C++ Function and Operator Overloading

한양대학교 컴퓨터소프트웨어학부

2017년 2학기

C++ Function Overloading

C++ allows defining same-named functions with different parameters.

- You've already seen some examples.
  - Class constructors.
  - Some STL container's member functions.
- Both regular functions and class member functions can be overloaded.

- Exactly same functions except return types are not allowed.
  - At least one parameter is different, or
  - the const-ness of the class should be different.
- Beware of default parameters, and implicit type casting of parameter values.

C++ Function Overloading Example

```cpp
class MyClass {
 public:
  MyClass() : x_(0) {}
  MyClass(const MyClass& c) : x_(c.x_) {}
  explicit MyClass(int x) : x_(x) {}  // Prevent implicit type conversion.

  int& x() { return x_; }  // Differ in constness.
  const int& x() const { return x_; }

  int DoSomething() { return x_ * x_; }
  double DoSomething(double c) { return c * x_; }
  void DoSomething(double a, double b) { x_ = a * x_ + b; }

  void DoSomething(double a, double b, double c = 0.0);  // Error!

 private:
  int x_;
};
```

C++ Operator Overloading

C++ even allows redefining built-in operators.

- Operators are just functions in specific forms.
  http://en.wikipedia.org/wiki/Operators_in_C_and_C++

```
class A {                             // A a0, a1;
  A& operator =(const A& a);          // a0 = a1;
  A operator +(const A& a) const;     // a0 + a1
  A operator +() const;               // +a0
  A& operator +=(const A& a);         // a0 += a1;
  A& operator ++();                   // ++a0
  A& operator ++(int);                // a0++
};

A operator +(const A& a0, const A& a1);  // a0 + a1
A operator +(const A& a0);               // +a0
A& operator +=(A& a0, const A& a1);      // a0 += a1;
A& operator ++(A& a0);                   // ++a0
A& operator ++(A& a0, int);              // a0++

std::ostream& operator <<(std::ostream& out, const A& a);   // cout << a0;
```

C++ Operator Overloading

C++ even allows redefining built-in operators.

- Most commonly overloaded operators are
  - Arithmetic operators : +, -, *, / …
  - Assignment operators : =, +=, -=, *= …
  - Comparison operators : <, >, <=, >=, ==, != …
  - For array or containers : [], () …
  - Rarely : ->, new, delete, , ...
- Operator overloading must be used very carefully, since it can hamper the readability seriously.

C++ Operator Overloading Example

```
class Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& Copy(const Complex& c) {
    real = c.real, imag = c.imag;
    return *this;
  }
  Complex Add(const Complex& c) const {
    return Complex(real + c.real, imag + c.imag);
  }

 private:
  double real, imag;
};

void Test() {
  Complex a(1.0, 2.0), b(2.0, 5.0);
  Complex c(a.Add(b));
  c.Copy(c.Add(a));
}
```

C++ Operator Overloading Example

```
class Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& operator=(const Complex& c) {
    real = c.real, imag = c.imag;
    return *this;  // This enables c = a = b;
  }
  Complex operator+(const Complex& c) const {
    return Complex(real + c.real, imag + c.imag);
  }

 private:
  double real, imag;
};

void Test() {
  Complex a(1.0, 2.0), b(2.0, 5.0);
  Complex c(a + b);
  c = c + a;
}
```

C++ Operator Overloading

- How can we make the following code to work?

```cpp
class Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& operator=(const Complex& c);
  Complex operator+(const Complex& c) const;

 private:
  double real, imag;
};

void Test() {
  Complex a(1.0, 2.0), b(2.0, 5.0), c;
  c = a + b;    // OK.
  c = a + 3.0;  // Error.
  c = 2.0 + b;  // Error.
}
```

  - Define operators for all possible parameters?

C++ Operator Overloading

- We can use implicit type conversion.

```
class Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double v) : real(v), imag(0.0) {}  // Constructor for a single v.
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& operator=(const Complex& c);
  Complex operator+(const Complex& c) const;

 private:
  double real, imag;
};

void Test() {
  Complex a(1.0, 2.0), b(2.0, 5.0), c;
  c = a + b;    // OK.
  c = a + 3.0;  // OK.
  c = 2.0 + b;  // Error.
}
```

  - Why the last line of the code still does not work?

C++ Operator Overloading

- Make the operator a non-member function.

```
class Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double v) : real(v), imag(0.0) {}  // Constructor for a single v.
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& operator=(const Complex& c);

 private:
  double real, imag;
};

Complex operator+(const Complex& lhs, const Complex& rhs) {
  return Complex(lhs.real + rhs.real, lhs.imag + rhs.imag);
}

void Test() {
  Complex a(1.0, 2.0), b(2.0, 5.0), c;
  c = a + b;    // OK.
  c = a + 3.0;  // Error.
  c = 2.0 + b;  // Error, but a different kind.
}
```

C++ Operator Overloading

- Make it to be a ‘friend’ to the class.

```
class Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double v) : real(v), imag(0.0) {}  // Constructor for a single v.
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& operator=(const Complex& c);

 private:
  double real, imag;
  friend Complex operator+(const Complex& lhs, const Complex& rhs);
};

Complex operator+(const Complex& lhs, const Complex& rhs) {
  return Complex(lhs.real + rhs.real, lhs.imag + rhs.imag);
}

void Test() {
  Complex a(1.0, 2.0), b(2.0, 5.0), c;
  c = a + b;    // OK.
  c = a + 3.0;  // OK.
  c = 2.0 + b;  // OK.
}
```

Friend Class and Function

- Functions or classes can be ‘friends’ of other classes.
  - Declare them as friends in the class definition.
  - Friends can access all members including private members.

```
class ClassA {
 private:
  int var_;
  friend ClassB;
  friend void DoSomething(const ClassA& a);
};

class ClassB {
  // ...
  void Function(const ClassA& a) { a.var_ = 0; }  // OK.
};

void DoSomething(const ClassA& a) { cout << a.var_; }  // OK.
```

```
struct Complex {
 public:
  Complex() : real(0.0), imag(0.0) {}
  Complex(double v) : real(v), imag(0.0) {}
  Complex(double r, double i) : real(r), imag(i) {}
  Complex(const Complex& c) : real(c.real), imag(c.imag) {}

  Complex& operator=(const Complex& c) {        // Complex a(1.0, 0.0), c;
    real = c.real, imag = c.imag;               // c = a;
    return *this;
  }

  Complex operator+() const { return *this; }                // c = +a;
  Complex operator-() const { return Complex(-real, -imag); }  // c = -a;

  double& operator[](int i) { return i == 0 ? real : imag; }   // i = c[0];
  const double& operator[](int i) const { return i == 0 ? real : imag; }

 private:
  double real, imag;

  friend Complex operator+(const Complex& lhs, const Complex& rhs);
  friend bool operator<(const Complex& lhs, const Complex& rhs);
};

Complex operator+(const Complex& lhs, const Complex& rhs) const {  // c + a
  return Complex(lhs.real + rhs.real, lhs.imag + rhs.imag);
}

bool operator<(const Complex& lhs, const Complex& rhs) {  // if (c < a)
  return lhs.real < rhs.real && lhs.imag < rhs.imag;
}
```

Other C++ Operators

```
class T {
  // Constructors...

  T operator+() const;              // +t
  T operator-() const;              // -t
  T operator+(const T& a) const;  // t + a (t - a, t * a, ...)

  T& opertor=(const T& a) { /* ... */ return *this; }   // t = a
  T& opertor+=(const T& a) { /* ... */ return *this; }  // t += a (t -= a ...)

  T& operator++();        // Prefix form: ++a
  T& operator++(int);   // Postfix form: a++

  T& operator[](int i);                 // t[i]
  const T& operator[](int i) const;  // t[i]
  T& operator()(int i, int j);                 // t(i, j)
  const T& operator()(int i, int j) const;  // t(i, j)
};

T operator-(const T& a, const T& b);           // a - b
bool operator==(const T& a, const T& b);       // a == b

ostream& operator<<(ostream& os, const T& a);  // cout << a
istream& operator>>(istream& is, T& a) {       // cin >> a
  is >> a.member;
  return is;
}
```

Summary

- Operators are just functions.
- Operators and functions can be overloaded.
- Friends can access the private members.

Q&A