

# Operator Overloading

August 9, 2010

# What's an Operator?

A specially named function defining the logic for C++ operations.

<code>a + b</code>	<code>a - b</code>	<code>a * b</code>	<code>a / b</code>
<code>a++</code>	<code>++a</code>	<code>a--</code>	<code>--a</code>
<code>«</code>	<code>»</code>	<code>a % b</code>	<code>...</code>

# What's Operator Overloading All About?

```
20
21 int main()
22 {
23     // two wallets for two different people
24     Wallet Bob( "Bob" ); // no cash
25     Wallet Alice( "Alice", 50, 32 ); // $50.32
26
27     // Dah-dum-duh-dum
28     // Bob and Alice have a wedding
29     Wallet BobAndAlice = Bob + Alice;
30
31     // Years of honest toil...
32     BobAndAlice = BobAndAlice + 200000;
33
34     // Baby Sally!
35     BobAndAlice = BobAndAlice - 6000;
36
37     // Sally goes to college ...
38     Wallet Sally( "Sally", 23000, 0 ); // $23K
39     BobAndAlice = BobAndAlice - Sally;
40
41     return 0;
42 }
```

# Lecture Question

Answer question 1.

## Two Different Kinds of Operator Implementations

1. Global operators implemented by the non-class author.
2. friend-Global operators implemented by the class author.

(There are actually four, but one is trivial combination of the above, and we don't cover member function operators in CSCI 261.)

## Global Operators

```
fooClass x, y; int z;
```

### In Code

### Function Implementation Header

```
z + x      int operator+( int lhs, const fooClass& rhs )
x*z       fooClass operator*( const fooClass& lhs, int rhs )
cout « x   ostream& operator«( ostream& os, const fooClass& rhs )
cin » y    istream& operator»( istream& is, fooClass& rhs )
objA @ objB  typeTBD operator@( const typeA& lhs, const typeB& rhs )
```

TBD  $\equiv$  To Be Determined (by the programmer!)

Two arguments representing either side of the binary operator.

# Global Operators

```
fooClass x, y; int z;
```

In Code

Function Implementation Header

```
cout « x    ostream& operator«( ostream& os, const fooClass& rhs )  
cin  » y    istream& operator»( istream& is, fooClass& rhs )
```

Why aren't the input and output stream arguments const?

## Operators by a Non-Class Author

path3d.cxx with Operator Overloading

```
10 // an overloaded subtraction operator written by the
11 // APPLICATION DEVELOPER
12 point operator-( const point& lhs , const point& rhs )
13 {
14     // subtraction must be accomplished through the public
15     // interface of the point class...
16     point negrhs = rhs.negate();
17     point difference = lhs.add( negrhs );
18     return difference;
19 }
20
21 // an overloaded > operator written by the
22 // APPLICATION DEVELOPER
23 bool operator>( const point& lhs , const point& rhs )
24 {
25     // the point farthest from the origin is the greater
26     return lhs.length() > rhs.length();
27 }
```

# Operators by a Non-Class Author

## Original path3d.cxx

```

38  /**
39   * add the distance between this point and
40   * its previous neighbor to the perimeter
41   */
42  int prevertex = (i+n-1)%n;
43  perimeter += path[i].add( path[prevertex].negate() ).length();
44  //          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
45  //          -v_{i-1}
46  //          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
47  //          v_i - v_{i-1}
48  //          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
49  //          | v_i - v_{i-1} |

```

## path3d.cxx with Operator Overloading

```

57  /**
58   * add the distance between this point and
59   * its previous neighbor to the perimeter
60   */
61  int prevertex = (i+n-1)%n;
62  point i_minus_prev = path[i] - path[prevertex];
63
64  perimeter += i_minus_prev.length();

```

# Operators by a Non-Class Author

## Original path3d.cxx

```
35 double perimeter(0), maxdist(-1); // gauranteed all lengths >-1
36 string maxname( "" );

51 /**
52  * look for the vertex farthest from the origin (length)
53  */
54 double l = path[i].length();
55 if( l > maxdist ) {
56     maxdist = l;
57     maxname = path[i].getName();
58 }
```

## path3d.cxx with Operator Overloading

```
54 double perimeter(0);
55 point farthest; // default ctor is at origin

66 /**
67  * look for the vertex closest to the origin (length)
68  */
69 if( path[i] > farthest ) {
70     // a new point farthest from the origin
71     farthest = path[i];
72 }
```

## friend-Global Ops by the Class Author

1. Prototyped in the class header file, **in** the class declaration, using the `friend` keyword.
2. Even though they are prototyped in the class declaration, **they are not written in class scope and they have no calling object!**
3. Implemented in the class source file.

# **friend-Global Ops by the Class Author**

[Edit Source](#)

## friend-Global Ops by the Class Author

1. Cool! friend-global operators have privileged access to a class' private data members and member functions.
2. They are not written in class scope (there is no `Classname::` before operator in the function header).
3. There is no calling object (class data members are available only through a function parameter and `."`-member access).

## Everyone Needs a friend

- ▶ May be used only inside of class declarations.
- ▶ Always modifies a non-member (global) function.
- ▶ friend goes before function's return type.
- ▶ friend **is not** used in function definition.
- ▶ A friended function has access to class instance members *as if it were in class scope*.

## Question 3 a–e.

# Operator Implementations

**Global or Top-Level Operators** A global function that defines object manipulations through a class' public interface.

These are written by a programmer that didn't write the class.

**friend Global Operators** A global function with special access to class internals defines the object manipulations.

AKA: "friend functions," "friended operators"

These are written by the class author.

Why would a class author need to write a (non-friend) global operator?

## Question 5.

## Input & Output Suggestions

- ▶ Opt for symmetry in << and >> operators.
  - ▶ Use operator<< and operator>> for **datafile i/o**; perhaps not easily read by humans.
  - ▶ Use operator<< and operator>> should be *symmetric*.  
**They should read and write the same data format.**
  - ▶ **Don't** embed newlines into these output methods.
- ▶ `print( ostream& os )` for human readable output (log files, debug files, printer, ...)

Feel like you've heard all of this before? You have, I mention it again, here, because these are widely considered best programming techniques. So you should hear about them more than once.

## Beware Your operator» Logic!

1. Read values from the input stream **into local, temporary variables**.
2. Return the input stream reference as soon as it enters a `fail()` state!
3. Don't change the class instance's value until **all** parameters have been successfully read from the input stream.

### Edit Source

```
135 // We must be careful here, we don't require
136 // that a point have a non-empty name!
137 string n;
138 double newvalues[ point::COORDS];
139 char closeb;
140 // is >> ws skips all whitespace before getline begins,
141 // very handy
142 if( !getline( is >> ws, n, point::OPENTUPLE ) ) {
143     // strange input failure (perhaps end of file?)
144     return is;
145 }
146 if( !n.length() ) {
147     // even unnamed points would have whitespace...
148     // this must be an error
149     is.setstate( ios::failbit );
150     return is;
151 }
```

`rhs` has its “value” changed only if all its variables are read successfully **and** they make sense with respect to the class definition.

Use `setstate` to signal the main routine of non-sensical data.

**finis**