Programming Assignment 2

Assigned: Sept. 27
Due: Oct. 25.

In this assignment, you will use hash tables and linked lists to implement a micro-version of Facebook.

Specifically, your program will accept from input a sequence of commands of the following forms, one command to a line:

- P ⟨name⟩ – Create a person record of the specified name. You may assume that no two people have the same name.
- F ⟨name1⟩ ⟨name2⟩ — Record that the two specified people are friends.
- U ⟨name1⟩ ⟨name2⟩ — Record that the two specified people are no longer friends.
- L ⟨name⟩ — Print out the friends of the specified person.
- Q ⟨name1⟩ ⟨name2⟩ — Check whether the two people are friends. If so, print "Yes"; if not, print "No"
- X – terminate the program.

For instance, this is one possible input and output:

```
Input                  Output
P Sam
P Liza
P Mark
P Amy
F Liza Amy
F Liza Mark
F Amy Sam
L Amy                   Liza Sam
L Sam                   Amy
U Liza Amy
L Amy                   Sam
Q Liza Mark             Yes
X
```

Moreover:

- All these commands should execute in time that is independent of the total number of people in the system.
- The command "L" should take time proportional to the number of friends that the person has. The command "U" should execute in time proportional to the sum of the friends that the two named people have. The commands "P", "F", and "Q" should execute in unit time.

Data Structures

You *must*

A. Define a "Person" class which has one field for the name and another field for the linked list of friends.

B. Store the friends of each person in a linked list. You may use any linked list class you want: The Java library linked lists, a linked list class you write yourself, any of the linked list classes presented in class, something you found somewhere on the Web, etc. However:

   – The list of friends *must* be in a linked list of some description, and *not* in an array.
   – If you use any class that is not a Java library class and is not of your own design then you *must* indicate its source in a comment. This applies to the linked list classes presented in class as well.
   – The list must be a list of `Person` objects, *not* a list of their names, as strings.

C. Create a global hash table that indexes each `Person` object under the name. Here, I recommend that you use the Java library functions; but you may use a hash table definition from anywhere else. The same rules apply on citations.

D. In order to execute the command "Q" in unit time, you will need a second hash table, that uses as a key a pair of names. The way you use a pair of names as a key is to put the two names in alphabetical order, then combine them separated by an asterisk. For instance, the pair of names "Sam" and "Liza" becomes the string "Liza*Sam". This second hash table uses these strings as keys, and booleans as values.

Executing commands

To execute a "P" command, create a `Person` object for the name, and save it in the first hash table under the name.

To execute an "F" command:

- Find the two `Person` objects in the single-name hash table.
- Add each person to the front of the linked list of the friends of the other person.
- Construct the two person key for the two names.
- Add the two person key to the two-name hash table, with value `true`.

To execute a "U" command:

- Find the two `Person` objects in the single-name hash table.
- Delete each person from the linked list of the friends of the other person.
- Construct the two person key for the two names.
- Delete the two person key from the two-name hash table.

To execute an "L" command", find the person object in the single-name hash table, and loop through the list of friends.

To execute a "Q" command, construct the two-name key and look it up in the two-name hash table.

Input/Output

You may assume that the input is correctly formatted. That is:

- Each line consists of a command character 'P', 'F', 'U', 'L', 'Q', or 'X' followed by a blank followed by one or two names separated by a blank. A name is a sequence of alphabetic characters. Do not worry about normalizing case.
- Any name mentioned in an F, U, L, or Q command has been already created by a P command.
- The sequence of commands ends with X.

What, if anything, you want to do about invalid inputs is up to you. It is OK for the program to crash.

However, the program *should* do the right thing under the following circumstances:

- A person friends or unfriends himself. In this case, the program should do nothing; it should *not* add the person to his own list of friends.
- A person unfriends someone who is not a friend. In that case, the program should do nothing.

The program may take its input either from standard input or from a text file in the same directory as the program named "input.txt" – your choice.

If this form of input seems to you too dreary for words, you can feel free to design a snazzier one, as long as it supports the above functionalities and it is immediately obvious to the grader how to work it. It is entirely up to the grader to decide what is obvious to her; I am not going to overrule that.

Submission

Send an email to the grader with the source code as an attachment. The "main" method should be in a class called `MicroFB`, and therefore in a file called MicroFB.java. If your code is all in one file, you may simply attach the .java file. If your source code is in several files, combine them into a single .jar file. Any instructions about how to run the code etc. should be in the body of the email message.

If you become a billionaire

If you turn this modest programming assignment into a commercial product that earns you a billion dollars, then do keep in mind the fine educational institution where you learned about hash tables, to say nothing of the inspiring course instructor, the ever-helpful recitation instructor, the diligent grader, etc. etc.

Honors assignment

Do the same assignment, but do the implementation in such a way that the command ‘U’ also executes in constant time. To do this, the two-name hash table must point to the record for each of the two friends on the friends list of the other. For example, if you execute the command “F Sam Liza” you should execute the following steps:

- Using the single-name hash table, find the objects for Sam and Liza.
- Add Sam at the front of the list of Liza’s friends and vice versa.
- Create the two-name key “Liza*Sam”
- Create an object `O` with two fields. The first is a pointer to the node corresponding to Sam in the linked list of Liza’s friends, and the second is a pointer to the node for Liza in the linked list of Sam’s friends.
- Save the object `O` in the two-name hash table under the key “Liza*Sam”.

Now, if at some later time Liza and Sam unfriend, you look up “Liza*Sam” in the two name hash table, you find the two nodes in the friends lists, and since you have the actual nodes, you can delete them in constant time.

If you are doing the assignment in this form, then I recommend the `MyList2` doubly-linked list definition presented in class. Feel free to modify this, of course. You should still use the Java library hash table.