CSCI 104L Lecture 10: Copy Constructors

```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 1.** Why did we not return by reference?

**Question 2.** 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 3.** 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 4.** Why did we return an ostream&?
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 5. Why did we return IntArray&?.

Question 6. What is missing in the above code?

Question 7. Are there any other problems?

Question 8. What happens in the following code if our copy constructor does a shallow copy and the destructor is implemented as indicated?

IntArray::~IntArray () { delete [] data; data = NULL; }
int main () {
    IntArray a1;
    IntArray a2 (a1);
    return 0;
}

In order to avoid this problem, you will need to do a deep copy:

IntArray::IntArray(const IntArray &a) {
    size = a.size;
    data = new int[size];
    for (int i = 0; i < size; i++) data[i] = a.data[i];
}

The Rule of Three states that if you need implement one of the following three functions, then you should implement all three of them:

- Destructor
- Copy Constructor
- Assignment operator