Operator Overloading

Sujoy Sinha Roy
School of Computer Science
University of Birmingham

```
int a = 5;
int b = 3;
int c;
```

In C/C++ we can declare and initialize objects of built-in data types.

```
Complex a(5.0, 2.1);
Complex b(3.0, 4.5);
Complex c;
```

C++ constructors enable initialization of objects of user-defined types.

```
int a = 5;
int b = 3;
int c;
c = a+b;
```

We can use '+' operator on built-in data objects to compute sum

```
Complex a(5.0, 2.1);
Complex b(3.0, 4.5);
Complex c;
```

Can we apply operators in a similar manner to objects of user-defined data types?
Example: c = a + b

```
int a = 5;
int b = 3;
int c;
c = a+b;
```

We can use '+' operator on built-in data objects to compute sum

```
Complex a(5.0, 2.1);
Complex b(3.0, 4.5);
Complex c;
```

Can we apply operators in a similar manner to objects of user-defined data types?
Example: c = a + b

C++ enables this feature using 'Operator overloading'.

Operator overloading

- C++ has ability to provide operators with **special meanings** for a user-defined data type.

- This ability is known as operator overloading

- Example: With an overload '+' operator we can add two complex numbers.

```
Complex a(5.0, 2.1);
Complex b(3.0, 4.5);
Complex c;
c = a+b;
```

Defining operator overloading

- A special function called ‘operator function’ is used to specify what the operator means to a chosen class.

- The syntax for declaring an operator ‘op’ for a class is:

```
T Classname :: operator op (argument list){
            // body of operator functions
}
```

where T is the return type of the operator function.

- Example: ‘op’ can be +, -, *, etc.

Example: overloading '+' for complex addition

```
class Complex {
  ...
  public:
  // Overloading + for addition of complex numbers
  Complex operator+(Complex b);
  ...
};
Complex Complex::operator +(Complex b) {
  double real = this->re + b.re;
  double imag = this->im + b.im;
  return Complex(real, imag);
}
int main(){
  Complex a(5.0, 6.0), b(-3.0, 4.0);
  Complex c;
  c = a + b;     // Adding complex numbers using +
  ...
}
```

The way operator overloading works in C++

When we use operator + to add complex numbers ‘a’ and ‘b’:

```
c = a + b;
```

internally object ‘a’ invokes the ‘**operator+()**’ function and passes object ‘b’ as argument

```
c = a.operator+(b);
```

```
Complex Complex::operator +(Complex b) {
  double real = this->re + b.re;
  double imag = this->im + b.im;
  return Complex(real, imag);
}
```

Overloading unary operator

- Similarly, we can overload unary operator
- Example: negation of a complex number

```
class Complex {
  ...
  public:
  // Overloading unary - for negation of a complex number
  Complex operator-();
  ...
};
Complex Complex::operator -() {
  double real = -this->re;
  double imag = -this->im;
  return Complex(real, imag);
}
int main(){
  Complex a(5.0, 6.0);
  Complex c;
  c = -a;     // c is negation of a
  ...
}
```

```
int a = 5;
int c;
c = a + 3.5;
```

Operation on two different data types

Can we do similar for class objects?

```
Complex a(5.0, 2.1);
Complex c;
c = a + 5.3;
```

complex

float

Operation on two different data types

```
int a = 5;
int c;
c = a + 3.5;
```

Operation on two different data types

Solution: create another overloaded operator

```
Complex a(5.0, 2.1);
Complex c;
c = a + 5.3;
```

complex

float

Operation on two different data types

```
Complex Complex::operator +(float f) {
   double real = this->re + f;
   double imag = this->im;
   return Complex(real, imag);
}
```

```
int a = 5;
int c;
c = 3.5 + a;
```

Operation on two different data types

Can we do similar for class objects?

```
Complex a(5.0, 2.1);
Complex c;
c = 5.3 + a;
```

Challenge: The first argument 5.3 is not an object of the class Complex. So, it cannot call any member function of Complex.

Operator as a 'friend' function

- **friend function:** it is defined outside the scope of a class, but it has the right to access all private and protected members of the class.

```
class Complex {
   private:
   double re;   // the real part
   double im;   // the imaginary part

    public:
     ...
    friend Complex operator+(float f, Complex c);
};

// A friend function is defined like an ordinary function.
// No class resolution operator is used.
Complex operator +(float f, Complex c) {
            double real = f + c.re;
            double imag = c.im;
            return Complex(real, imag);
}
```

```
Complex a(5.0, 2.1);
Complex c;
c = 5.3 + a;
```

Operators that cannot be overloaded

Operators	
sizeof()	Size of operator
.	Membership operator
.*	Pointer to member operator
::	Scope resolution operator
?:	Conditional operator