OOP/COMPUTER PROGRAMMING

By: Dr. Danish Shehzad

OPERATOR OVERLOADING

- Consider the following class:

```
class Complex{
private:
   double real, img;
public:
   Complex Add(Complex C);
   Complex Subtract(Complex C);
   Complex Multiply(Complex C);
   …
}
```

OPERATOR OVERLOADING

- Function implementation:

```
Complex Complex::Add(
        Complex c){
   Complex t;
   t.real = real + c.real;
   t.img  = img  + c.img;
   return t;
}
```

OPERATOR OVERLOADING

- The following statement:

```
Complex c3 = c1.Add(c2);
```

Adds the contents of **`c2`** to **`c1`** and assigns it to **`c3`** (copy constructor)

OPERATOR OVERLOADING

- To perform operations in a single mathematical statement e.g:

```
c1+c2+c3+c4
```

- We have to explicitly write:

```
c1.Add(c2.Add(c3.Add(c4)))
```

OPERATOR OVERLOADING

- Alternative way is:

```
t1 = c3.Add(c4);

t2 = c2.Add(t1);

t3 = c1.Add(t2);
```

OPERATOR OVERLOADING

- If the mathematical expression is big:
  - Converting it to C++ code will involve complicated mixture of function calls
  - Less readable
  - Chances of human mistakes are very high
  - Code produced is very hard to maintain

OPERATOR OVERLOADING

- C++ provides a very elegant solution:

  "Operator overloading"

- C++ allows you to overload common operators like **+**, **–** or ***** etc…

- Mathematical statements don't have to be explicitly converted into function calls

OPERATOR OVERLOADING

- Assume that operator **`+ has`** been overloaded
- Actual C++ code becomes:

```
c1+c2+c3+c4
```

- The resultant code is very easy to read, write and maintain

BINARY OPERATORS

- General syntax:

  Member function:

```
TYPE1 CLASS::operator B_OP(
                          TYPE2 rhs){
    ...
}
```

BINARY OPERATORS

- General syntax:

  Non-member function:

```
TYPE1 operator B_OP(TYPE2 lhs,
                    TYPE3 rhs){
    ...
}
```

BINARY OPERATORS

- The "`operator OP`" must have at least one formal parameter of type class (user defined type)
- Following is an error:

```
int operator + (int, int);
```

BINARY OPERATORS

```
void Complex::operator+(
        Complex rhs){

        real = real + rhs.real;

        img = img + rhs.img;

};
```

BINARY OPERATORS

- Drawback of void return type:
  - Assignments are not possible
  - Code is less readable
  - Debugging is tough
  - Code is very hard to maintain

BINARY OPERATORS

```
void Complex::operator+(Complex
rhs){
      real = real + rhs.real;
      img = img + rhs.img;
};
```

BINARY OPERATORS

- we have to do the same operation `c1+c2+c3` as:

```
c1+c2

c1+c3

// final result is stored in c1
```

BINARY OPERATORS

- Overloading + operator:

```
class Complex{
private:
   double real, img;
public:
   …
   Complex operator +(
        Complex rhs);
};
```

BINARY OPERATORS

```
Complex Complex::operator +(
Complex rhs){
   Complex t;
   t.real = real + rhs.real;
   t.img = img + rhs.img;
   return t;
}
```

BINARY OPERATORS

- The return type is Complex so as to facilitate complex statements like:

```
Complex t = c1 + c2 + c3;
```

- The above statement is automatically converted by the compiler into appropriate function calls:

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

BINARY OPERATORS

- The binary operator is always called with reference to the left hand argument
- Example:
  - In `c1+c2, c1.operator+(c2)`
  - In `c2+c1, c2.operator+(c1)`

BINARY OPERATORS

- The above examples don't handle the following situation:

```
Complex c1;

c1 + 2.325
```

- To do this, we have to modify the **`Complex`** class

BINARY OPERATORS

- Modifying the complex class:

```
class Complex{

   ...

   Complex operator+(Complex rhs);

   Complex operator+(double rhs);

};
```

BINARY OPERATORS

```
Complex operator + (double rhs){
   Complex t;
   t.real = real + rhs;
   t.img = img;
   return t;
}
```

BINARY OPERATORS

- Now suppose:

```
Complex c2, c3;
```

- We can do the following:

```
Complex c1 = c2 + c3;
```

and

```
Complex c4 = c2 + 235.01;
```

BINARY OPERATORS

- But problem arises if we do the following:

```
Complex c5 = 450.120 + c1;
```

- The + operator is called with reference to `450.120`
- No predefined overloaded + operator is there that takes **`Complex`** as an argument

BINARY OPERATORS

- Now if we write the following two functions to the class, we can add a **Complex** to a **real** or vice versa:

```
Class Complex{

    …

    friend Complex operator + (Complex
    lhs, double rhs);

    friend Complex operator + (double
    lhs, Complex rhs);

}
```

BINARY OPERATORS

```
Complex operator +(Complex lhs,double
rhs){

      Complex t;
      t.real = lhs.real + rhs;
      t.img = lhs.img;
      return t;
}
```

BINARY OPERATORS

```
Complex operator + (double lhs,Complex
rhs)
{

      Complex t;
      t.real = lhs + rhs.real;
      t.img = rhs.img;
      return t;
}
```

BINARY OPERATORS

- Other binary operators are overloaded very similar to the + operator as demonstrated in the above examples
- Example:

```
Complex operator * (Complex
   c1, Complex c2);

Complex operator / (Complex
   c1, Complex c2);

Complex operator - (Complex
   c1, Complex c2);
```