CS 104 (Fall 2013) — Assignment 6
Due: 10/29/2013, 11:59am

BitBucket directory name for this homework (case sensitive): HW6

This homework is the second part of your long-term Social Networking Site project. It will be graded as part of the project (rather than a regular homework). You should be building on top of your code for HW3, though 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 C++ Interlude 6 and Chapters 10, 12, 20 from the textbook.

(2) Now that we have learned about the List data type, you should realize that a List is really a better implementation of a user's Wall than just a linked list or such. Go back to your implementation, and use the List data type for a user's 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, and are welcome (in fact, encouraged) to reuse your code from Homework 4. Notice that for the UserList, it's probably better to keep them as a Bag, since the sorting doesn't matter.

(3) Now that we have learned about iterators, add iterators both to your Wall class and to your UserList class, so that you can display all of their items. Use the iterators to implement the functionality that you previously got by passing around the heads of the lists. The recommended, and probably easiest, way to do this is to add an iterator to your underlying List and Bag classes.

(4) Implement a way for a user to have friends. Most likely, what you want to do is to give each user another field, which is a Bag of friends. In this way, you will implement an undirected graph on users as an adjacency list. Of course, the friend lists should be saved and loaded along with all other user data. You can choose the format in which you'd like to store them.

(5) For the user interface, add the following:

(a) A way for a user (once logged in) to search for other users by name. What you should do here is to let the user enter a string, and return on the screen all other users whose names contain that string. For example, if the user types "Roja", you should find "Tommy Trojan" if he's in the database. (Notice that this means that you should ignore case.) You are strongly encouraged to make your life easier here by learning about C++ string functions first. Your iterator of your UserList will likely come in handy here.

(b) A way for a user to select from the search results and send a friend request to another user.

(c) A way for a user (once logged in) to see all pending friend requests, and confirm or delete them (or leave them alone). This may entail adding another field to your User type, such as a Bag of pending friend requests.

(d) A way for a user to look at the list of all his/her friends and delete friends.

For now, you do not have to allow anyone to see anyone else's Wallposts or reply to them or like them. Nor do you have to implement messages for now. Though in your design, you may want to keep in mind that a future homework will ask you to add both of those.

(6) Chocolate Problem: This problem does not count towards your grade, but if you solve it, you can earn up to 2 chocolate bars. Feel free to specify your preferred type of chocolate — your requests may be accommodated if they are reasonable. Submitting this problem a few days late is ok, but in that case, please submit it directly by e-mail to David.

One of the many many fun things you can do with graphs is use them to model transitions between words. For instance, when you see the following text fragment:

> I have always liked computer games. Now I am studying computer science. It is not all fun and games. In fact, it is a lot of work.

you can learn that "I" is followed half the time by "have" and half the time by "am". "computer" is followed half the time by "games" and half the time by "science". "games" is always followed by ".", and "it" is always followed by "is". When you analyze enough text, you get a fairly accurate idea of what English looks like, or maybe not the entire language, but the text corpus you're looking at.

Write a program that takes in a text file, and learns from it the transition probabilities between all of its words as labels on edges of a graph. It's probably a good idea to treat punctuation marks as words. You probably also want to turn everything into lower-case.

Once your program has learned the transition probabilities, use your program to generate some "pseudo-English gibberish" by following random transitions according to your learned probabilities. Then amuse yourself with the funny stuff that comes out.