Templates

This homework is intended to familiarize you with operator overloading, templated classes, and C++ style IO. For this homework, you may not use C style IO (printf, fgets, etc)- you may only use C++ streams (cin/cout).

1 A Linked List

First, you will implement a generic LinkedList (one templated over the type of data it holds) in `ll.h`. The class definition is provided for you in the file `ll.h` (write your code in the same file i.e. remove the ; and add the method/constructor body).You will probably also need to write some sort of node class and add some private data. You must write each of the following methods:

LinkedList() The default constructor for your list should initalize whatever private data is needed.

LinkedList() The destructor for your list should free any resources that need to be freed.

LinkedList & operator+=(const T & d) The += operator which takes a data item should add the item to the list (in whatever position you choose). It should then return a reference to the list.

LinkedList & operator+=(const LinkedList<T> &other_list) The += operator which takes another list should add all items found in the other list to this list. For simplicity, you may assume that `other_list != this`, however, you may choose to implement this operator to function properly when `other_list == this` for extra credit.

LinkedList & operator-=(const T & d) The -= operator should remove all occureneces of d (as defined by == on T) from this list.

int size(void) const The size() function should return a count of how many items are in the list. It is declared as a const function because it should not modifiy the state of the list.

T& operator[(int index) const] The [] operator should return a reference to the indexth element of the list.

Additionally, the LinkedList class has a friend function which allows you to output it with the stream operator <<:

```
template<class T> ostream & operator<<(ostream & out,
                                       const LinkedList<T> & list)
```

You should fill in this function so it prints the list as followes:

- If the list is empty, it should output `[]`
- If the list has only one item, it should output `[item]`
- If the list has multiple items, it should output `[item1, item2, ... itemN]`

This function should return `out` when it is done so that `<<`s may be chained together.

You may NOT define any other public functions or friends of this class. If you need to declare private helper methods, you may feel free to do so.

2 The LinkedList Tester

The file `lltest.cpp` contains most of the code for a menu drive list of lists testing of your list implementation. It maintains a LinkedList of LinkedLists of ints, and allows the user to perform the basic operations you defined on them. Most of the code is written for you, but for some practice with C++ IO and using your overloaded operators, you must write:

int printMenu(void) This function first print out the menu (the strings can be found in the array `menuItems`) in a format that looks like this:

```
(0) Create New List
(1) Add To A List
(2) Remove From A List
(3) Add One List To Another List
(4) Quit
```

Next, this function should prompt the user to enter a choice, and then read and return a selection within the valid range (hint: use the `readNumberTo` function defined in `lltest.cpp` to restrict them to the range you want).

void remove_from_list (LinkedList<LinkedList<int>> &data) This function should:

1. Prompt the user for, and read the list number to remove from. Use `readNumberTo` to restrict the user to entering a valid list number to remove from.
2. Prompt the user for, and read the item to remove from the selected list. This may of course be any number.
3. Remove the item from the selected list.

void combine_list (LinkedList<LinkedList<int>> &data) This function should:

1. Prompt the user for, and read the list number to add into.
2. Prompt the user for, and read the list number to be added.
3. If the two list numbers are the same, inform the user that such an operation is not supported, and return. (Ommit this check if you do the extra credit)
4. Otherwise, add all of the contents of the later list to the former.

3 Extra Credit

As mentioned above, you may allow the += operator to handle the case where `other_list == this` for 5 points extra credit. If you do this, then if x is `[1,2,3]` then x+=x should result in the list `[1,2,3,1,2,3]`. You should ensure that your combine list function in `lltest.cpp` allow for testing of this case if you choose to do the extra credit.

4 Turn in

Turn in ll.h, ~~ll.cpp~~, lltest.cpp, and Makefile(with targets all and clean).