Assignment: Data Structures for a Social Media Platform

ELEC 278: Fundamentals of Information Structures — Fall 2024

Objective: To apply your knowledge of data structures and algorithms to design and implement key functionalities of a social media platform.

Instructions: For each prompt, design and implement the required functionality using appropriate data structures. Provide a brief explanation of why you chose each data structure.

Deliverables:

1. **Report.pdf**: Report outlining your design choices. For each prompt (as applicable), provide a pros-and-cons table briefly outlining your data structure design choices and outlining the data structure options you compared. Briefly explain the run-time complexity of your algorithms.
2. **functions.c**: A C file with your functions.
3. **functions.h**: A header C file with your struct definitions and function names.

Handouts:

1. **functions.c**: A starter C file.
2. **functions.h**: A starter header C file.
3. **test_cases.csv**: A CSV file with rows of function calls to test your code's functionality.
4. **main.c**: A C file which parses the CSV file and calls your functions.
5. **output.txt**: A txt file showing the expected output of running main.c on test_cases.csv.
6. **tasks.json**: A VSCode configuration file showing how `"args"` should be configured to link your files to run main.c.
7. **CMakeLists.txt**: A CMake file showing how to link your files to build and run main.c.

Notes:

1. Feel free to add print statements in your code to test your code when developing the code. However, make sure to remove all unnecessary print statements before submitting. Make sure your program only and precisely prints what is required.
2. Feel free to include test cases to demonstrate the correctness of your implementations in the `main()` function in the main.c file or the test_cases.csv file.
3. Use the **same exact** function and struct names as listed in the code blocks in the assignment prompts below and in the functions.c and functions.h files.
4. Make sure there are no typos in your function names as your code will be auto-graded. A grader main.c file will validate your functions against a larger set of test cases. You can use the provided main.c to verify.

5. You can add additional utility or helper functions or additional data structures to implement your functionalities.
6. You can add any attributes to any struct to ease your writing of the code.
7. The report should be single-spaced, 11-13 font, and up to (and including) 6 pages.
8. The deliverables will be cross-checked for plagiarism and use of generative AI.

Deadline: December 3rd, 2024 at 11:59PM.

Submission: Submit your deliverables as separate files to the assignment dropbox on onQ.

1 User Profiles Management

Prompt: Design and implement a data structure to manage user profiles. Each user has a unique user ID, name, email, and a list of friends.

Constraints:

- The user ID fits into an integer type variable.
- The name and email are strings of at most 49 characters
- The social media platform can handle up to a 10,000 users.
- Users can be deleted.
- Users can change their name and/or email.
- Users' names and emails are unique.

```
User* create_user(const char* name, const char* email); // int user_id is auto-
    generated to be unique
void add_friend(User* user1, User* user2); // users user1 and user2 are now
    friends
void delete_friend(User* user1, User* user2); // users user1 and user2 are no
    longer friends
void delete_user(User* user); // user is deleted
void print_users(); // prints all user names in ascending order
void change_user_name(User* user, char* new_name);
void change_user_email(User* user, char* new_email);
void print_friends(User* user); // prints user's friends' names in ascending order

User* user1 = create_user("Alise", "alise@example.com");
User* user2 = create_user("Ahmed", "ahmed@example.com");
User* user3 = create_user("Mehmet", "mehmet@example.com");
User* user4 = create_user("Viktor", "viktor@example.com");
print_users(); // prints "Ahmed,Alise,Mehmet,Victor\n"
add_friend(user1, user2); // Alise and Ahmed are now friends
add_friend(user3, user2); // Mehmet and Ahmed are now friends
print_friends(user2); // prints Ahmed's friends: "Alise,Mehmet\n"
delete_user(user4); // Viktor is no longer a user
change_user_name(user1, "Alice"); // Alise changed her user name to Alice
change_user_email(user1, "alice@example.com");
```

2 Search Functionality

Prompt: Design and implement a search functionality that allows users to search for other users by name or email.

```
User* search_user_by_name(const char* name);
User* search_user_by_email(const char* email);

User* user = search_user_by_name("Ahmed");
// or
// User* user = search_user_by_email("alice@example.com");
if (user != NULL) {
    printf("Found user: %s, Email: %s\n", user->name, user->email);
} else {
    printf("User not found\n");
}
```

Notice that the data structure managing all users is global. Hence, it not passed as an argument into the functions.

3 Mutual Friends

Prompt: Design and implement a functionality to retrieve the mutual friends between two users.

```
User** mutual_friends(User* user1, User* user2); // returns an array of pointers
    to the mutual friends
void print_mutual_friends(User** friends); // prints mutual friends' user names in
     acsending order

print_mutual_friends(mutual_friends(user1, user3)); // prints "Ahmed\n"
add_friend(user1, user3);
print_mutual_friends(mutual_friends(user1, user3)); // prints "Ahmed\n"
```

4 Messaging System

Prompt: Design and implement a messaging system where users can send and receive messages. Each message has a unique message ID, sender, receiver, and content.

Constraints:

- Messages are sent from one sender to one receiver with no functionality for group communication.
- Each message is at most 256 characters long.
- The message ID fits in an integer type variable.
- Each chat between 2 users will have at most 50 messages. Delete oldest message and keep the 50 most recent messages.
- Users cannot delete messages.
- Only friends can message each other.
- No escape sequences will be in the messages.

```
Message* create_message(User* sender, User* receiver, const char* content); // int
     message_id is auto-generated to be unique
```

```
void print_message(Message* message);
void display_chat(User* user1, User* user2); // print messages in FIFO

Message* msg = create_message(user1, user2, "Hi Ahmed!");
print_message(msg); // prints "Hi Ahmed!\n"
Message* msg = create_message(user2, user1, "Hello Alice!");
Message* msg = create_message(user2, user1, "How are you?");
display_chat(user1, user2); // prints "[Alice:]Hi Ahmed!,[Ahmed:]Hello Alice!,[
    Ahmed:]How are you?\n"
```

5 Posts

Prompt: Design and implement a functionality to collect new posts made by users. Each post has a unique post ID, user (who posted the post), number of likes, and content.

Constraints:

- Each post is at most 256 characters long.
- Users cannot delete posts.
- No escape sequences will be in the posts.

```
Post* new_post(User* user, const char* content); // post id is auto-generated to
    be unique

Post* post1 = new_post(user1, "Hello");
Post* post2 = new_post(user2, "It's me");
Post* post3 = new_post(user3, "I've been wondering if");
Post* post4 = new_post(user1, "after all these years");
Post* post5 = new_post(user2, "you'd like to meet?");
```

6 Like System

Prompt: Design and implement a system to manage likes on posts.

Constraints:

- The number of likes on a post fits into an integer type variable.
- A user can like their own post.
- A post cannot be liked multiple times by the same user.

```
void add_like(Post* post, User* user); // user is the individual who liked the
    post

add_like(post3, user2); // post3 has 1 like
add_like(post4, user2);
add_like(post4, user3);
add_like(post4, user3); // post4 has 2 likes
add_like(post5, user2); // post5 has 1 like
```

7 Feed Generation

Prompt: Design and implement a data structure to generate and display a user's feed, which consists of posts made by themselves and their friends, prioritized by number of likes and then

recency (oldest shows first on the feed).

Constraints:

- A user's feed will contain at most 20 posts. Delete older posts to maintain the most recent 20 posts.

```
void display_feed(User* user1);

display_feed(user1); // all three users are friends
// prints "[Alice:]after all these years,[Mehmet:]I've been wondering if,[Ahmed:]
    you'd like to meet?,[Alice:]Hello,[Ahmed:]It's me\n"
```

8 Marking Scheme

As this course is about data structures, the marking will be based on the description and implementation of the data structures and how your functions use them.

There are four key data structures to manage Users, Friends, Messages and Posts. For the report, each of the four data structures will be marked out of 5 as:

- Description of core data structure (1 mark) - which data structure did you use (linked list, tree, hashtable, queue, etc.) and how did you customize it for this application.
- Description of tradeoffs (1 mark). Why did you choose this data structure? What is the complexity of the data structure you chose?
- Description of the fields individual structure used to represent single entities (0.5 marks). For example, the description of the fields of the User structure used to represent individual users
- How is the data structure modified by the functions we asked you to implement (1.5 marks). For exmaple, how does the change email function modify the data structure?
- How is the data structure searched and or read by the functions we asked you to implement (1 mark)

Thus the report is worth 20 marks.

You may choose to implement all of them separately or combined two or more into a single data structure. In this case you would describe both of them separately but also include how they are combined.

The implementation will be marked as 1 mark per function (18 marks total) with 2 marks for the implementation of each data structure for a total of 26 marks for the implementation.