COINS 315 Project 2

**Ordered Multi-List, again**

Introduction

This project involves reimplementing OrderedMulti-List, your first ADT. The internal data structure of this ADT **must** be a singly-linked, linear list. NO arrays allowed.

Interface

You must implement the following methods EXACTLY as described EXCEPT that you **MUST** declare methods to be const when appropriate. The method signatures must be the same as given below (same as in Project 1 except for const). There is no DEFAULT\_MAX\_ITEMS since there is no array.

class OrderedMultiList {

public:

//Constructors/destructors, no copy constructor/= required

OrderedMultiList(); //creates an empty list

~OrderedMultiList(); //destructor will be needed

//accessors

bool empty() ; //returns true iff (if and only if) list is empty

int uniqueSize() ; //returns the number of distinct items in the list

int size() ; //returns the total number of items in the list

//If 1, 10, 3, 10, 10, and 10 are inserted into an OrderedMultiList oml then //oml.uniqueSize() would return 3 and oml.size() would return 6.

bool find(int val) ; //returns true iff the list contains val

int count (int val) ; //returns number of times val is in the list

void print() ; //prints out the content of the list in order, see below

//mutators

bool insert(int val); //inserts item val into the list (in order),

//returns true iff val was able to fit in list, the only reason it //returns false is if there is no more memory

bool insert(int val, int count); //inserts item val count number of times

//returns false if count less than 1 or if there is no more memory

bool remove(int val); //removes 1 instance of val from list

//returns true iff val was in list

int removeAll(int val); //removes all instances of val from list

//returns number of items removed, which could be 0

private:

//Put node definition here, you may add any private methods you want. };

As in Project 1, you may **NOT** add, subtract, or change any of the public functions other than the addition of const when appropriate. For examples see Project 1. Note that the constant DEFAULT\_MAX\_ITEMS is no longer needed and should not be present in Project 2.

# Strategy

* Do not try to start with your (or my) Project 1 solution. It will be of no real help.
* Code the constructor first as nothing will work correctly without it
* insert is the key. Again try to find ways to reuse code for the two inserts. Test insert extremely thoroughly by using the debugger. Expand the pointers (the + signs) to check that the nodes are pointing in the proper order.
* find should be easy since it was basically given in class. Others will be similar as well.
* Make sure when an insert or remove changes the last node that the new last node’s next pointer is NULL.
* After insert, do the rest of the accessors.
* remove and removeAll are the most difficult as in Project 1. Again, try to reuse code.

Submitting

You will submit through Blackboard, a single zip named **project2.zip** file containing exactly the following:

Your OrderedMultiList code should be stored in two files:

* OrderedMultiList.h
* OrderedMultiList.cpp.

The class header (declaration) and inline functions (if you have any) should be in the .h file the function implementations should be in the .cpp file.

You don’t need to submit your test cases, though you should test your code. You should have NO test code in the two files you submit. There should be NO main in the files you submit as it should be in another file. Since the interface is the same as Project 1, feel free to use your old test cases from testOrderedMultiList.cpp

Your program should not have any undefined (random) behavior. Good style should be used, including comments (which are graded).