```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

typedef struct Post {
char username[50];
char content[200];
char timestamp[50];
struct Post* next;
} Post;
typedef struct Friend {
char friendName[50];
struct Friend* next;
} Friend;
typedef struct User {
int userID;
char username[50];
char password[50];
int isAdmin;
Post* posts;
Friend* friends;
struct User* next;   // Linked List of users
} User;
typedef struct Profile {
int userID;
char username[50];
char bio[100];
struct Profile* left, *right;  // BST
} Profile;
```

```
// Queue for Friend Requests / Messages
typedef struct QueueNode {
int senderID, receiverID;
char message[200];
struct QueueNode* next;
} QueueNode;
typedef struct {
QueueNode *front, *rear;
} Queue;
// Stack for Undo
typedef struct StackNode {
char actionType[20];   // "POST" / "FRIEND" / "MESSAGE"
int senderID, receiverID;
char content[200];
struct StackNode* next;
} StackNode;
// ---------- Globals ----------
User* users = NULL;
Profile* profiles = NULL;
User* loggedInUser = NULL;
Queue friendRequests = {NULL, NULL};
Queue messages = {NULL, NULL};
StackNode* undoStack = NULL;
// ---------- Utility ----------
char* getTimestamp() {
static char buffer[50];
time_t now = time(NULL);
struct tm* t = localtime(&now);
strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", t);
return buffer;
}
```

```
User* findUserByID(int id) {
for (User* u = users; u; u = u->next)
if (u->userID == id) return u;
return NULL;
}
User* findUserByName(const char* name) {
for (User* u = users; u; u = u->next)
if (strcmp(u->username, name) == 0) return u;
return NULL;
}
// ---------- Queue ----------
void enqueue(Queue* q, int sender, int receiver, const char* msg) {
QueueNode* node = malloc(sizeof(QueueNode));
node->senderID = sender;
node->receiverID = receiver;
strcpy(node->message, msg);
node->next = NULL;
if (q->rear) q->rear->next = node;
else q->front = node;
q->rear = node;
}
QueueNode* dequeue(Queue* q) {
if (!q->front) return NULL;
QueueNode* temp = q->front;
q->front = temp->next;
if (!q->front) q->rear = NULL;
return temp;
}
// ---------- Stack ----------
void pushUndo(const char* action, int sender, int receiver, const char* content) {
StackNode* node = malloc(sizeof(StackNode));
```

```
strcpy(node->actionType, action);
node->senderID = sender;
node->receiverID = receiver;
strcpy(node->content, content);
node->next = undoStack;
undoStack = node;
}
StackNode* popUndo() {
if (!undoStack) return NULL;
StackNode* temp = undoStack;
undoStack = temp->next;
return temp;
}
// ---------- BST ----------
Profile* insertProfile(Profile* root, int id, const char* username, const char* bio) {
   if (!root) {
      Profile* node = malloc(sizeof(Profile));
      node->userID = id;
      strcpy(node->username, username);
      strcpy(node->bio, bio);
      node->left = node->right = NULL;
      return node;
   }
   if (id < root->userID) root->left = insertProfile(root->left, id, username, bio);
   else if (id > root->userID) root->right = insertProfile(root->right, id, username, bio);
   return root;
}
Profile* searchProfile(Profile* root, int id) {
   if (!root || root->userID == id) return root;
   if (id < root->userID) return searchProfile(root->left, id);
   return searchProfile(root->right, id);
```

```
}

// -------------------- Users --------------------
void registerUser() {
    User* newUser = malloc(sizeof(User));
    newUser->userID = rand() % 10000;
    printf("Enter username: ");
    scanf("%s", newUser->username);
    printf("Enter password: ");
    scanf("%s", newUser->password);
    newUser->isAdmin = 0;
    newUser->posts = NULL;
    newUser->friends = NULL;
    newUser->next = users;
    users = newUser;

    profiles = insertProfile(profiles, newUser->userID, newUser->username, "Hello, I am new!");
    printf("User '%s' registered (ID=%d)\n", newUser->username, newUser->userID);
}

void deleteUser() {
    if (!loggedInUser) { printf("Login first!\n"); return; }
    User *prev = NULL, *cur = users;
    while (cur) {
        if (cur == loggedInUser) {
            if (prev) prev->next = cur->next;
            else users = cur->next;
            free(cur);
            loggedInUser = NULL;
            printf("Your account has been deleted.\n");
            return;
```

```
        }
        prev = cur;
        cur = cur->next;
    }
}

void loginUser() {
    char uname[50], pwd[50];
    printf("Enter username: ");
    scanf("%s", uname);
    printf("Enter password: ");
    scanf("%s", pwd);

    User* u = users;
    while (u) {
        if (strcmp(u->username, uname) == 0 && strcmp(u->password, pwd) == 0) {
            loggedInUser = u;
            printf("Login successful! Welcome %s\n", u->username);
            return;
        }
        u = u->next;
    }
    printf("Invalid login!\n");
}

// -------------------- Posts --------------------
void createPost() {
    if (!loggedInUser) { printf("Login first!\n"); return; }

    Post* newPost = malloc(sizeof(Post));
    getchar();
```

```
    printf("Enter post: ");
    fgets(newPost->content, sizeof(newPost->content), stdin);
    newPost->content[strcspn(newPost->content, "\n")] = 0;
    strcpy(newPost->username, loggedInUser->username);
    strcpy(newPost->timestamp, getTimestamp());
    newPost->next = loggedInUser->posts;
    loggedInUser->posts = newPost;

    pushUndo("POST", loggedInUser->userID, -1, newPost->content);
    printf("Post created!\n");
}

void showAllPosts() {
    printf("\n=== All Posts ===\n");
    for (User* u = users; u; u = u->next) {
        for (Post* p = u->posts; p; p = p->next) {
            printf("[%s] %s: %s\n", p->timestamp, p->username, p->content);
        }
    }
    printf("==================\n");
}

// -------------------- Friends --------------------
void sendFriendRequest() {
    if (!loggedInUser) { printf("Login first!\n"); return; }
    char name[50];
    printf("Enter username to send request: ");
    scanf("%s", name);
    User* target = findUserByName(name);
    if (!target) { printf("User not found!\n"); return; }
    enqueue(&friendRequests, loggedInUser->userID, target->userID, "FRIEND_REQUEST");
```

```
    printf("Friend request sent to %s\n", target->username);
}

void acceptFriendRequest() {
    if (!loggedInUser) { printf("Login first!\n"); return; }
    QueueNode* req = dequeue(&friendRequests);
    if (!req || req->receiverID != loggedInUser->userID) {
printf("No friend requests!\n");
return;
}
User* sender = findUserByID(req->senderID);
if (!sender) return;
// add to each other's friend list
Friend* f1 = malloc(sizeof(Friend));
strcpy(f1->friendName, sender->username);
f1->next = loggedInUser->friends;
loggedInUser->friends = f1;
Friend* f2 = malloc(sizeof(Friend));
strcpy(f2->friendName, loggedInUser->username);
f2->next = sender->friends;
sender->friends = f2;
pushUndo("FRIEND", sender->userID, loggedInUser->userID, sender->username);
printf("You are now friends with %s!\n", sender->username);
free(req);
}
void showFriends() {
if (!loggedInUser) { printf("Login first!\n"); return; }
printf("Friends of %s: ", loggedInUser->username);
for (Friend* f = loggedInUser->friends; f; f = f->next)
printf("%s ", f->friendName);
printf("\n");
```

```
}

// -------------------- Messaging --------------------
void sendMessage() {
    if (!loggedInUser) { printf("Login first!\n"); return; }
    char name[50], msg[200];
    printf("Send message to: ");
    scanf("%s", name);
    getchar();
    User* target = findUserByName(name);
    if (!target) { printf("User not found!\n"); return; }
    printf("Enter message: ");
    fgets(msg, sizeof(msg), stdin);
    msg[strcspn(msg, "\n")] = 0;
    enqueue(&messages, loggedInUser->userID, target->userID, msg);
    pushUndo("MESSAGE", loggedInUser->userID, target->userID, msg);
    printf("Message sent!\n");
}

void readMessages() {
    if (!loggedInUser) { printf("Login first!\n"); return; }
    int found = 0;
    QueueNode* prev = NULL, *cur = messages.front;
    while (cur) {
        if (cur->receiverID == loggedInUser->userID) {
            User* sender = findUserByID(cur->senderID);
            printf("From %s: %s\n", sender ? sender->username : "Unknown", cur->message);
            if (prev) prev->next = cur->next;
            else messages.front = cur->next;
            free(cur);
            found = 1;
```

```
            break;
        }
        prev = cur;
        cur = cur->next;
    }
    if (!found) printf("No new messages!\n");
}

// -------------------- Undo --------------------
void undoAction() {
    StackNode* act = popUndo();
    if (!act) { printf("Nothing to undo!\n"); return; }

    if (strcmp(act->actionType, "POST") == 0) {
        User* u = findUserByID(act->senderID);
        if (u) {
            Post* prev = NULL, *cur = u->posts;
            while (cur) {
                if (strcmp(cur->content, act->content) == 0) {
                    if (prev) prev->next = cur->next;
                    else u->posts = cur->next;
                    free(cur);
                    printf("Undid last post.\n");
                    break;
                }
                prev = cur;
                cur = cur->next;
            }
        }
    }
    else if (strcmp(act->actionType, "MESSAGE") == 0) {
```

```
        printf("Undid message: %s\n", act->content);
    }
    else if (strcmp(act->actionType, "FRIEND") == 0) {
        printf("Undid last friend action.\n");
    }
    free(act);
}

// ---------- Menu ----------
void menu() {
    int choice;
    do {
        printf("\n--- Social Media Simulator ---\n");
        printf("1. Register\n");
        printf("2. Login\n");
        printf("3. Delete My Account\n");
        printf("4. Create Post\n");
        printf("5. Show All Posts\n");
        printf("6. Send Friend Request\n");
        printf("7. Accept Friend Request\n");
        printf("8. Show Friends\n");
        printf("9. Send Message\n");
        printf("10. Read Messages\n");
        printf("11. Undo Last Action\n");
        printf("0. Exit\n");
        printf("Choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: registerUser(); break;
            case 2: loginUser(); break;
```

```
            case 3: deleteUser(); break;
            case 4: createPost(); break;
            case 5: showAllPosts(); break;
            case 6: sendFriendRequest(); break;
            case 7: acceptFriendRequest(); break;
            case 8: showFriends(); break;
            case 9: sendMessage(); break;
            case 10: readMessages(); break;
            case 11: undoAction(); break;
            case 0: printf("Bye!\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while (choice != 0);
}

int main() {
    srand(time(NULL));
    menu();
        return 0;
}
```