CSCI 104L Lecture 8: Operator Overloading

Can you define both of these functions simultaneously?

```cpp
int minimum(int x, int y);
double minimum(double x, double y);
```

How about these?

```cpp
bool foo(int x);
void foo(int y);
```

Operators such as `==`, `<`, etc, are functions too. They can be overloaded as well.

```cpp
class IntArray {
public:
    bool operator==(const IntArray& otherArray) {
        if (this->size != otherArray.size) return false;
        for (int i = 0; i < size; i++)
            if (this->data[i] != otherArray.data[i]) return false;
        return true;
    }
    int& operator[](int index) {
        return data[index];
    }
    IntArray& operator++ () {
        for (int i = 0; i < size; i++) data[i]++;
        return *this;
    }
private:
    int size;
    int* data;
};
```

We can now write code such as the following:

```cpp
if (firstArray == secondArray) ++firstArray;
secondArray[0] = 0;
```

**Question 1.** Why do you suppose `operator++` has a return type, instead of return `void`?

**Question 2.** Why do you suppose `operator[]` returns by reference?

Let’s look at the implementation of `operator++`.

```cpp
IntArray IntArray::operator++(int dummy) {
    IntArray copy = *this;
    *this++;
    return copy;
}
```
Let's also look at the implementation of `operator*`.

```cpp
IntArray IntArray::operator* ( int multiplier ) {
    IntArray newArray;
    newArray.size = size;
    for ( int i = 0; i < size; i++ ) newArray.data[i] = data[i]*multiplier;
    return newArray;
}
```

**Question 3.** Why did we not return by reference?

**Question 4.** Could we write `7*OurArray`?

Consider this code:

```cpp
IntArray operator* ( int multiplier, const IntArray& oldArray ) {
    IntArray newArray;
    newArray.size = size;
    for ( int i = 0; i < size; i++ ) newArray.data[i] = data[i]*multiplier;
    return newArray;
}
```

**Question 5.** Are there any problems with the above code?

```cpp
friend IntArray operator* ( int multiplier, const IntArray& oldArray );
```

A common thing to overload is the `<<` operator:

```cpp
ostream& operator<< ( ostream& o, const IntArray &a ) {
    for ( int i = 0; i < a.size; i++ ) o << a.data[i] << " ";
    o << endl;
    return o;
}
```

**Question 6.** Why did we return an ostream&?

```cpp
IntArray& IntArray::operator= ( const IntArray &otherArray ) {
    this->size = otherArray.size;
    this->data = new int[this->size];
    for ( int i = 0; i < this->size; i++ ) this->data[i] = otherArray.data[i];
    return *this;
}
```

**Question 7.** Why did we return IntArray&?

**Question 8.** What is missing in the above code?

**Question 9.** Are there any other problems?
CSCI 104L Lecture 8: Complex Number Class

A complex number is \(a + ib\), where \(a\) and \(b\) are real numbers, and \(i\) is such that \(i^2 = -1\).

class Complex {
    private:
        double re, im;
    public:
        Complex (double re, double im) {
            this->re = re;
            this->im = im;
        }
        Complex operator+ (const Complex &other) const {
            double reSum = re+other.re;
            double imSum = im+other.im;
            return Complex(reSum, imSum);
        }
        Complex operator- (const Complex &other) const {
            double reSub = re-other.re;
            double imSub = im-other.im;
            return Complex(reSub, imSub);
        }
        Complex operator* (const Complex &other) const {
            double reMul = re*other.re-im*other.im;
            double imMul = re*other.im+im*other.re;
            return Complex(reMul, imMul);
        }
        boolean operator==(const Complex &other) const {
            return (re == other.re) && (im == other.im);
        }
        boolean operator!=(const Complex &other) const {
            return (*this == other);
        }
    friend ostream& operator<<(ostream &out, const Complex &c);
};
ostream& operator<<(ostream &out, const Complex &c) {
    out << c.re << "+" << c.im << "\cdot i";
    return out;
}