lab 07: Virtual Classes and Deep Copying

Instructions: In this lab, utilize a List interface to implement our Array class. List.hpp has been provided:

```
#ifndef LIST_HPP
#define LIST_HPP

class List {
public:
    /* Returns the index in the array where value is found.
     * Return -1 if value is not present in the array.
     */
    virtual int indexOf(const int value) = 0;

    /* Removes an item at position index by shifting later elements left.
     * Returns true iff 0 <= index < size.
     */
    virtual bool remove(const int index) = 0;

    /* Insert the integer val at position pos.
     * Shift all values after pos up ("to the right") by one.
     * This means the last element will be shifted out of the array
     * (that is fine.)
     * If pos is beyond the size of the array, increase the size of the array
     * so val can be inserted.
     */
    virtual void insert(const int pos, const int val) = 0;

    /* Retrieves the element at position pos
       Returns -1 if pos is invalid.*/
    virtual int get(const int pos) const = 0;

    /* Sets the element at position pos to the value val.
       Returns -1 if pos < 0.*/
    virtual int set(const int pos, const int val) = 0;

    /* Returns if the two lists contain the same elements in the
     * same order.
     */
    virtual bool equals(const List &list) = 0;
};

#endif
```

Now provide an IntArray class that implements List and utilizes deep copy constructors. Consider the following code:

```
    const int ary[] = {10, 50, 34, 20};
    IntArray *ary1 = new IntArray(ary, 5);
    IntArray *ary2 = new IntArray(*ary1);
    std::cout << "(ary1 == ary2)?" << (ary1 == ary2) << "\n";
    std::cout << "ary1->equals(*ary2)?" << ary1->equals(*ary2) << "\n";
    ary2.set(2, 10);
    std::cout << "ary1->equals(*ary2)?" << ary1->equals(*ary2) << "\n";
    ary2.set(2, 34);
    std::cout << "ary1->equals(*ary2)?" << ary1->equals(*ary2) << "\n";
};
```

Write some test cases:
Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code.

Memory Management:
Now that we are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code!

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Webhook: The webhook is:
http://coins.csuniv.edu:2234/github/build-csci-315-fall-2017.php
Remember, after the first push, please wait 5-10 minutes for the auto-grader to get your repository. Then subsequent pushes should receive a grade.

Due Date: September 25, 2017 2359

Teamwork: No teamwork, your work must be your own.