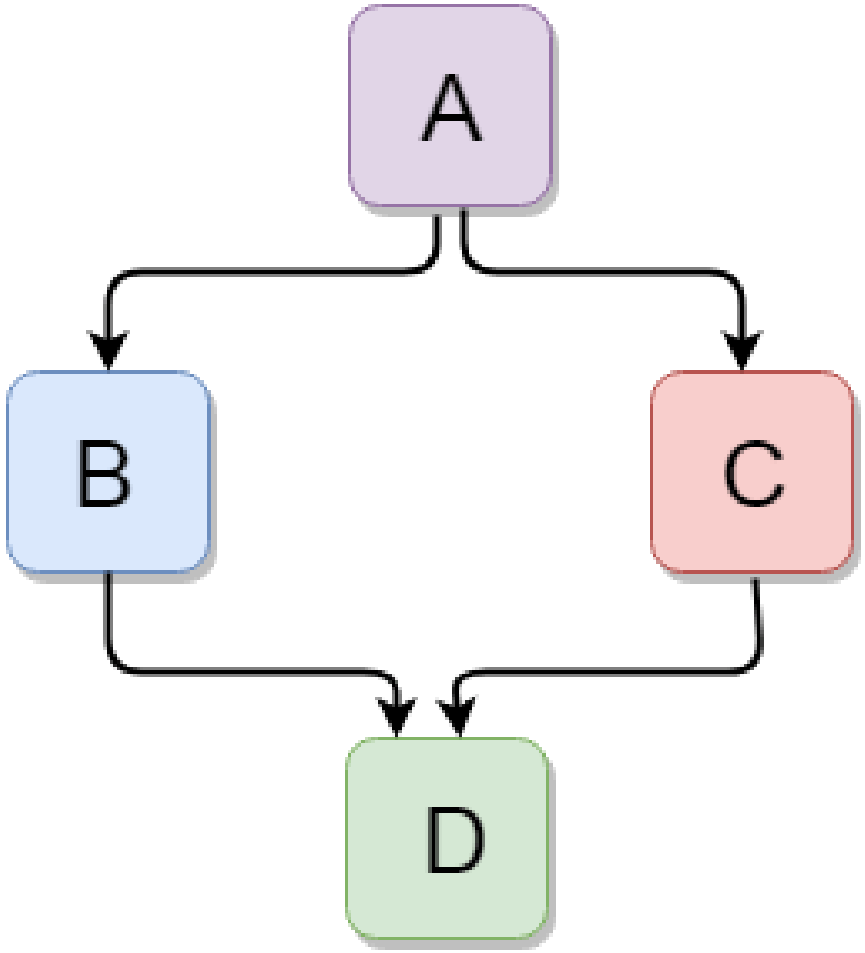
Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



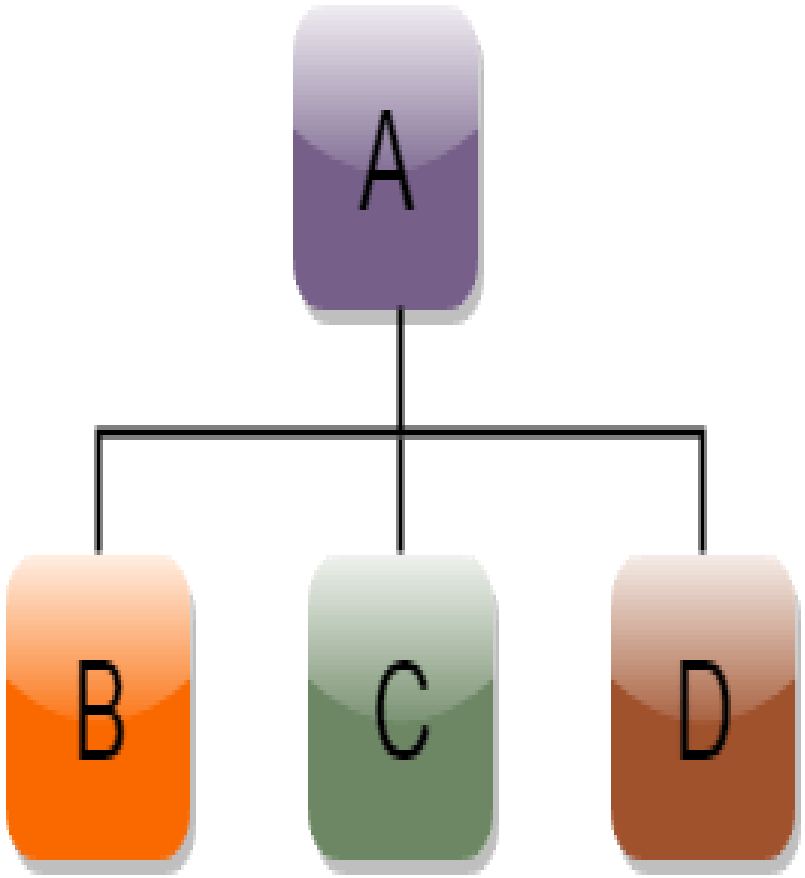
Hybrid Inheritance

- Hybrid inheritance is a combination of more than one type of inheritance.



Hierarchical Inheritance

- Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



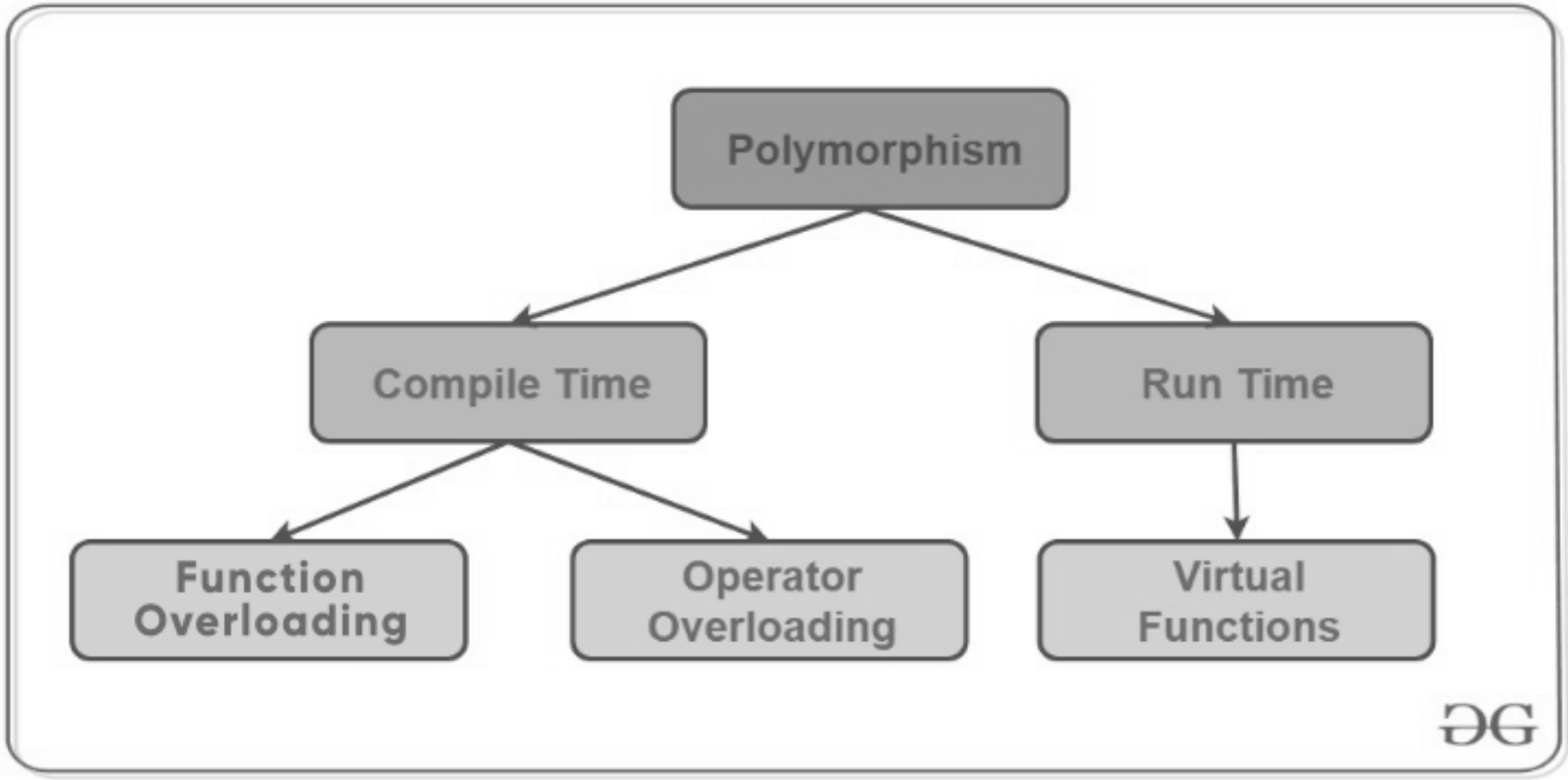
Polymorphism

- 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. A 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 posses 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

Polymorphism



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**.

Function Overloading

```
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;
    }
};
```

Function Overloading

```
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;
}
```

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 two operands. So a single operator ‘+’ when placed between integer operands , adds them and when placed between string operands, concatenates them.

Operator Overloading

```
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; }
};
```

Operator Overloading

```
int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to
   "operator+"
    c3.print();
}
```

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++ Virtual Functions

- In C++, we may not be able to override functions if we use a pointer of the base class to point to an object of the derived class.
- Using **virtual functions** in the base class **ensures** that the function can be overridden in these cases.
- Thus, **virtual functions** actually fall under **function overriding**.