Section 13 : Operator Overloading

15 April 2022 16:49

© Rajat Kumar
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Section 13 : Operator Overloading

Operator Overloading :

- For our own data types, that is user define data types (class). We can overload operators. So there are various operators that can overload in C++ except few of them.
- +, *, - (,), ++, new, delete, elc.

```
main()
{
    Complex c1(3,7)
    Complex c2(5,4)
    Complex c3;

    c3=c1.add(c2);
    c3=c2.add(c1);
```

```
class Complex
{
  private:
    int real;
    int img;
  public:
    Complex(int r=0,int i=0)
    {
      real=r;
      img=i;
    }
    Complex add(Complex x)
    {
      complex temp;
      temp.real=real+x.real;
      temp.img=img+x.img;
      return temp;
    }
```

- Operator Overloading, by changing the name of the function from add to **operator+**, now we can direct call c3 = c1 + c2;

```
c3=c1.add(c2);
c3=c1.operator+(c2);
c3=c1+c2
```

Both are same

```
public:
  Complex(int r=0,int i=0)
  {
    real=r;
    img=i;
  }
  Complex operator+(Complex x)
  {
    complex temp;
    temp.real=real+x.real;
    temp.img=img+x.img;
    return temp;
  }
```

Friend Operator Overloading :

- There is one more method for overloading an operator that is using friend function.
- We just declare friend function in class, like :
  - friend Complex operator+(Complex c1, Complex c2);
- It doesn't belong to a class but is a friend of a class. So we don't use any scope resolution operate
- The same function is implemented outside the class without using scope resolution.

```
class Complex
{
    private:
        int real;
        int img;
    public:
friend Complex operator+(Complex c1,complex c2)
};

Complex operator +(Complex c1,Complex c2)
{
  Complex t;
  t.real= c1.real +c2.real;
  t.img = c1.img +c2.img;
  return t;
}
```

- Some operators we can load as member functions as well as you we can overload them as friend functions.
- C3 = c1 + c2 is same as c3 = operator+(c1,c2);

Insertion Operator Overloading :

- how the overload output stream operates. That is ostream operator.
- We use cout and cin for displaying some values on the screen and reading some data from the keyboard, these operators also we can overload. That is insertion and extraction operator.
- Syntax : ostream& operator<<(ostream &o, Complex &c)
    - The Operator function is taking 2 parameters from two different types of objects, so it cannot belong to complex number class. So we have to make it as a friend.
    - Friend ostream& operator<<(ostream &o, Complex &c);

```
,7)          class Complex
             {
                 private:
                     int real;
                     int img;
                 public:
friend ostream & operator<<(ostream &o,Complex &c1)
        };

ostream & operator<<(ostream &o,complex &c1)
{
        o<<c1.real<<"+i"<<c1.img;
        return o;
}
```

- Insertion << as well as extraction >> operators can be overloaded by implementing friend functions.

```
        void display() {
            cout<<real<<" + i"<<img<<endl;
        }

        friend ostream & operator<<(ostream &out, Complex &c);

};

ostream & operator<<(ostream &out, Complex &c) {
    out<<c.real<<" + i"<<c.img<<endl;
    return out;
}
```

```
        void display() {
            cout<<real<<" + i"<<img<<endl;
        }

        friend ostream & operator<<(ostream &out, Complex &c);

};

ostream & operator<<(ostream &out, Complex &c) {
    out<<c.real<<" + i"<<c.img<<endl;
    return out;
}


int main()
{
    Complex c1(5, 9);
    c1.display();
    operator<<(cout, c1);
    cout<<c1;


    return 0;
}
```

Both are same

- Operators can be overloaded using Friend and member function.
- Assignment and Type cast operators can also be overloaded.
- Scope resolution operator cannot be overloaded.