CSCI 104L Lecture 10: Copy Constructors

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. Could we write 7∗OurArray?

Consider this code:

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 2. Are there any problems with the above code?

friend IntArray& operator∗( int multiplier , const IntArray& oldArray);

A common thing to overload is the << operator:

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

Question 3. 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 4. Why did we return IntArray&?

Question 5. What is missing in the above code?

Question 6. Are there any other problems?

Question 7. What happens in the following code if our copy constructor does a shallow copy and the destructor is implemented as indicated?

```cpp
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:

```cpp
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