CSC221 Data Structures & Algorithms I Fall 2015

Programming Assignment #2 – Linked Lists

In this lab our goals are to:

- continue to review and extend your C++ programming skills
- implement operations on linked lists
- work with header files, object files, linking, and Makefiles

Project Description

In this assignment we will process a series of transactions simulating the "friend" feature of a new social media site named "Phase Book", especially appealing to folks who study electromagnetic wave propagation .

The input is a sequence of transactions. The only transactions allowed are listed below:

- A new member joins
- A current user quits
- Member *A* sends a friend request to user *B*
- Member *B* accepts a friend request from user *A*
- Member *A* un-friends *B*
- Print member *A*'s account information, including:
  - all friends of *A*
  - all pending friend requests sent by member *A*
  - all pending friend requests received by member *A*
- List all members and all their friends

Your program will maintain a list of members. For each member, you will store the on-line name and three lists. The data members of class "member" include:

- member's name
- a list of friends
- a list of pending friend requests sent
- a list of pending friend requests received

Each member name is an on-line alias consisting of a single character string containing only upper and lower case letters, underscore, and numerals. E.g., **turtle_dove**, **BetterFocus**, and **refractive_index4** are all acceptable member names. Each user name must begin with a letter: A – Z, a – z.

The input is read from standard input (cin). Use UNIX redirection to enable your program to process an input file. A test file will be provided on

http://menehune.opt.wfu.edu/csc221

Input Format

The input format is one transaction per line. Each line contains of a single letter transaction code, followed by zero or more member names, as required for that transaction. There is at least one space between the transaction code and the user names; also, there is at least one space between user names for those transactions which involve more than one member. The transaction codes are illustrated by example in the following table.

Transaction	Interpretation
J Phil	Phil asks to join Phase Book
Q Bob	Bob quits Phase Book
S Mary Alice	Mary sends a friend request to Alice
A Alice Mary	Alice accepts Mary's friend request
R Alice Mary	Alice rejects Mary's friend request
U Bob Phil	Bob un-friends Phil
L Bob	List Bob's account, including his friends, his pending friend requests sent and his pending requests received.
L *	Special character '*' lists all user accounts.

Suggested Organization

Your code should be organized in 5 files:

namelist.cc – Contains the implementation two classes:

- **listcell** – list cell for a linked list of names
- **namelist** – implements a linked list of names

namelist.h – Header file for classes implemented in namelist.cc

members.cc – Contains the implementation of two classes:

- **member** – list cell for a member. Member data includes:
  - **name** – the on-line name of the member
  - **friends** – list of friends
  - **invitations_rcvd** – list of friend invitations received
  - **invitations_sent** – list of friend invitations sent
- **memberlist** – implements a linked list of members

members.h – Headers for classes implemented in members.cc

main.cc – Main program to loop through transactions.

Implementation Details

The structure of the lists is illustrated in Figure 1. There are several details to consider for the operations relating to friend requests, accepts, and rejects. Special considerations for each operation are listed below:

J *name* Remember to check if member *name* already exists. If so, reject the transaction and issue an error message.

Q *name* When a member quits Phase Book, that member's on-ine name must be removed from all lists including: the member lists, all friend lists, and all pending request lists. If the name does not appear in the member list, issue an error message.

S *sender receiver* When a friend request is sent, the sender's name will usually go on the receiver's "friend request received" list. In this case, the receiver's name must also go on the sender's "friend request sent" list. If either the sender or the receiver is not found as expected, issue an error message.

Further, do not duplicate requests on the lists. I.e., it may be that *sender* has already sent *receiver* a friend request. Issue a warning message when you detect a duplicate friend request.

It is also possible that *sender* and *receiver* mutually send friend requests before either accepts. E.g., Phil sends Joe a friend request, and subsequently Joe has also sent Phil a friend request. If this occurs, you should treat the second request as if it were an acceptance. Both members should be taken of each other's requests-pending lists and put on their friends list.

Finally, it could happen that Phil sends Joe a friend request when Phil and Joe are already friends. In this case, do not add the request to either request lists. Issue a warning message indicating that Phil and Joe are already friends.

A *acceptor requester* The name of the requester must be deleted from the acceptor's "friend request received" list and placed on the acceptor's "friends" list. Similarly, the name of the acceptor must be deleted from the requester's "friend request sent" list and placed on the requester's "friends" list. If either acceptor or requester are not found on the expected lists, issue an error message.

R *rejector requester* The name of the requester must be deleted from the rejector's "friend request received" list. Similarly, the name of the rejector must be deleted from the requester's "friend request sent" list. If either rejector or requester are not found on the expected lists, issue an error message.

U *member other-member* The name of the other-member must be deleted from the member's "friends" list. Also, the member must be deleted from the other-member's "friends" list. If either the member or the other-member are not found on the expected lists, issue an error message.

L *member* List the member's name along with his/her friends, his/her friend requests sent, and his/her friend requests received. Use labeling and indentation to produce a user-friendly listing.

L *** List all members with all details.

Collaboration and Teamwork:

On this project you are permitted to work in groups of two if you choose. If you choose this option, you must send e-mail to *torgerse@wfu.edu* saying who is in the group. The group size is limited to two students. The basic premise is that these groups should be equal partnerships with each student contributing equally to the completed project. This

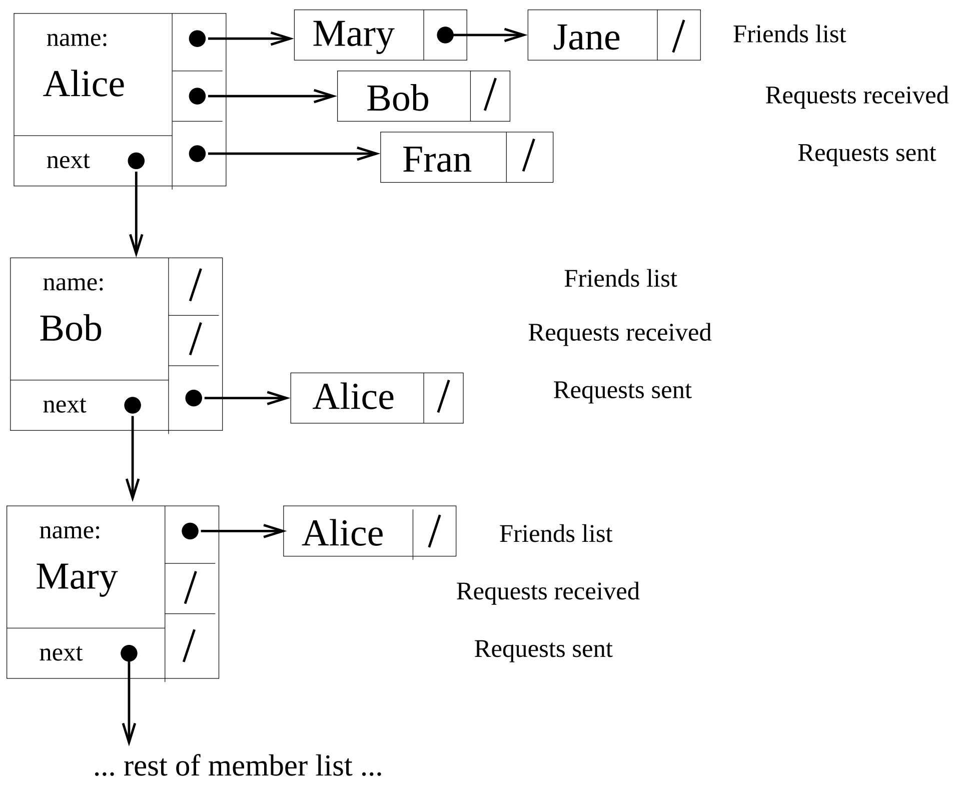

Figure 1: List Structure

dynamic is difficult to sustain as groups become larger, thus the limit of groups size two. Both students in the group will receive the same grade for the project.

During your University experience, most class assignments must be completed individually, with no significant outside assistance. After graduation it is rare that projects are developed by a single person working alone. One of the goals in this assignment is to give you some collaborative experience. I prefer that you form groups of your own choosing, since forming a team for a project is also a workplace skill. But, if you wish to work in a group of two, but do not know anyone in the class well enough to ask them, you may e-mail me. I can collect e-mails from those students looking to form a group, and randomly assign partners.

Turn In:

Keep all your work in a sub-directory named `Lab2`. Change to the parent directory of `Lab1` and create a tar archive of your work using the command:

```
% tar cf Lab2.tar Lab2
```

Upload the file `Lab2.tar` to your account on telesto.

If you are working in a group, you should decide on which partner is the corresponding author, and which is the supporting author. If you are the corresponding author, upload your code to telesto. If you are the supporting author, upload a text file named "lab2_collaborator.txt". The contents of the text file should indicate the name of the corresponding author.