COMP 2611: Data Structures
Assignment #4
Date Due: 11:59 p.m., 8th July, 2019

Description

In a certain social media application, there are *n* users. An *n* x *n* matrix *F* represents the “friend” relationship between users. The values stored in the matrix are as follows:

- If $i == j$, $F_{i,j}$ is *false*
- If *i* is a friend of *j* or *j* is a friend of *i*, $F_{i,j}$ is *true*; otherwise, $F_{i,j}$ is *false*

The users of the application are stored in a file, `friends.txt`. The format of the file is as follows:

- The first value in the file is the number of users.
- This is followed by the name of each user (as a string).
- This is followed by the $F_{i,j}$ values for *F* (*n* x *n* values in all)

You are required to read the `friends.txt` file and store the $F_{i,j}$ values conserving storage as much as possible. Your program should then present a menu from which various operations are performed.

The assignment consists of two parts. In Part A, a one-dimensional array is used to store the friend information between users. In Part B, a linked-list is used to store the friend information.

Part A

Using a one-dimensional *bool* array to store the friend relationship between users, write the code for the functions listed in Table 1 below.

Function and Description
`int getUser (string users[], string user, int numUsers):` Returns the index of the given *user* in the *users* array or -1 if *user* is not present.
`bool isFriend (bool A[], int numUsers, int i, int j):` Returns true if *i* is a friend of *j*, and *false* otherwise.
`void setFriend (bool A[], int i, int j, bool value):` If *i* is a friend of *j*, sets the corresponding location in *A* to *true*; otherwise, sets the corresponding location to *false*.
`int getNumFriends (bool A[], int numUsers, int i):` Returns the number of friends of User *i*.
`void displayFriends (bool A[], string users[], int numUsers, int i):` Displays the friends of User *i*, using their names, rather than indices.
`bool paradox (bool A[], int numUsers, int i):` The "friendship paradox" is the phenomenon that most people have fewer friends than their friends have, on average. This function returns *true* if the paradox is true for User *i* and *false*, otherwise.
`int readFriendsFile (char fileName[], string users[], bool A[]):` Reads the data from the `friends.txt` file and returns the number of users. The names of the users are stored in the *users* array and the friend matrix *F* is stored in a one-dimensional array, *A*.
`void displayFriendsMatrix (string users[], bool A[], int numUsers):` This function displays the original matrix *F*, as read from the `friends.txt` file. The row headings and column headings are the names of the users.

Table 1

Main Program

Your main program should read the data from the `friends.txt` file and then provide a menu from which various operations are performed. The operations are performed by calling functions from Table 1.

The following is the menu that must be displayed:

```
Assignment 4: Working with Matrices
--------------------------------------------------

1. Display the Friends Matrix
2. How Many Friends Does User A Have?
3. List the Friends of User A
4. Is User B a Friend of User A?
5. Defriend User A and User B
6. Does the Friendship Paradox Hold for User A?
7. Quit

Please enter an option:
```

NB: The input for Options 2-6 should be the *names* of the respective users and **not** the integer values representing their row or column indices.

Part B

With millions of users in a social media application, it is likely that even when a one-dimensional array is used to store friend information, there will be a large amount of locations with the value *false* (indicating that *i* and *j* are not friends).

An alternative scheme is to store the *true* elements of row *i* (representing the friends of User *i*) in a linked list (see Pages 310-312 of the text). A node in the linked list is defined as follows:

```
struct Friend {
      int column;
      Friend * next;
};
```

Using this approach, write the set of functions in Table 2.

Function and Description
`Friend * createFriend(int column):` Creates a *Friend* node for a linked list with the given column and returns the address of the node created.
`int getUser (string users[], string user, int numUsers):` Returns the index of the given *user* in the *users* array or -1 if *user* is not present.
`bool isFriend (Friend * A[], int numUsers, int i, int j):` Returns true if *i* is a friend of *j*, and *false* otherwise.

`void setFriend (Friend * A[], int i, int j):` If *i* is a friend of *j*, sets the corresponding location in *A* to *true*; otherwise, sets the corresponding location to *false*.
`int getNumFriends (Friend * A[], int numUsers, int i):` Returns the number of friends of User *i*.
`void displayFriends (Friend * A[], string users[], int numUsers, int i):` Displays the friends of User *i*, using their names, rather than indices.
`bool paradox (Friend * A[], int numUsers, int i):` The "friendship paradox" is the phenomenon that most people have fewer friends than their friends have, on average. This function returns *true* if the paradox is true for User *i* and *false*, otherwise.
`int readFriendsFile (char fileName[], string users[], Friend * A[]):` Reads the data from the `friends.txt` file and returns the number of users. The names of the users are stored in the *users* array and the friend matrix *F* is stored in a one-dimensional array of linked lists (of type *Friend*), *A*.
`void displayFriendsMatrix (string users[], Friend * A[], int numUsers):` This function displays the original matrix *F*, as read from the `friends.txt` file. The row headings and column headings are the names of the users.

Table 2

Main Program

The main program should provide the same functionality as in Part A.

Submission

You must submit two files for this assignment:

(1) `Assignment4-PartA.cpp`
(2) `Assignment4-PartB.cpp`