C++ Operators

Jim Fawcett

CSE687-OnLine

Summer 2017

C++ Binary Operator Model

- A C++ operator is really just a function. Assignment, for example, may be invoked either way shown below:

```
x = y;
or
x.operator=(y);
```

  Here, the x object is invoking the assignment operator on itself, using y for the assigned values.

- The left hand operand is always the invoking object and the right hand operand is always passed to the function as an argument.

- General form of the binary operator:

x@y ⇔ x.operator@(y) - member function

x@y ⇔ operator@(x,y) - global function

Indexing Operators

- Indexing operators should usually come in pairs:

```
val& X::operator[](int n);          x[3] = 'a';

val X::operator[](int n) const;     char ch = x[2];
```

- The second form allows you to pass an indexed object into a function by const reference and still be able to read indexed values.

  With only the first form, any indexing in the function will result in a compile time error since the operator does not guarantee not to change the const object.

Unary Increment/Decrement Operators

This example based on iterators pointing to contiguous memory

```
iterator& operator++()
{ /* ++(this->ptr); return *this */ }

  iterator operator++(int)
{ /* iterator temp = *this, ++(this->ptr), return temp */ }

  iterator& operator--()
{ /* --(this->ptr); return *this */ }

  iterator operator--(int)
{ /* iterator temp = *this; --(this->ptr), return temp */ }
```

Sum Operators

- Arithmetic operators should come in pairs. Addition looks like this:

```
X& X::operator+=(const X &x);
X X::operator+(const X &x);
```

  Addition should be implemented this way:

```
X X::operator+(const X &x) {
   X temp = *this;    // copy of me
   temp += x;       // copy of me + x
   return temp;
}
```

- You implement `operator+=(…)` first, and get `operator+(…)` almost for free.

Overloading Arithmetic Operators

- Define:

  `operator+, operator-, operator*, and operator/`

  in terms of :

  `operator+=, operator-=, operator*=, and operator/=`

- Remember the binary operator model:

```
operators as class members:    x@y ⇔  x.operator@(y)
operators as global functions: x@Y ⇔  operator(x,y)
```

Insertion

- The insertion and extraction operators:

```
ostream& operator(ostream& out, const X &x);
istream& operator(ostream& in, const X &x);
```

  Have to be implemented as global (non-member) functions since they are invoked with the statements:

```
out << x;   and   in >> x;
```

- Since the streams, out and in, appear on the left side of the operator, and are not objects of the X class, we must use the global form shown at the top of this slide.

- You should try to implement them without making them friends of the X class. You may need to implement public helper functions to do that.

End of Presentation