C++: Classes -- Methods, Operators

Reminder
• Exam 1 – next Thursday, Sept 29th
• Details to be given Thursday.

Logistics
• Project
  – Part 1 (clock and design) due Sunday, Sept 25th
  – Start thinking about partners for Parts 2-3
• Questions?

Plan for Today
• C++ Classes
  – Overloading functions/methods
  – Overloading operators

Function overloading
• The same function / method name can be used several times:
  – The argument set and return type must be different for each function definition
  – Overloaded functions cannot differ by return type alone.

Function overloading
```cpp
class Foo {
  public:
    char f();
    char f (int x);
    char f (int x, int y);
    double f (double x, double y);
    int f (int x);   // not allowed
}
```
Function overloading

• Why bother?

```cpp
void print_int (int i);
void print_char (char c);
void print_double (double d);
```

Compared to

```cpp
void print (int i);
void print (char c);
void print (double d);
```

Operator overloading

• All C++ operators can be overloaded on a class by class basis.

• Overloaded operators call specially named class methods.

  - Keyword `operator` followed by operator to be overloaded.
  - E.g.

```cpp
  *operator+
```

Operator overloading

```cpp
class Complex
{
  private:
    double re, im;
  public:
    Complex (double r, double i);
    Complex operator+ (Complex &c) const;
    Complex operator-() const;
    bool operator== (Complex &c) const;
    Complex& operator+= (Complex &c);
    Complex& operator+= (double d);
};
```

Operator overloading

Once overloaded, operators can be used in the same manner as for basic types.

• E.g.

```cpp
Complex c1, c2, c3;
double d=5.0;
c2 = c1 + c3;
c3 = -c1;
c3+=c2;
c3+=d;
if (c3 == c1) { ... }
```

Operator overloading

• What would the definition of an overloaded operator function look like?

```cpp
Complex Complex::operator+ (Complex &c) const
{
  return Complex (re + c.re, im + c.im);
}
```

```cpp
bool Complex::operator== (Complex &c) const
{
  return ((re == c.re) && (im == c.im));
}
```

Operator overloading

• Using overloaded operators is just a shorthand for calling the specially named class methods.

• E.g.

```cpp
c2 = c1 + c3;
  * Is the same as
  c2 = c1.operator+ (c3);
```
Operator overloading

- What would the definition of an overloaded operator function look like?

```
Complex& Complex::operator+= (Complex &c) {
    re += c.re;
    im += c.im;
    return (*this);
}
```

Operator overloading

- It would be nice for all operators to return references…but this is difficult

```
Complex& Complex::operator+( Complex &c ) const {

    // Here memory associated with CC will go away once function completes
    Complex CC(re + c.re, im + c.im);
    return (CC); // Returning ref to a var that will no longer exist.
}
```

Operator overloading

- So when can references be returned?
  - Operators that modify themselves and return references to themselves.
  - Const operators, which just use the values of an object will generally create an pass back a new object.
  - Logical operators should return `bool`.

Operator overloading

- Overloaded operators can also be defined globally as non-members (outside of the class definition)

```
Complex& Complex::operator+( Complex &c ) const {

    // We could try to allocate on the free store
    Complex *CC
        (new Complex (re + c.re, im + c.im));
    return (*CC); // but who’s going to clean this up?
}
```

Operator overloading

- Friends
  - By declaring a function as a friend, we allow it access to a class’s private data members (both data and methods)
Operator overloading

- Global operator definitions
  
  ```
  friend Complex operator+(Complex &c1, Complex &c2);  
  friend Complex operator-(Complex &c1);  
  friend bool operator==(const Complex &c1, const Complex &c2);  
  friend Complex& operator+=(Complex &c1, const Complex &c2);  
  friend Complex& operator+=(Complex &c, double d);  
  ```

- Why use friend?
  - Used for operators that have another class as the left operand
    - E.g. `<<` (as we’ll see in next slide)
  - Permit operators to be commutative.

```
Complex c1, c2;
double d;
c1 = c2 + d;  // Not allowed if member
```

I/O overloaded operators

- Overloading `<<` and `>>`
  ```
  friend ostream& operator<<(ostream& output, const Complex c) {  
    output << c.re << " + " << c.im << " i";  
    return output;  
  }
  ```

```c1 = 1.0, 2.0;  
cout << "My complex number is: " << c1;  
My complex number is: 1.0 + 2.0 i```

Assignment operator

- `operator=`
  - Called when an assignment is made
  - Copies all relevant data from object assigner to assignee.
  - Must be declared as a class member
  - Should check for self-assignment!
Assignment operator

- If no assignment operator is defined for a class, the default assignment operator is used.
  - Member by member copy of data from one object to another.
  - Can be troublesome if class have pointers as data members.

Operators that can be overloaded

- Unary Arithmetic Operators
  - -, +=, -=, *=, /=, ++, --
- Binary Arithmetic operators
  - +, -, /, *, %
- Relational operator (should return bool)
  - <, >, <=, >=, ==
- Assignment
  - =

Operators that can be overloaded

- Subscript (For collection classes)
  - []
- Pointer to member (for pointer classes)
  - ->
- Call operator
  - ()
- Memory management
  - new, delete

Summary

- Function overloading
- Operator overloading

Next time

- Templates and the Standard Template Library