CISC220H Homework 2

Lab portion due October 5th at midnight on Sakai. The paper copy is due Tuesday, October 6th in lab.

Each part is marked as **[PAIR]** or **[INDIVIDUAL]** and must be completed accordingly. Parts marked with an asterisk are worth zero points – you do not have to complete these parts but you must understand the concepts.

Do not use any classes from any template library; the point of these exercises is for you to code from scratch. All work must be typed, including individual short answer questions.

Academic Honesty Reminder

For parts that are marked as **[PAIR]** you may work with one other student. Both students must submit separate copies of the assignment and include the names of both members of the group.

For parts that are marked as **[INDIVIDUAL]** you may discuss general algorithms with other people, but you may not share any of your code in **any** form. You may not help other people debug their programs, except in the very limited way described on the class website. You may not use any code written by others, whether they are students or not. The TA and the instructor are available for office hours, and by appointment if you can't make hours because of a class conflict.

Assignment

1. **[INDIVIDUAL] 24 pts.** Find the Big-O notation running time for each of these functions. Make sure to show your work.
   (a) $T(n) = 3n + 2n + 5$
   (b) $T(n) = 6n + 3n^2 + 1000$
   (c) $T(n) = O(nlogn) + O(1) + O(n) + O(n^2)$
   (d) $T(n) = 3T(n/2) + n^2$
   (e) $T(n) = 3T(n/3) + n$
   (f) $T(n) = 2T(n/4) + n$
2. **[INDIVIDUAL] 6 pts.** Write a function in C++ with the signature **void f(int n)** that would produce the running time in part 1a. You can use comments,

   ```
   // simple statement
   ```

   to represent a constant time operation.
3. **[PAIR] 40 pts.** Consider the LinkedList implementation available on the course website under homework 2. Implement the following functions on the singly linked list that have declarations in the LinkedList.h file but are not yet implemented in LinkedList.cpp:

   ```
   /**
    * Returns true if the given integer value is a member of the Collection.
    */
   bool member(int) const;
   ```

```
/**
 * Appends the given integer value to the Collection. Placement index of
 * the newly added element is assumed to be last (index=size-1).
 */
void append(int);

/**
 * Removes the value corresponding to the index from the Collection.
 */
void remove(int);

/**
 * Adds the given integer value to the Collection. Placement index of
 * the newly added element is in value order, assuming this LinkedList
 * is already sorted from lowest to highest value.
 */
void addSorted(int);

/**
 * Adds all of the elements of the given LinkedList to the Collection.
 * The given LinkedList and this LinkedList are both assumed to be sorted
 * from lowest to highest value. Placement index of all newly added
 * elements is in value order.
 */
void addSorted(LinkedList*);

/**
 * Appends all of the elements of the given LinkedList in order to the
 * Collection. Placement index of the first newly added element is assumed
 * to be the previous last (index=size-1). This makes new LinkedList nodes
 * for each of the appended values.
 */
void append(LinkedList*);
```

- Edit the LinkedListTest.cpp to include tests and corresponding output for each of the newly implemented functions.
- Based on your code, calculate the order of the running time for each function and put this in your comments for the function (for example, as implemented size has O(n) running time).
- Submit your implemented code and a script file that uses LinkedListTest to test each of your new functions.

4. **[*TEAM 0 pts.]** Deques (short for double-ended queues) are a data structure that combines the functionality of both Stacks and Queues. The Deque ADT defines the following six functions:

```
/**
 * Returns the value at the back of the deque, but does not remove it.
 */
int back();

/**
 * Returns the value at the front of the deque, but does not remove it.
 */
int front();

/**
 * Pushes the given value to the back of the deque.
 */
void pushBack(int);

/**
 * Pushes the given value to the front of the deque.
 */
void pushFront(int);

/**
 * Pops the back element off of the deque and returns its value.
 */
int popBack();

/**
 * Pops the front element off of the deque and returns its value.
 */
int popFront();
```

Implement a deque using your LinkedList class. You will need to do the following:

- Add a Deque.h definition for a Deque, and use multiple inheritence to extend the Deque with your LinkedList.
- Implement each of the Deque ADT functions using your LinkedList data structure.
- What is the running time for each operation using your singly-linked list?
- Improve the running time so that all of the Deque operations run in O(1) time by implementing a doubly-linked list. This will require you to add a **previous** pointer to the LinkNode structure, and you will need to update your implementation for functions in Part 3 to update this pointer.
- Show your Deque working by adding additional test cases to the LinkedListTest.cpp file.

5. **[TEAM 30 pts.]** In part 2, you were asked to implement an append function that creates new linked list nodes for each of the added data elements. In this part, you must implement the append function in O(1) time by reassigning the tail node's pointer to point to the head node of the passed in list.

- Given the restriction that a list instance should not be modified except by calls to its own functions, what kinds of problems will this cause for other operations? Hint: With two lists, A and B, try appending B to A and then removing a value from B. The restriction states that list A should not have its contents changed when a value is removed from B.
- Devise a strategy that will allow the append function to continue to operate in O(1) time, but handles correctly the remove(int) function. Your strategy may add extra fields to the LinkedList class and/or the LinkNode struct, but it may not increase the big-O running time of the remove(int) function.
- Show that your modifications correctly implement the intended behavior for the append(LinkedList*) and remove(int) functions in the LinkedListTest.cpp.
- BONUS (not required but is quite a challenge!): You can modify the other functions in LinkedList to also correctly implement the intended behavior (that a list instance should not be modified except by calls to its own functions).

Your solution must clean up ALL memory allocated from the heap before exiting the main function. You must also document every function in your code and use descriptive variable names.

Submission of completed Homework 2 is due October 5th at midnight. You should submit the following to Sakai and printed copies to your TA:

1. Short answers for Part 1.
2. Code for Part 2 (does not need to compile or run).
3. Code for Part 3, including Makefile.
4. Script output showing execution of Code for Part 3.
5. Code for Part 5, including Makefile.
6. Script output showing execution of Code for Part 5 (and any modified code from Part 3).