CS 130A (Fall 2015)

Project - Part 2

Date Assigned: Monday, October 12, 2015
Due: Thursday, October 29, 2015 at 11:59 PM
Submission name for this part: Project_Part2

This is the second part of your long-term Social Network project. You should be building on top of your code from part 1. This would also be a good time to fix some things you did back then that you would like to improve. (If you feel it is necessary, you can also start from scratch.)

As a piece of advice, it is recommended that you first get some of the functionality to work fully, then add more. You will receive more partial credit for a functioning solution missing some of the features than for a lot of almost correctly implemented code that does not compile.

1. Review chapters 3.1, 3.2 from the textbook.

2. Implement a pure abstract version of the `List` class. (Hint: we almost completely solved that for you below.) It should have following four functions:

```
template <class T>
class List {
void insert (int pos, const T & item);
    /* Inserts the item right before position pos, growing the list by 1.
        pos must be between 0 and the current length of the list.
        (feel free return bool, if you want.) */
void remove (int pos);
    /* Removes the element at position pos, shrinking the list by 1.
        pos must be between 0 and the current length of the list minus 1. */
void set (int pos, const T & item);
    /* overwrites position pos in the list with item.
        Does not change the length of the list.
        pos must be between 0 and the current length of the list minus 1. */
T const & get (int pos) const;
    /* returns the item at position pos, not changing the list.
        pos must be between 0 and the current length of the list minus 1. */
};
```

   It is up to you to decide what to do when pos is out of its legal range.

   (a) Derive from `List` a class that builds an implementation based on linked lists. You may use your code from previous part if it helps you.

   (b) Derive from List a class that builds an implementation based on arrays. Remember that internally, you will store the items in an array. When you have inserted enough items to exceed the current size of the array, you allocate a larger array and copy over the elements. You need to double the array size in such case. Watch out for memory leaks.

   (c) For both of your implementations, measure how long they take to insert n numbers into a `List<int>`. Recall that inserting an element in position i takes $\Theta(i)$ for the Linked List, and $\Theta(n-i)$ for the Array. So to be fair, all elements should be inserted in the current middle of the list (plus/minus 1). For values n = 2000, 4000, 6000, 8000, 10000, measure

how long this insertion takes. In this case, do not read the numbers from a file, but rather just add the numbers from 1 to n (so we don't measure the time to read a file).

3. Now that you have implemented `List` class, you should realize that a List is really a better implementation of a users Wall than just a linked list or such. Go back to your implementation, and use the List data type for a users Wall. This should also simplify the interaction with the user when deleting a Wall post. You can internally use either an array-based or linked list-based implementation.

4. Now add iterators both to your `Wall` class and to your UserNetwork class. In previous implementation, you might have passed head pointer around to go through all the list items. You should now instead use the iterators to implement the same functionality. The recommended, and probably easiest, way to do this is to add an iterator to your underlying `List` classes.

5. Each user in the network may have some friends. Friendship is mutual i.e. if A is a friend of B, B is also friends with A. You need to incorporate this functionality in your User class. This can be achieved by having another field in your class that stores Bag of friends. This way, you will implement an undirected graph on users as an adjacency list. Of course, you will need to modify other functions so that the friend lists should be saved and loaded along with all other user data. You can choose the format in which youd like to store them.

6. For the user interface, add the following:

   Once a user is logged in, enable the user to:

   (a) Create a new wall post.
   (b) Delete an existing wall post. Think about a good way in which the user can select the post to delete.
   (c) Change any of the fields that you think should be changeable.
   (d) Log out. When a user logs out, all the changes that were made should be stored in the file so they are available next time.
   (e) Delete his/her profile from the network. Its a good idea to ask for confirmation again before taking any action.
   (f) Search for other users by name. User should be able to enter a query string and you should return list of users whose names contain the query string (i.e. partial matches are allowed). Matches should not be case-sensitive. For example, if the user types Ring, you should find Alan Turing if hes in the database. You are recommended to learn C++ string functions before implementing search functionality. In-built string library functions will make this job easier for you. The iterator you included in `UserNetwork` class will come handy to go through all the users in the network.
   (g) After user gets the search results, implement a way for the user to send a friend request to another user which appears in the results.
   (h) When a user logs in, show the user all pending friend requests. User should be able to accept or delete individual friend requests. He can also take no action so that the list is shown again the next time user logs in. You may want to add a Bag of friend requests member to your User class to implement this functionality and some additional functions.
   (i) Implement a functionality for the user to look at his current friend list and delete friends. Make sure deletion happens mutually.

   The current implementation does not allow a user to look at anybody elses wall (and hence to comment on or like any of the posts). Users are also not able to interact now via messages. But while designing your implementation, keep in mind that future parts of the project will ask you to add these functionalities.