CL-2001 - Data Structures Lab

Project (Maximum 2 Group Members of your choice)

Note: Carefully read the following instructions (*Each instruction contains a weightage*)

1. Use understandable name of variables.
2. Write a code in C++ language.
3. **Submit .cpp files.**
4. Code the problem statement on MS Visual Studio C++ compiler, It is a console based project. No graphics are included.
5. Please submit a zip file in this format **Group members rollno_DSLabProject.**
6. Do not submit your project after deadline. Late and email submission is not accepted.
7. Do not copy code from any source otherwise you will be penalized with **ZERO** marks in the Project.

Project Evaluation Guidelines:

- Each student will undergo an individual practical viva.
- All students must sit in the lab and download their project code onto university systems.
- You will be required to add or update specific modules in your project within a timeframe of **30-45 minutes**.
- **Internet access will be disabled** during the evaluation.
- **No verbal viva** will be conducted for the groups

Mini Instagram

In this project you will have to implement a micro-version Instagram using various types of data structures by implementing them. **Do Not Use any built-in library** of the data Structures like stack, queue etc. Specifically, your program will have a menu. Use name as your ID (Suppose Names are unique). **Use linked list-based implementation of each data structure**.

1. **A User Profile:**
    a. **Node Data**: Include additional user attributes, such as:
        i. Name (unique identifier)
        ii. Password (for login)
        iii. City
        iv. posts (data & time, text content)
        v. Last login timestamp (to display active status)
    b. **Edge (Friend Relationship) Data**:
        i. Relation type (friend)
        ii. Status (active, blocked, pending request)

2. **Users:**
    a. Use a **graph** to represent **users** and their relationships. Each user is a node, and each friendship or connection is an edge.
    b. For following requests, add a new node in the adjacency list and manage pending requests with the **queue**.
    c. Use **Hash** lookup for username and password verification for login.
    d. If the network grows, implement graph traversal optimizations to manage friend suggestions and mutual friends more effectively.
    e. Make an **Admin User**, who can delete/update other users' info.

3. **Messages**:
    a. Use a stack for each conversation between users, where each user has a separate message stack with each friend.
    b. The latest message is always at the top, allowing easy retrieval of recent messages.

4. **Posts**:
    a. Use a stack for each post, where each user has a separate post stack for his newsfeed.
    b. Use a stack for each post of his followers, where each user has a separate post stack for his newsfeed.

5. **Friend Request**:
    a. Use a queue to manage friend requests for each user. Requests are processed in a First-In-First-Out (FIFO) order, ensuring that the oldest requests are handled first.

6. **Notification**:
    a. Store notifications (e.g., new friend requests, accepted requests, messages received) in a queue.
    b. This structure allows users to view notifications in the order they were received.
7. **Real-time Messaging**:

    Implement a queue where messages are enqueued upon sending. The receiver can then dequeue messages when they open their inbox, emulating real-timemessaging.
8. **Search Users**:

    Use a **BST** to keep users in **sorted order** by username. Each node in the BST contains the user's username. It allows users to search for another user by its name with **O(log(n))** time complexity. Hint: use **AVL or Heaps**.
9. **Menu Functionalities:**
    a. **Signup:** Validate usernames for uniqueness and enforce password strength rules.
    b. **Login:** Allow password **resets** using **security questions**. Save the old passwords in a file if user repeats the same password again (**File-Handling**).
    c. **Logout:** Allow the user to logout from his account.
    d. **Follow Request:** Show pending requests and allow bulk approvals.
    e. **Cancel and Accept Requests:** Add an option to notify the user when a request is accepted.
    f. **Posts:** Implement a basic timeline view to display posts from followers.
    g. **Notifications:** Display notifications queue of the unread messages, related to follow request...
    h. **Messaging:** Allow users to message to whom they are following.
    i. **Search Users:** Display the list of searched users using **BST** traversal.
    j. **Followers List:** Display the user's follower's list
    k. **Newsfeed:** Display user's own posts.
10. **Scalability and Modular Design**:

    Split functionalities into separate functions, classes (e.g., User, Followers, Message) and **must follow Three-File Structure format at least**.
11. **Robust Error Messages**:

    Handle invalid input using assert functions and Provide clear error messages (e.g., "Friend request already sent" or "Invalid username/password").

Best of luck ☺