CA 3: "Facebook Lite"
CS-240 Fall 2017: Data Structures

Goal

In CA 3, you will continue with C++ I/O and text processing, and you will begin to utilize somewhat more complex data structures to store information. In particular, you will implement a *linked list* container to store, manipulate, and save Facebook-like posts and related information. CA 3 will provide you experience with:

- a simple singly linked list container that requires pointers
- C++ heap memory management (strategic and extensive use of `new` and `delete`)
- deep copies of stored data, including in copy constructors and assignment operators
- operator implementation and overloading (including assignment operators, unary operators, binary operators, and `friend` functions)
- some simple C++ class and object design.

This program will be difficult for some of you so please begin early and work on it consistently. You can do it and we will help! When you finish successfully, you will *really* understand linked lists, operator overloading, deep vs. shallow copy, and more, *guaranteed*!

Overview

Write a C++ program that reads and stores information from and to text files that contain Facebook Lite Users, which are held in your program as instances of a new `FBLUser` class that you write. Each `FBLUser` object should contain a unique Userid, a Password, a First name, and a Last name. To store multiple users, you should implement an `FBLUser` Linked List class (which we will call `FBLUserLL`), of which your program will create exactly one instance (similar to how your CA 2 program created a single instance of the Donor Database).

In addition to login and name information, each `FBLUser` object (and therefore your `FBLUser` C++ class) should also contain a Linked List of `FBLPost` objects. That is, each `FBLUser` object should contain an `FBLPost` Linked List object, which we will call `FBLPostLL`, and that holds a linked list of instances of an `FBLPost` C++ class.

In Facebook Lite, each `FBLPost` can be *Liked* or *Commented* on (like Facebook). Your program need not track which user likes a post, it should just count the number of likes, by also maintaining an integer counter in each `FBLPost` object. You will not have to make sure that a post is not liked more than once by a user. `FBLComments` should be stored in an `FBLComment` Linked List object called `FBLCommentLL`, which is a data member of the `FBLPost` class.

Finally, Facebook Lite Users can *Friend* one another. Each `FBLUser`'s friends are stored in a *Linked List of pointers to FBLUsers*. That is, the type of the data field of an `FBLFriendLL` Node is a pointer to an `FBLUser`.

Tackling the Project in Phases

You will complete this assignment in *three phases*, to introduce you to large-scale program development strategies. That is, rather than tackling an entire task (assignment) all at once, it can be useful to develop functionality incrementally. Phase 1 will be submitted as CA 3.1, Phase 2 as CA 3.2, and Phase 3 as CA 3.3, with the functionality of each described below. Each phase will be graded separately.

Phase 1 (Turned in as CA 3.1)

For this phase, you will implement the `FBLUser`, `FBLUserLL`, `FBLPost`, and `FBLPostLL` C++ classes, along with two menus.

Top Level Menu

A *Top Level Menu* should accept the following commands from your program's user.

`CREATE <Userid> <Password> <First> <Last>`

This operation creates a new Facebook Lite User, assigning into the data fields the information on the rest of the line. You may assume (i) that the fields are separated by whitespace, (ii) that none of them contain whitespace, and (iii) that there are exactly 5 C++ "strings" on the line. You should check to make sure the `<Userid>` has not yet been used, but the other fields have no limitations, even in terms of special characters, etc. (Error checking this input is not part of Phase 1.)

`LOGIN <Userid> <passwd>`

This operation logs a user in, so that (for example), the correct feed will be displayed when requested. The `LOGIN` operation should bring your program's user to the *2nd Level Menu*, described below.

`QUIT`

This command ends the program.

2nd Level Menu

The 2nd Level Menu should support the following commands:

`LOGOUT`

The `LOGOUT` operation brings the user back to the Top Level Menu.

`POST <text>`

This operation creates a new Facebook Lite posting, "from" the currently logged in FBL User. The `POST` command is separated by whitespace from the text of the post itself; after "POST" and all of the whitespace that follows it, the entire rest of the line should be considered the user's post (including subsequent whitespace). The post will not, however, span multiple input lines. So in response to a `POST` command, your program should create a new `FBLPost` object and insert it into the currently logged in FBL user's `FBLPost` Linked List.

`READ`

This operation should display the first `FBLPost` *in the currently logged in user's own list of postings* (for now), and remove it from that list.[1]

Posts should be displayed in the order they are inserted… the oldest post should be displayed first. If your program's user tries to `READ` from an empty feed, your program should display the message "Nothing to

[1] We will change this functionality for future phases. *Later*, an `FBLUser`'s feed will contain posts from other FBLUsers; for CA 3.0, it contains the user's *own* posts only.

READ".

Summary

That's it for Phase 1 (CA 3.1): A Top Level menu consisting of three operations: CREATE, LOGIN, and QUIT, along with a 2nd Level menu that supports LOGOUT, POST, and READ. Minimal error checking; you may assume that input lines are all well-formed.

To implement this functionality, you will need the following four C++ Classes:

`FBLUser`, `FBLUserLL`, `FBLPost`, and `FBLPostLL`

You will also need two different Linked List Node C++ Classes, one for the `FBLUserLL` class and one for the `FBLPostLL` class.

For CA 3.1, you should *not* support *Likes*, *Comments*, or *Friends*. Focus instead on reading in the commands from the user, and getting the simple Linked List operations working correctly. `CREATE` and `POST` are essentially Linked List `insert()` operations, on `FBLUserLL` and `FBLPostLL` linked list objects, respectively. `LOGIN` and `CREATE` require traversals of `FBLUserLL`, and `READ` requires you to remove a single node from one end of an `FBLPostLL` object.

Phase 2 (CA 3.2)

So far (in Phase 1), when a user creates a post, that post gets copied only into his or her own linked list of posts. The next phase of the assignment will allow users to read *other* users' posts, made by their friends only. Each user should have a *wall* of his or her own posts (only, in Facebook Lite), and a *feed* of posts made by all friends.

Add an STL vector of friends to the `FBLUser` class, so that each user has a vector of friends. Pick an appropriate type for the vector template parameter, based on what you will need to do with those friends! (below)

Now, add to the 2nd level list a FRIEND option, as follows:

`FRIEND <userid>`

That should make the user identified by `<userid>` be a friend of the currently logged in user, by adding

Now, when a user posts something, your program should insert that post into the poster's own wall (i.e. his or her linked list of posts), as for Phase 1, but also into each friend's *feed*. For now, make a *copy* of the post and insert that *copy* into each friend's feed. You may make the feed a vector of posts, or a linked list of posts, it's up to you.

Now, make a new function in the 2nd level menu to display all of the logged-in user's friends' full names. So typing:

`MYFRIENDS`

…might show:

Joe Schmo
Tom Brady
Barack Obama

Add two functions called MYFEED and MYWALL that show the logged-in user's *feed* linked list of posts (by other users), and his or her *wall* linked list of posts that he or she has made. These options should display all the posts, but not remove any.