Operator Overloading

Branch: S6 ECE

Faculty: SREEDIVYA

- Operator overloading is a compile-time polymorphism
- Operator overloading is used to overload or redefine most of the operators available in C++.
- It is used to perform the operation on the user-defined data type.
- For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.
- The advantage of Operators overloading is to perform different operations on the same operand.

Operator that can be overloaded are as follows:

- Unary operators
- Binary operators
- Special operators ([ ], (), etc.)

Operators that can be overloaded	Examples
Binary Arithmetic	+, -, *, /, %
Unary Arithmetic	+, -, ++, —
Assignment	=, +=,*=, /=,-=, %=
Bitwise	& , | , << , >> , ~ , ^
De-referencing	(->)
Dynamic memory allocation, De-allocation	New, delete
Subscript	[ ]
Function call	()
Logical	&, | |, !
Relational	>, < , = =, <=, >=

Operator that cannot be overloaded are as follows:

- Scope operator (::)
- Sizeof()
- member selector(.)
- member pointer selector(*)
- ternary operator(?:)

Syntax

```
return_type class_name  : : operator op(argument_list)
{
    // body of the function.
}
```

operator op is an operator function where op is the operator being overloaded, and the operator is the keyword.

Rules for Operator Overloading

- Existing operators can only be overloaded
- Overloaded operator contains atleast one operand of user-defined data type.
- We cannot use friend function to overload certain operators.
- Member function can be used to overload those operators.
- When unary operators are overloaded through a member function they take no explicit arguments, but, if they are overloaded by a friend function, it takes one argument.
- When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

Overloading Unary Operators

Example: C++ program for unary minus (-) operator overloading

```
#include<iostream>
using namespace std;
class NUM
{
   private:
      int n;
   public:
      void getNum(int x)
      {
         n=x;
      }
```

```
void dispNum(void)
      {
         cout << "value of n is: " << n;
      }
void operator - (void)
      {
         n=-n;
      }
};
void main()
{
   NUM num;
   num.getNum(10);
   -num;
   num.dispNum();
   cout << endl;
}
```

OUTPUT

Value of n is -10

Overloading Binary Operators

Example : Program to find sum of complex number

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

class Complex{
   float x, y;
   public:
   Complex(){}
   Complex(float real, float imag)
   {
      x= real;
      y= imag;
   }
   Complex operator+(complex);
   void display(void);
};
```

```
complex complex:: operator+(complex c)
{
        complex temp;
        temp.x = x + c.x;
        temp.y = y + c.y;
        return temp;
}
void complex::display(void)
{
cout<<x<<“+i”<<y<<“\n”;
}
```

```
int main()
{
Complex C1,C2,C3;
C1 = complex(2.5,3.5);
C2 = complex(1.6,2.7);
C3 = C1+C2;
cout<<“C1 =  ”; C1.display();
cout<<“C2 =  ”; C2.display();
cout<<“C3 =  ”; C3.display();
return 0;
}
```

OUTPUT

C1 = 2.5 + i3.5

C2 = 1.6 + i2.7

C3 = 4.1 + i6.2

Another Example Program to find sum of complex number

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

class ComplexNumber {
private:
   int real;
   int imaginary;

public:
   ComplexNumber(int real, int imaginary)
   {
      this->real = real;
      this->imaginary = imaginary;
   }
   void print() { cout << real << " + i" << imaginary; }
   ComplexNumber operator+(ComplexNumber c2)
   {
      ComplexNumber c3(0, 0);
      c3.real = this->real + c2.real;
      c3.imaginary = this->imaginary + c2.imaginary;
      return c3;
   }
};
```

```
int main()
{
    ComplexNumber c1(3, 5);
    ComplexNumber c2(2, 4);
    ComplexNumber c3 = c1 + c2;
    c3.print();
    return 0;
}
```

Output

5 + i9

Operator Overloading using Friend function

```
#include <iostream>
using namespace std;
class A
{
int a;
public:
void set_a();
void get_a();
friend A operator -(A);          // Friend function which takes an object of A and return an object of A type.
};
void A :: set_a()
{
a = 10;
}
void A :: get_a()
{
cout<< a <<"\n";
}
```

```
A operator -(A ob)
{
ob.a = -(ob.a);
return ob;
}
int main()
{
A ob;
ob.set_a();
cout<<"The value of a is : ";
ob.get_a();
ob = -ob;
cout<<"The value of a after calling operator overloading friend function - is : ";
ob.get_a();
}
```

OUTPUT
The value of a is : 10
The value of a after calling operator overloading friend function - is : -10O