Electrical & Computer Engineering Department, University of Maryland, College Park

Spring 2020 **ENEE150** Dr. Gang Qu

Project 3: My Tiny Social Network Graph

Discussed: Wednesday, April 15, 2020
Posted: Wednesday, April 15, 2020
Due date: Friday 11:59 pm May 8, 2020

Project Objectives

1. Learn separate compilation and linking for large project
2. Master basic (singly) linked list operations: build, search, insert, delete.
3. Understand and implement some basic graph algorithms
4. Master the concepts of pointers, pointer array, and dynamic memory allocation.

Project Description

This project is an advanced version of the previous project on the design of a simple social network called My Tiny Social Network (MyTSN). Updates and new requirements are highlighted in this document. The input file will have the same format. You will be given several files including one that contains the `main()` function and a `MyTSN.h` file that has all the function prototypes required for by the `main()` function. Your assignment is to implement all the functions defined in `MyTSN.h` in a file called `MyTSN.c` so the `main()` function can run correctly. You are required to use (1) a given structure to represent each user (the structure contains user ID, user password, the number of friends, and a root pointing to the head of the list of friends) and (2) singly linked lists to represent the MyTSN database as a graph.

General Requirements

For this project, your program should run with the following command:

```
a.out user_database.txt output.txt
```

where `a.out` is the compiled executable file (how to generate this executable will be discussed in the class and posted in the Q_and_A.txt), `user_database.txt` is the name of the file that has the information of all the current users of MyTSN, `output.txt` is the file of the output and the updated user information after the execution of the program. You should do a safety check to ensure that you have the correct number of command line arguments and the files are valid.

Input file:

Each user in MyTSN has a unique 6-digit user ID. Each line in the input file will have one of the following two formats (ignore the <>):

```
<UID>#
<UID1> <UID2>
```

The first one indicates the ID of a user. The `#` sign is not part of the ID, it just indicates the end of an ID. An ID has exactly 6 digits. If it has less than 6 digits, you should pad with 0's. For example, `123` should be printed out both on the screen and output file as `000123`. This also applies to the input file and the user input from the keyboard. Each user must have a line like this to register in MyTSN. The second one indicates a friend relationship between two users. A user may have multiple friends and some users may not have any friends. The size of the input file is unknown.

Output file:

This file stores the information after the updates made by the user during the execution of the program. Use the following format:

```
<UID>: <password>, <number_of_friends>, <UID1>, <UID2>, ... <UIDk>.
Connections of pairs of users:
1. <UID1_1>, <UID>, ..., <UID1_2>
2. <UID2_1>, <UID>, ..., <UID2_2>
3. <UID3_1> and <UID3_2> cannot be connected.
4. ...
```

The first part of the output file is the updated user information in the MyTSN database, similar to that in the previous project. The second part reports the connections of pairs of users `(UID1_1, UID1_2), (UID2_1, UID2_2),...` that your program has been asked to find. More details of this part will be explained next in the "description of the program".

Description of the program:

Once the program is running, you should read in the user information from the input file and generate a random password for each user (see the description of choice 4 for more details on the requirements of password) , then print out the following main menu:

```
Welcome to MyTSN!

1: search for a user
2: list a user's friends
3: add a new friend to a user
4: add a new user
5: remove a friendship
6: remove a user
7: find a connection
0: EXIT

Enter your choice (0-7):
```

Your program will read in the choice (user request), process the request accordingly (see below for the detailed elaboration), and then display this main menu for the next request until user enters `0` to terminate the program.

`1: search for a user`

On this option, your program should prompt the user to enter a user ID and print out one of the following two messages based on whether the user ID is found or not, where `<UID>` is the user ID entered by the user.

```
User <UID> found.
User <UID> not found.
```

`2: list a user's friends`

On this option, your program should first prompt the user to enter a user ID,

```
Enter the user ID:
```

and then print out the user ID of all the friends for this user.

```
User <UID>’s friends: <UID1>, <UID2>, ... <UIDk>.
```

Note that the friend UIDs are separated by a ‘`,`’ followed by a space, except that the last friend’s UID will followed by a ‘`.`’. The k friend UIDs can be listed in any order. It is ok to print out “`friends`” when k=1. If a user does not have any friend, print out

```
User <UID> has no friends.
```

You can assume that the UID is valid (i.e., it consists of only 0-9 and no more than 6 of them), but you need to check whether the UID entered already exists in the MyTSN database. If not, print out the following message and prompt the user to enter another UID.

```
User <UID> does not exist.
Enter the user ID:
```

If the user fails to enter an existing UID in three attempts, your program should bring the user back to the main menu.

`3: add a new friend to a user`

On this option, your program should prompt the user to enter two user IDs. You can assume that the two UIDs are both valid and different (one cannot be a friend of himself/herself).

```
Enter the user IDs of the two friends:
```

Add in the MyTSN database that these two users have become friends and bring the user to the main menu. If the two users are already friends, print out the following message.

```
Users <UID1> and <UID2> are already friends.
```

`4: add a new user`

On this option, your program should prompt the user to enter a user ID

```
Enter the user ID:
```

You need to check the UID to ensure that it has only digits 0-9 and no more than 6 digits, and the UID is not already in the MyTSN database. If there is any violation, print out one of the followings.

```
<UID> can only have up to 6 digits.
<UID> already exists.
```

If the user fails to enter a valid UID in three attempts, your program should bring the user back to the main menu.

When a valid UID is entered, you need to generate a random password for this new user. A user password must meet all the following requirements: (i) 8-16 characters (ii) at least one digit, (iii) at least one upper case letter, (iv) at least one lower case letter, (v) at least one of the following eight special characters: ~, !, @, #, $, %, &, *. You do not have to check whether different users are assigned the same password.

`5: remove a friendship`

On this option, your program should prompt the user to enter two UIDs, you can assume that the two UIDs entered are both valid and different.

```
Enter the user IDs of the two friends:
```

Then you need to check whether the two UIDs are friends or not based on the MyTSN database. If they are not, print out the following message and bring the user to the main menu

```
Users <UID1> and <UID2> are not friends.
```

If they are, remove their friendship from the database, print out the following message and bring the user to the main menu.

```
Users <UID1> and <UID2> are no longer friends.
```

`6: remove a user`

On this option, your program should prompt the user to enter the UID of the user to be removed from the MyTSN database. You can assume that the UID is valid. Then remove the user and his/her friendship information from the MyTSN database, print out the following message and bring the user to the main menu.

```
Users <UID> has been removed from MyTSN.
```

`7: find a connection`

On this option, your program should prompt the user to enter two user IDs. You can assume that the two UIDs are both valid and different.

```
Enter the user IDs of the two users to be connected:
```

Then you need to find a connection for these two users (we will learn how to solve this problem as the shortest path between two nodes in a graph). The results will be reported in the output file.

`0: EXIT`

On this option, your program should write the current MyTSN user information into the output file with the format described earlier, print out the following good-bye message and terminate.

```
Thank you for using MyTSN!
```

Submission Requirements:

1. The project must be implemented using C under the GLUE UNIX system in a file named `MyTSN.c` which should be submitted with the follow command

```
submit 2020 spring enee 150 010x 3 MyTSN.c
```

2. Your submission must conform to the expectations outlined above for program execution and input/output files.
3. A `p3.c` file will be posted in the GLUE public directory, you must implement all the functions specified in `MyTSN.h` such that `p3.c` can be compiled and executed correctly.
4. A sequence of p3_xx.c codes will be posted in the GLUE public directory to guide you on how to complete the project as well as the structures and functions defined in `MyTSN.h`.
5. We will use the posted `MyTSN.h` and `p3.c` to grade your project. Thus, you CANNOT make any changes to these two files.
6. Code `MyTSN.c` must be reasonably commented to allow a user to follow the program.

Grading Criteria:

Correct compilation

Correct execution

Correct file operations (read and write).

Correct implementation of all the functions in `MyTSN.h`

Correct use of the structure defined in `MyTSN.h`

Coding style and documentation

Files that cannot be compiled will be given a 0.

Late penalty is 20% per day. Submissions more than two days late will not be accepted.