OOP Course

Created by : Israa Abdelghany
LinkedIn : www.linkedin.com/in/israa-abdelghany-4872b0222
GitHub : https://github.com/IsraaAbdelghany9

Lecture 4

Online

- the default copy constructor creates shallow copy
- friend function can access private members
- friend & stand alone function violates the rules of OOP
- friend name of stand alone function

1-send object by value to function
2-return object by value
3- stack s1(s2);
4- Stack s2=s1;
// complex copy ctor can not be implemented
// stack static allocated array we can not implement copy ctor
// stack : dynamic allocated array shallow copy bitwise copy
we must implement copy ctor

operator + ⟹ overload // Complex + Complex

```
        img= _img;
    }
    //complex.opertaor+(complex)
    // complex + complex
    Complex operator+ (Complex c)
    {
        Complex res;
        res.real=real+c.real;
        res.img=img+c.img;
        return res;
    }
};
int main()
```

if I added real= 9000; will change c1

```
int main()
{
    Complex c1(10,20);

    Complex c2(5,6);
    Complex c4(8,9);
    Complex c3;
    //c3=c1.operator+(c2);
    c3=c1+c2;
    c3.print();

    return 0;
}
```

I can use + operator now

c1 + c2

c1 caller object

By value

```
    }
    //complex.operator+(complex)
    // complex + complex
    //c1+c2
    Complex operator+ (Complex& c)
    {
        Complex res;
        res.real=real+c.real;
        res.img=img+c.img;
        return res;
    }
};
```

const (better to add it)

```
Complex::Complex()
Complex::Complex(int _real, int _img)
class Complex {...}
```

this will not call copy Const (preffered)

```
// complex + complex
//c1+c2
Complex operator+ (Complex& c)
{
    Complex res(real+c.real,img+c.img);
    //res.real=real+c.real;
    //res.img=img+c.img;
    return res;
}
// complex +int
//c1+3
Complex operator+(int x){
    Complex res;
    res.real=real+x;
    res.img=img;
    return res;
}
```

اختصار للكود

if 3+C1 → Compilation error

```
54
55  friend Complex operator+(int x, Complex c);
56
57  };
58
59  Complex operator+(int x, Complex c)
60  {
66  int main()
67  {
68      Complex c1(3,4);
69      Complex c2(30,4);
70      if(c1==c2)
71          cout<<"equal\n";
72      else
73          cout<<"not equal\n";
74
75      Complex c4(8,9);
```

binary will print not equal

```
    int operator == (Complex c){
        return real==c.real && img==c.img;
    }
    int operator!=(Complex c){
        return !(*this==c);
    }

friend Complex operator+(int x, Complex c);

};
```

→ Solution

→ Implementation of == & !=

```
Complex operator +=(Complex c){
    real=real+c.real;
    img=img+c.img;
    return *this;
}
```

→ Implementation of +=

c6 = c3+=c1 this type need return

Unary operators

```
Complex operator++()//prefix
{
    real=real+1;
    return *this;
}
```

→ Implementation of ++ pre fix

```
Complex operator++(int)//postfix
{
    Complex temp=*this; // temp 3,5
    real=real+1; c1 ,real=4,5
    return temp;//3,5
}
```

→ Implementation of ++ post fix

```
4+5J
3+5J
4+5J07+5J
Process returned 0 (0x0)   execution time : 3.889 s
Press any key to continue.
```

Rules of operator overload.

```
}
explicit operator  int(){
    return real;
}
```

→ Cast

```
x=(int)c1;
```

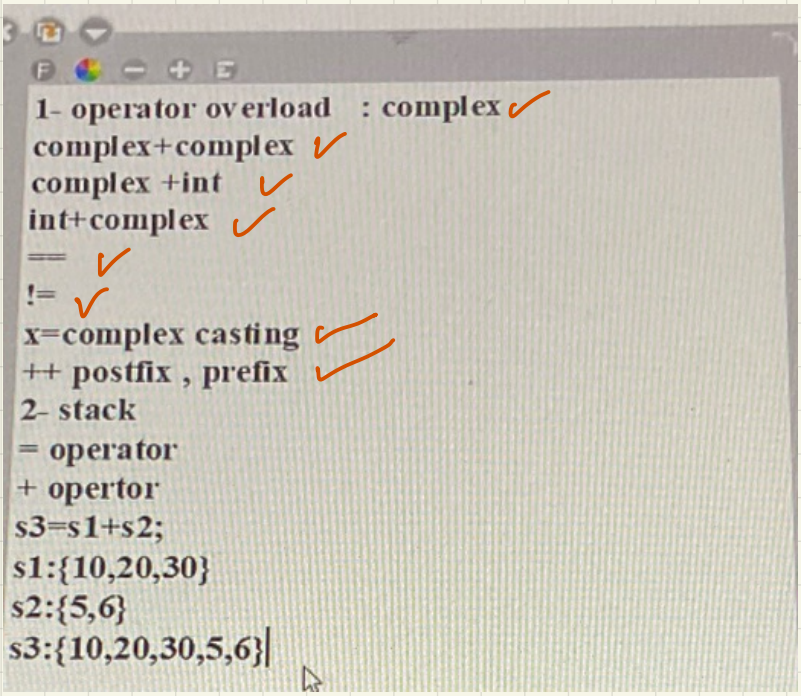
```
Complex operator=(Complex c){
    real=c.real;
    img=c.img;
    return *this;
}
```

```
    cout<<endl;

    Complex c6;
    c1.print();
    c6.print();

    Complex c4(8,9);
    //Complex c3;
    //c3=c1.operator+(c2);
   // c3=c1+c2;
    c3=c1+3;
    c3=3+c1;
    c3.print();
    // complex + complex member function , sta
    //complex + int member function , stand-al
    //int + complex , stand alone function
```

```
Complex c1(3,5);
Complex c2(30,4);
Complex c3;
c3=c2=c1;
c2.print();
cout<<endl;
//c2=c1++;
int x;
x=(int)c1;
c1=x;// constructor casting
cou int main::x
cout<<endl;
c1++.print();
c1.print();
cout<<endl;
c2.print();
cout<<endl;
```

```
Complex (int real)
{
    real = -real ;
    img = 0 ;
}
```

LAB



3-fraction number
num/den data, attributes
ctor , paramless, one param , two param
setters, getters
print
fraction+fraction