OPERATOR OVERLOADING

- The mechanism of giving a different meaning to an operator is known as operator overloading.
- Operator overloading is done by using a special member function called as "OPERATOR" function.

Syntax of operator function:

```
return_type classname:: operator op(arguments)
{
 Block of Statements;
}
```

- **op** means the operator that is to be overloaded.

- **Types of operators which can be overloaded are:**
  1. **Unary operators**
  2. **Binary operators**
  3. **Special operators**
  4. **Insertion and extraction operators**

- Unary operators are operators that act upon only single operand.
  Example: ++, - -, -
- Binary operators are operators acting upon two operands
  Example: +, /.%,<,> etc.,
- Operators like new,delete,(,),[,] etc., are special operators.
- >> and << are extraction and insertion operators.

The operations which cannot be overloaded are:

1. **Scope resolution operator (::)**
2. **Ternary operator(?:)**
3. **Size of operator**
4. **Member access operator(.)**
5. **Indirection operator(.*)**

PROGRAM FOR UNARY OPERATOR OVERLOADING

```
#include<iostream>
using namespace std;
class unary
{
int a,b;
public:
void get();
void display();
void operator -();
};

void unary::get()
{
cout<<"enter the values of a and b"<<"\n";
cin>>a>>b;
}
void unary:: display()
{
cout<<"a= "<<a<<" ,b= "<<b<<endl;
}

void unary::operator -()
{
a=-a;
b=-b;
}
main()
{
unary u1;
u1.get();
```

```
u1.display();
-u1;
u1.display();
}
```

PROGRAM FOR CREATION OF COMPLEX CLASS WITH OPERATOR OVERLOADING (binary operator overloading)

```
#include<iostream>
using namespace std;
class complex
{
float x,y;
public:

complex()
{
x=0;
y=0;
 }

complex( float r, float i)
{
x=r;
y=i;
}

complex operator +(complex);

void display()
{
cout<<x<<"  "<<y<<endl;
}
```

```
};

complex complex:: operator  +(complex C)
{
complex t;
t.x=x+C.x;
t.y=y+C.y;
return(t);
}



main()
{
complex c1(2.6,3.6),c2(4.6,5.6),c3;
c3=c1+c2;
c1. display();
c2.display();
c3.display();
}
```

RULES FOR OPERATOR OVERLOADING

- Only existing opearators can be overloaded.New opearators can not be created
- the overload opearator must have at least on operand that is of user defined type
- we can not change the basic meaning of an operator i.e we can not use + for subtraction
- overload operators follow the syntax rules of the orginal opearators.they cannot be overriden.
- sizeof,.(membership opearator),. *(pointer to member opearator) ,:: (scope resolution operator),?: (conditional operator) can't be overloaded.
- =(assignment operator), ()(function call operator),[] (subscripting operator),-> (class member access operator) can't be overloaded using friend functions.

- Unary operators,overloaded by means of a member function take no explicit arguments and return no explicit values,but, those overloaded by means of a friend function, take one referenece argument(the object of the relvant class).
- Binary operators overloaded through a member function function take one explicit argument and those which are overloaded through a friend function take 2 explicit arguments.
- When using binary operators overloaded through a member function, the left hand operand must be an object of the relvant class.
- Binary arithmetic operator such as +,-,*,and / must explicitly return a value.They must not attempt to change their own arguments.
    - **Operator overloading can be done using friend function also**

Overloading Insertion and extraction operator using friend function.

C++ is able to input and output the built-in data types using the stream extraction operator **>>** and the stream insertion operator **<<**. The stream insertion and stream extraction operators also can be overloaded to perform input and output for user-defined types like an object.

- Here, it is important to make operator overloading function a friend of the class because it would be called without creating an object.
- Following example explains how extraction operator **>>** and insertion operator **<<**.

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

class Distance
{
   private:
      int feet;             // 0 to infinite
      int inches;           // 0 to 12
   public:
      // required constructors
      Distance(){
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i){
         feet = f;
         inches = i;
      }
      friend ostream &operator<<( ostream &output,
```

```
                                        const Distance &D )
        {
           output << "F : " << D.feet << " I : " << D.inches;
           return output;
        }

        friend istream &operator>>( istream  &input, Distance &D )
        {
           input >> D.feet >> D.inches;
           return input;
        }
};
int main()
{
   Distance D1(11, 10), D2(5, 11), D3;

   cout << "Enter the value of object : " << endl;
   cin >> D3;
   cout << "First Distance : " << D1 << endl;
   cout << "Second Distance :" << D2 << endl;
   cout << "Third Distance :" << D3 << endl;


   return 0;
}
```