Lecture-27

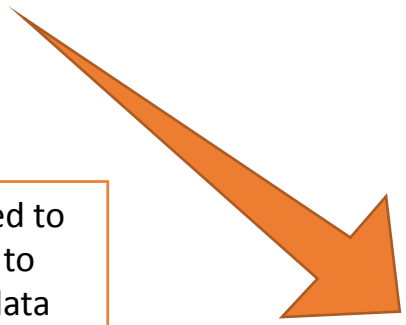
Class – operator overloading

Sum two complex numbers

```
1   #include <iostream>
2   #include "complex.h"
3   using namespace std;
4
5   int main(){
6
7       complex c1(1.2, 3.2);
8       complex c2 (-2.1,4.5);
9       complex c3;
10      c3 = sum(c1,c2);
11  }
12
```

Instead of a function call sum(c1,c2), we would like to do sum naturally as c1 + c2 and assign the result to c3.

To accomplish this we need to overload the operator (+) to handle our user defined data type complex.



```
1   #include <iostream>
2   #include "complex.h"
3   using namespace std;
4
5   int main(){
6
7       complex c1(1.2, 3.2);
8       complex c2 (-2.1,4.5);
9       complex c3;
10      c3 = c1 + c2;
11  }
12
```

class – operator overloading

```
41  complex operator+(complex& c1, complex& c2){
42      complex result;
43      result.Re = c1.Re + c2.Re;
44      result.Im = c1.Im + c2.Im;
45      return result;
46  }
```

Write **a global function** named *operator* followed by symbol of the operator. Pass the two complex numbers as arguments and return the resulting complex number by value.

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3   #include <iostream>
4   using namespace std;
5
6   class complex {
7       private:
8           double Re, Im;
9           static int counter;
10      public:
11          complex();                  // Constructor function
12          complex(double, double);    // Overloading constructor
13          ~complex();                 // Destructor function
14
15          static int getCounterValue();
16          friend complex sum (complex& c1, complex& c2);
17          friend complex operator+(complex& c1, complex& c2);
18  };
19
20  #endif // COMPLEX_H
```

Add the function declaration to the class as *friend*. The output is same as before. Here the statement *c3 = c1 + c2* is interpreted as *c3 = operator+(c1, c2)*.

class – operator overloading rules

- The precedence of an operator can not be changed.
- The associativity of an operator can not be changed.
- The arity (binary or unary) of the operator can not be changed.
- No new operators can be defined.
- Meaning of operator remains same.
- Each operator must be overloaded separately.
- Operator overloading can be done using class member and non-member functions
  - (), [], -> and any assignment operator must be overloaded as member function only.
- Member access (.), pointer member access (.*), scope resolution (: :) and ternary (?:) operators can not be overloaded.

class – operator overloading

```
49  complex complex::operator+(complex& c1){
50      complex result;
51      result.Re = this->Re + c1.Re;
52      result.Im = this->Im + c1.Im;
53      return result;
54  }
```

Write a **member function** named *operator* followed by symbol of the operator. Pass the second complex number as argument and return the resulting complex number by value.

We have used the *this* pointer which the compiler implicitly creates for each object. *This* pointer points at the address of the calling object.

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3   #include <iostream>
4   using namespace std;
5
6   class complex {
7       private:
8           double Re, Im;
9           static int counter;
10      public:
11          complex();                      // Constructor function
12          complex(double, double);        // Overloading constructor
13          ~complex();                     // Destructor function
14
15          static int getCounterValue();
16          friend complex sum (complex& c1, complex& c2);
17          //friend complex operator+(complex& c1, complex& c2);
18          complex operator+(complex& c1);
19  };
20
21  #endif // COMPLEX_H
22
```

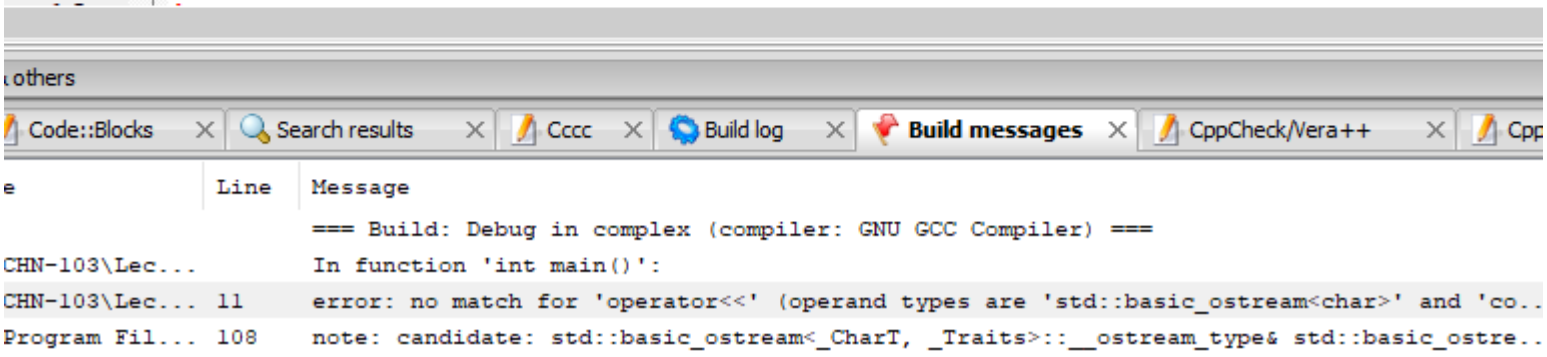
Add the function declaration to the class. The output is same as before. Here the statement *c3 = c1 + c2* is interpreted as *c3 = c1.operator+(c2)*.

Calling object is one on the left of operator

class – operator overloading

```
1   #include <iostream>
2   #include "complex.h"
3   using namespace std;
4
5   int main(){
6
7       complex c1(1.2, 3.2);
8       complex c2 (-2.1,4.5);
9       complex c3;
10      c3 = c1 + c2;
11      cout << "Result is " << c3 << endl;
```

Now we would like to output the complex numbers in the format (Re) + i (Im) by just streaming the object to cout object.



Error is produced because the *operator<<* does not know how to handle the object on the right. We have to enable it by overloading the operator. The *cout* object should appear on left of operator and our user defined should appear on right. Therefore, it has to be done using non-member function only.

class – operator overloading

```
56  ostream &operator<<(ostream &out, const complex& c){
57      out << "(" << c.Re << ") +i (" << c.Im << ")";
58      return out;
59  }
```

Write **a global function** named *operator<<*. Pass the *ostream&* object and *complex&* object as arguments and return *ostream&* as output. Returning the reference to same *ostream* object enable ***cascading*** of stream.

```
1   #ifndef COMPLEX_H
2   #define COMPLEX_H
3   #include <iostream>
4   using namespace std;
5
6   class complex {
7       private:
8           double Re, Im;
9           static int counter;
10      public:
11          complex();                      // Constructor function
12          complex(double, double);        // Overloading constructor
13          ~complex();                     // Destructor function
14
15          static int getCounterValue();
16          friend complex sum (complex& c1, complex& c2);
17          //friend complex operator+(complex& c1, complex& c2);
18          complex operator+(complex& c1);
19          friend ostream &operator<<(ostream &out, const complex& c);
20  };
21
22  #endif // COMPLEX_H
```

The global function is added to class definition as a *friend* to enable accessing the private data members.

class – operator overloading

```
1   #include <iostream>
2   #include "complex.h"
3   using namespace std;
4
5   int main(){
6
7       complex c1(1.2, 3.2);
8       complex c2;
9       cout << "Enter the second complex no. as Re +i Im: ";
10      cin >> c2;
11      complex c3;
12      c3 = c1 + c2;
13      cout << "Result is " << c3 << endl;
14  }
15
```

Now we would like to input the complex numbers in the format Re +i Im by just streaming input stream from cin to the object.

```
60  istream &operator>>(istream &in, complex& c){
61      in >> c.Re;
62      in.ignore(4); // ignore space, +, i, space
63      in >> c.Im;
64      return in;
65  }
```

Write **a global function** named *operator>>*. Pass the *istream&* object and *complex&* object as arguments and return *istream&* as output. Returning the reference to same *istream* object enable ***cascading*** of stream.

class – operator overloading