CS 2506 Computer Organization
Project 3
The Social Network

Assigned: October 27, 2012
Due: 11:55 p.m., November 14, 2012
Late submission policy mentioned below
Purpose: To help you learn Data Structures and Pointers in C.

In this project, you are required to build a C program which will emulate the functionality of a social network. You are required to use adjacency list and the data structures provided. The list of supported commands is as follows:

- **add_user :** Adds a user to the social network
  Usage: add_user name age gender location
  name = alpha-numeric username of max length 20 characters (No spaces allowed)
  age = Integer. Between 1 and 100
  gender = male/female (convert this to 1 or 0 while storing)
  location = alpha-numeric location of max length 20 characters (No spaces allowed)
  Output: If adding the user is successful, print "username joined facebook"
  User Data Structure: Please use the following data structure to store users:

```
struct userNode{
        char name[20];
        int age;
        int gender;
        char location[20];
        struct friendNode *friend;
        struct userNode *nextUser;
};
```

  Error Handling: 1. Make sure that both username and location is alphanumeric with no spaces and their length is not more than 20 characters
  2. Make sure that age is between 1 and 100
  3. While adding make sure that the username is unique and doesn't exist before
  4. Make sure that the user provides name, age and location
  5. Make sure that for gender the user enters either male or female
  Suggestions: Keep the list of users sorted. See the video for more suggestions.

- **add_friend :** Takes two usernames and adds them as friend in the social network
  Usage: add_friend username1 username2
  username1 = Name of the first user
  username2 = Name of the second user
  Output: If added successfully, print, "username1 and username2 are friends now"
  Friend Data Structure: Please use the following data structure to store friends' information

```
struct friendNode{
        struct friendNode *nextFriend;
        struct userNode *user;
};
```

  Error Handling: 1. Make sure that the user has provided you two usernames
  2. Make sure that the provided usernames exist.
  3. Make sure that they are not already friends in first place.
  Suggestions: Keep the list of friends sorted. Make a function isExist to see whether the particular user exists or not. Make another function areFriends to see whether the two users are friends or not.

- **delete_friend :** Takes two usernames and unfriend them
  Usage: delete_friend username1 username2
  username1 = Name of the first user
  username2 = Name of the second user
  Output: If successfully deleted, print, "username1 and username2 are no longer friends"
  Error Handling: 1. Make sure that the user has provided you two usernames
  2. Make sure that those usernames exist.
  3. Make sure that they are friends in first place.
  Suggestions: Use the function isExist to see whether a user exists or not. Use the function areFriends to see whether the two users are friends or not. Make sure that you modify the friends list for both the users.

- **delete_user :** Takes a username, remove all his friends and then delete the user
  Usage: delete_user username
  username = Name of the user to delete
  Output: If successful, print, "username left facebook"
  Error Handling: 1. Make sure that the user has provided you a username
  2. Make sure that the username exists
  Suggestions: Call the function delete_friend for all the friends of the user. Then delete the user. Use the function isExist to see whether the user exists or not.

- **display_friends :** Takes a username and displays all his friends
  Usage: display_friends username
  Username = Name of the user
  Output: If the user has friends, then an alphabetically sorted list of friends along with the Count (print it before printing the list). Otherwise print, "username has no friends"
  Error Handling: Make sure that the username exists
  Suggestions: Keeping the friend list sorted will help in printing.

- **search_users :** Search users based on either name, location, age or gender. Searching can be done based on only one of the keywords (ie only one among location/age/gender/location).
  So no AND or OR operators supported.
  Usage: search_users location location_name
  search_users name username
  search_users age age_int
  search_users gender male/female
  Output: If users found, print the alphabetically sorted list of users in following format
  username1/age/sex/location, username2/age/sex/location; along with the count.
  If no users found, print, "No users found matching the given search criteria"
  Error Handling: 1. Make sure that only one of the 4 search keywords is provided
  2. For gender search keyword, make sure that either male or female is provided

- **suggest_friends :** Takes a username and suggests users of opposite sex having maximum no. of mutual friends
  Usage: suggest_friends username
  username: Name of the user
  Output: If users found, print the alphabetically sorted list of users in following format
  username1/age/location, username2/age/location
  If no users found, print, "No users to suggest"
  Error Handling: Make sure that the username exists
  Suggestions: Start from the root and find all the users who are not friends (using the areFriends function) and are of opposite sex. Then count the maximum number of mutual friends (by going through the list and using areFriends function).

- **display_graph:** Display the whole graph of users and friends
  Usage: display_graph
  Output: A sorted list of users along with the sorted list of their friends
  (see the TA's implementation for the exact formatting)

- **exit_program :** Quits the program
  Usage: exit_program_

If the user enters any command other than the commands mentioned above, print that the entered command is invalid and ask him to try again.

Note that all the commands are of variable length.

The program should accept a few command-line switch arguments, which are described as below:

./social_network [-i]|[-f input_file output_file] [-h]

- **-i** **:** To run the program interactively and read the commands from the command-line and output back to the command-line. All error messages are also reported to the command-line.
- **-f input_file output_file :** To read the program command inputs from input_file and output the results to the output_file. All error messages are also reported in the output_file Make sure that if the user is using this switch, he is providing names for input_file and output_file
  Only one of –i or –f switch should be provided. If both are provided –i takes precedence over –f and the program will behave interactively.
- **-h** **:** Prints the usage information. The usage information includes the commands supported by the program (a one line description and usage information) and the switch arguments. –h takes the highest priority.

If no command-line switch argument is provided, then your program should print the usage information, which is same as using the –h switch argument. Ignore all other invalid switch arguments.

OTHER DETAILS

Files Information
Your program might have the following files:

- social_network.c : This file will contain the main function and will call other function from other .c files
- project2.c : If you are using any functions from project2, put them here
- graph.c : This file will contain the implementation of your social graph functions
- project 2.h : Header file for project2 functions
- graph.h : Header file for graph functions

Make sure that header files are properly included. Improper usage of header files can lead to point deductions.

Suggestions: These are just the basic set of files that your program can have. In case, if you want to include more .c files or .h files, feel free to do that.

Compilation Information
The following command will be used to compile your program:

```
gcc *.c –o social_network –Wall -Werror
```

This command will take care of all the extra .c files that you have included.
Note the –Werror flag. This flag will treat all your warnings as errors. So, make sure that your program has no warnings, otherwise it won't compile.

TA's Implementation
The executable version of TA's implementation for this program can be accessed as follows:

- Login to rlogin cluster to your home directory
- Use the following command to copy the executable to your current directory:
  cp /home/courses/cs2506/fall2012/project3/social_network .
- To run the program use the following command:
  ./social_network

In case of doubts with respect to the project specification, use this implementation to clarify how the program should behave in a particular situation. If for some reason you think that the TA's implementation is not behaving as the per specifications, please mention those on Piazza under the thread "Project 3 TA's Implementation".

Grading
More information about the grading will be published later along with the auto-grading scripts.

There will be some points for error handling. So, make sure that your program doesn't fail if the user enters some erroneous input at any stage of the program. Rather handle the error and output an error message.

Submission Deadlines
The project is due by 11:55 pm on November 14th. The late submission guidelines are as follows:

- You can submit till 10:00 am on November 15th with 10% penalty.
- You can submit till 11:55 pm on November 16th with 30% penalty.
- No submissions are allowed after that.

Submission Guidelines

- Put all your .c and .h files in a .tar or .zip file and name that file project3.tar or project3.zip
- Submit just 1 file on scholar: project2.tar/project2.zip
- Make sure that your files are actually submitted on scholar and you received a confirmation email. Strictly no submissions will be accepted over the email.

Suggestions

- See the linked_list.c file to see how a linked list is implemented. I used the same file in the C lecture
- Search online for the implementation of adjacency lists to see how to use them.

!!! All the Best !!!