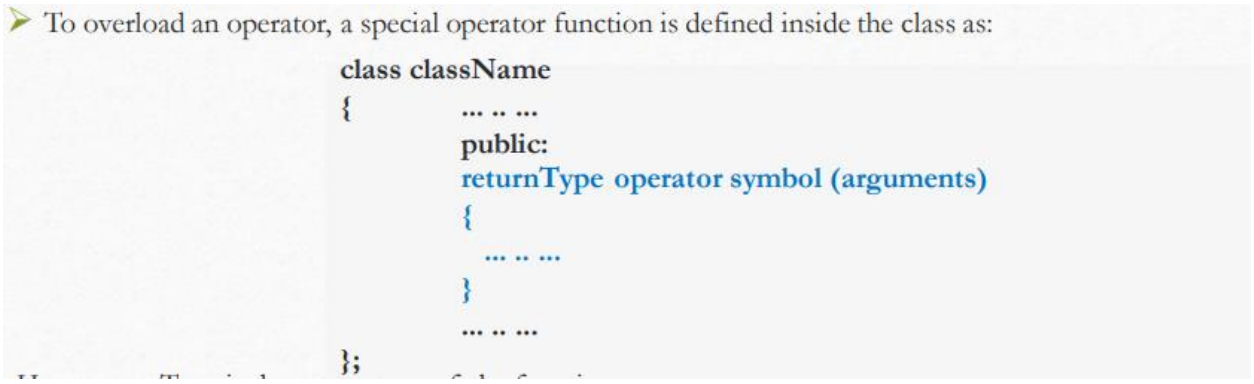
Autumn-2018

Group A

>>

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.



➤ To overload an operator, a special operator function is defined inside the class as:

```
class className
{           ... .. ...
            public:
            returnType operator symbol (arguments)
            {
              ... .. ...
            }
            ... .. ...
};
```

>>

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


class Float
{
      float num;
   public:
      void accept()
      {
            cin>>num;
      }
      Float  operator+(Float &a)
      {
            Float t;
            t.num=num+a.num;
```

```
                return t;
        }
        Float  operator-(Float &a)
        {
                Float t;
                t.num=num-a.num;
                return t;
        }
        Float operator*(Float &a)
        {
                Float t;
                t.num=num*a.num;
                return t;
        }
        Float  operator/(Float &a)
        {
                Float t;
                t.num=num/a.num;
                return t;
        }
        void display()
        {
                cout<<num;
        }
};
int main()
{
        Float a1, a2, a3;
        a1.accept();
        a2.accept();
        a3=a1+a2;
        cout<<"\n Addition of Two Numbers       :   ";
        a3.display();
        a3=a1-a2;
        cout<<"\n Subtraction of Two Numbers    :   ";
        a3.display();
        a3=a1*a2;
        cout<<"\n Multiplication of Two Numbers  :   ";
        a3.display();
        a3=a1/a2;
        cout<<"\n Division of Two Numbers       :   ";
        a3.display();
        return 0;
}
```

>>

A friend function is a non-member function of the class to which it has been defined as friend. Therefore it just uses the functionality (functions and data) of the class. So it does not consist the implementation for that class. That's why it cannot be used to overload the assignment operator.

>>

i.

Operator Overloading does not mean changing the function and behavior of that particular operator forever or for the built in data types. It only facilitates you to perform operations on your user defined data types similar to the ones you do on integral types.

ii.

Overloading an operator cannot change its precedence.

Overloading an operator cannot change its "arity" (i.e. number of operands)

<u>>></u>

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

We can summarize the different access types according to - who can access them in the following way –

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

<u>>></u>

```
#include <iostream>

using namespace std;

class Mammal {

  public:

    Mammal() {

      cout << "Mammals can give direct birth." << endl;

    }

};

class WingedAnimal {

  public:

    WingedAnimal() {

      cout << "Winged animal can flap." << endl;
    }
};
class Bat: public Mammal, public WingedAnimal {};
int main() {
    Bat b1;
    return 0;
}
```

__>>__

When we inherit class into another class then object of base class is initialized first. If a class do not have any constructor then default constructor will be called. But if we have created any parameterized constructor then we have to initialize base class constructor from derived class.

```
public class Abc
{
    public int p, q;
    public Abc(int p1, int p2)
    {
        p = p1;
        q = p2;
    }
    public int sum(int x, int y)
    {
        return (x + y);
    }
}
//derived class/ child class
public class Pqr : Abc
{
    public int a;
    public Pqr(int a1,int p1, int p2):base(p1,p2)
    {
        a = a1;
    }
    public int sub(int x, int y)
    {
        return (x - y);
    }
}
```

3(a).

When deriving a class from a public base class, public members of the base class become public members of the derived class, the private members in the base class cannot be directly accessed in the derived class.

3(b).

```
error: request for member 'cheers' is ambiguous
```

```
#include<iostream>
using namespace std
class A
{
  public:
        void cheers() { cout<<"Class A: Hip-hip-hooray";}
};
class B
{
  public:
        void cheers() { cout<<"Class B: Hip-hip-hooray";}
};
class C: public A, public B{};
```

```
int main()

{

    C obc;

    obc.A::cheers();

    cout<<endl;

    obc.B::cheers();

}
```

3(c).

Yes.

Constructor are use to initialize data members.

1. Constructor are **not inherited**. So base class constructor are not present in derived class. Thus derived class needs its own constructor.
2. If you don't create constructor then **default constructor** are created automatically by compiler.
3. Constructor are used while creation of object, thus having constructor for a class is mandatory.