#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

#define MAX\_LENGTH 200

#define MAX\_USERS 50

#define MAX\_FOLLOWING 20

#define MAX\_LIKES 1000

#define DATA\_FILE "social\_media.dat"

// Structure for comments

typedef struct Comment {

int id;

char author[50];

char message[MAX\_LENGTH];

time\_t timestamp;

struct Comment \*parent; // For threading/replies

struct Comment \*replies; // Linked list of replies

struct Comment \*next; // Next comment at same level

} Comment;

// Structure for posts

typedef struct Post {

int id;

char author[50];

char message[MAX\_LENGTH];

int likes;

char likedBy[MAX\_LIKES][50]; // Track who liked the post

int likeCount;

time\_t timestamp;

time\_t lastEditTime; // Track when post was last edited

Comment \*comments; // Linked list of comments

struct Post \*next;

} Post;

// Structure for User

typedef struct User {

char username[50];

char password[50];

char bio[MAX\_LENGTH];

int followingCount;

char following[MAX\_FOLLOWING][50];

} User;

// Global variables

Post \*head = NULL;

int postCount = 0;

int commentCount = 0;

User users[MAX\_USERS];

int userCount = 0;

char loggedInUser[50] = "";

// Utility functions

void clearInputBuffer() {

int c;

while ((c = getchar()) != '\n' && c != EOF);

}

void trimWhitespace(char \*str) {

int i = strlen(str) - 1;

while (i >= 0 && isspace(str[i])) {

str[i--] = '\0';

}

}

void printTimestamp(time\_t timestamp) {

char buffer[26];

struct tm\* tm\_info = localtime(&timestamp);

strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm\_info);

printf("%s", buffer);

}

// User management

int findUserIndex(const char \*username) {

for (int i = 0; i < userCount; i++) {

if (strcmp(users[i].username, username) == 0) {

return i;

}

}

return -1;

}

void registerUser() {

if (userCount >= MAX\_USERS) {

printf("User limit reached!\n");

return;

}

User newUser;

printf("Enter username: ");

fgets(newUser.username, 50, stdin);

trimWhitespace(newUser.username);

if (findUserIndex(newUser.username) != -1) {

printf("Username already exists!\n");

return;

}

printf("Enter password: ");

fgets(newUser.password, 50, stdin);

trimWhitespace(newUser.password);

printf("Enter bio (optional): ");

fgets(newUser.bio, MAX\_LENGTH, stdin);

trimWhitespace(newUser.bio);

newUser.followingCount = 0;

users[userCount++] = newUser;

printf("User registered successfully!\n");

}

int loginUser() {

if (strlen(loggedInUser)) {

printf("You are already logged in as %s\n", loggedInUser);

return 1;

}

char username[50], password[50];

printf("Enter username: ");

fgets(username, 50, stdin);

trimWhitespace(username);

printf("Enter password: ");

fgets(password, 50, stdin);

trimWhitespace(password);

for (int i = 0; i < userCount; i++) {

if (strcmp(users[i].username, username) == 0 &&

strcmp(users[i].password, password) == 0) {

strcpy(loggedInUser, username);

printf("Welcome back, %s!\n", username);

return 1;

}

}

printf("Invalid username or password!\n");

return 0;

}

void logoutUser() {

if (strlen(loggedInUser)) {

printf("Goodbye, %s!\n", loggedInUser);

loggedInUser[0] = '\0';

} else {

printf("No user is currently logged in.\n");

}

}

void viewProfile(const char \*username) {

int userIdx = findUserIndex(username);

if (userIdx == -1) {

printf("User not found!\n");

return;

}

User \*user = &users[userIdx];

printf("\n=== %s's Profile ===\n", username);

printf("Bio: %s\n", user->bio);

printf("Following: %d\n", user->followingCount);

// Count user's posts

int postNum = 0;

Post \*current = head;

while (current) {

if (strcmp(current->author, username) == 0) postNum++;

current = current->next;

}

printf("Posts: %d\n", postNum);

printf("===================\n");

}

void followUser() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

char username[50];

printf("Enter username to follow: ");

fgets(username, 50, stdin);

trimWhitespace(username);

if (strcmp(loggedInUser, username) == 0) {

printf("You can't follow yourself!\n");

return;

}

if (findUserIndex(username) == -1) {

printf("User not found!\n");

return;

}

int userIdx = findUserIndex(loggedInUser);

User \*user = &users[userIdx];

// Check if already following

for (int i = 0; i < user->followingCount; i++) {

if (strcmp(user->following[i], username) == 0) {

printf("You are already following %s!\n", username);

return;

}

}

if (user->followingCount >= MAX\_FOLLOWING) {

printf("You can't follow more users!\n");

return;

}

strcpy(user->following[user->followingCount++], username);

printf("You are now following %s!\n", username);

}

// Post and Comment management

Post\* findPost(int postId) {

Post \*current = head;

while (current) {

if (current->id == postId) {

return current;

}

current = current->next;

}

return NULL;

}

Comment\* findComment(Comment \*comments, int commentId) {

if (!comments) return NULL;

if (comments->id == commentId) {

return comments;

}

Comment \*found = findComment(comments->replies, commentId);

if (found) return found;

return findComment(comments->next, commentId);

}

void addPost() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

Post \*newPost = (Post \*)malloc(sizeof(Post));

if (!newPost) {

printf("Memory allocation failed!\n");

return;

}

printf("Enter your post message (max %d chars):\n", MAX\_LENGTH);

fgets(newPost->message, MAX\_LENGTH, stdin);

trimWhitespace(newPost->message);

if (strlen(newPost->message) == 0) {

printf("Post cannot be empty!\n");

free(newPost);

return;

}

newPost->id = ++postCount;

strcpy(newPost->author, loggedInUser);

newPost->likes = 0;

newPost->likeCount = 0;

newPost->timestamp = time(NULL);

newPost->lastEditTime = 0;

newPost->comments = NULL;

newPost->next = head;

head = newPost;

printf("Post added successfully!\n");

}

void editPost() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

int postId;

printf("Enter Post ID to edit: ");

scanf("%d", &postId);

clearInputBuffer();

Post \*post = findPost(postId);

if (!post) {

printf("Post not found!\n");

return;

}

if (strcmp(post->author, loggedInUser) != 0) {

printf("You can only edit your own posts!\n");

return;

}

printf("Current content: %s\n", post->message);

printf("Enter new content (max %d chars):\n", MAX\_LENGTH);

fgets(post->message, MAX\_LENGTH, stdin);

trimWhitespace(post->message);

if (strlen(post->message) == 0) {

printf("Post cannot be empty!\n");

return;

}

post->lastEditTime = time(NULL);

printf("Post updated successfully!\n");

}

void deletePost() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

int postId;

printf("Enter Post ID to delete: ");

scanf("%d", &postId);

clearInputBuffer();

Post \*current = head, \*prev = NULL;

while (current) {

if (current->id == postId) {

if (strcmp(current->author, loggedInUser) != 0) {

printf("You can only delete your own posts!\n");

return;

}

if (prev) {

prev->next = current->next;

} else {

head = current->next;

}

// Free comments recursively

Comment \*cmt = current->comments;

while (cmt) {

Comment \*temp = cmt;

cmt = cmt->next;

free(temp);

}

free(current);

printf("Post deleted successfully!\n");

return;

}

prev = current;

current = current->next;

}

printf("Post not found!\n");

}

void likePost() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

int postId;

printf("Enter Post ID to like: ");

scanf("%d", &postId);

clearInputBuffer();

Post \*post = findPost(postId);

if (!post) {

printf("Post not found!\n");

return;

}

// Check if already liked

for (int i = 0; i < post->likeCount; i++) {

if (strcmp(post->likedBy[i], loggedInUser) == 0) {

printf("You already liked this post!\n");

return;

}

}

if (post->likeCount >= MAX\_LIKES) {

printf("Maximum likes reached for this post!\n");

return;

}

strcpy(post->likedBy[post->likeCount++], loggedInUser);

post->likes++;

printf("Post liked successfully!\n");

}

void addComment() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

int postId;

printf("Enter Post ID to comment on: ");

scanf("%d", &postId);

clearInputBuffer();

Post \*post = findPost(postId);

if (!post) {

printf("Post not found!\n");

return;

}

Comment \*newComment = (Comment \*)malloc(sizeof(Comment));

if (!newComment) {

printf("Memory allocation failed!\n");

return;

}

printf("Enter your comment:\n");

fgets(newComment->message, MAX\_LENGTH, stdin);

trimWhitespace(newComment->message);

if (strlen(newComment->message) == 0) {

printf("Comment cannot be empty!\n");

free(newComment);

return;

}

newComment->id = ++commentCount;

strcpy(newComment->author, loggedInUser);

newComment->timestamp = time(NULL);

newComment->parent = NULL;

newComment->replies = NULL;

newComment->next = post->comments;

post->comments = newComment;

printf("Comment added successfully!\n");

}

void editComment() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

int postId, commentId;

printf("Enter Post ID containing the comment: ");

scanf("%d", &postId);

printf("Enter Comment ID to edit: ");

scanf("%d", &commentId);

clearInputBuffer();

Post \*post = findPost(postId);

if (!post) {

printf("Post not found!\n");

return;

}

Comment \*comment = findComment(post->comments, commentId);

if (!comment) {

printf("Comment not found!\n");

return;

}

if (strcmp(comment->author, loggedInUser) != 0) {

printf("You can only edit your own comments!\n");

return;

}

printf("Current comment: %s\n", comment->message);

printf("Enter new comment (max %d chars):\n", MAX\_LENGTH);

fgets(comment->message, MAX\_LENGTH, stdin);

trimWhitespace(comment->message);

if (strlen(comment->message) == 0) {

printf("Comment cannot be empty!\n");

return;

}

printf("Comment updated successfully!\n");

}

void addReply() {

if (!strlen(loggedInUser)) {

printf("Please log in first.\n");

return;

}

int postId, parentCommentId;

printf("Enter Post ID containing the comment: ");

scanf("%d", &postId);

printf("Enter Comment ID to reply to: ");

scanf("%d", &parentCommentId);

clearInputBuffer();

Post \*post = findPost(postId);

if (!post) {

printf("Post not found!\n");

return;

}

Comment \*parent = findComment(post->comments, parentCommentId);

if (!parent) {

printf("Comment not found!\n");

return;

}

Comment \*newReply = (Comment \*)malloc(sizeof(Comment));

if (!newReply) {

printf("Memory allocation failed!\n");

return;

}

printf("Enter your reply:\n");

fgets(newReply->message, MAX\_LENGTH, stdin);

trimWhitespace(newReply->message);

if (strlen(newReply->message) == 0) {

printf("Reply cannot be empty!\n");

free(newReply);

return;

}

newReply->id = ++commentCount;

strcpy(newReply->author, loggedInUser);

newReply->timestamp = time(NULL);

newReply->parent = parent;

newReply->replies = NULL;

newReply->next = parent->replies;

parent->replies = newReply;

printf("Reply added successfully!\n");

}

// Print comments with threading

void printComments(Comment \*comments, int depth) {

while (comments) {

// Indent based on depth

for (int i = 0; i < depth; i++) printf(" ");

printf("%d. %s (", comments->id, comments->author);

printTimestamp(comments->timestamp);

printf("): %s\n", comments->message);

// Print replies recursively

if (comments->replies) {

printComments(comments->replies, depth + 1);

}

comments = comments->next;

}

}

// Print post with threaded comments

void printPost(Post \*post) {

printf("\n=== Post #%d ===\n", post->id);

printf("Author: %s\n", post->author);

printf("Posted: ");

printTimestamp(post->timestamp);

if (post->lastEditTime > 0) {

printf("\nEdited: ");

printTimestamp(post->lastEditTime);

}

printf("\nContent: %s\n", post->message);

printf("Likes: %d\n", post->likes);

if (post->likeCount > 0) {

printf("Liked by: ");

for (int i = 0; i < post->likeCount; i++) {

printf("%s", post->likedBy[i]);

if (i < post->likeCount - 1) printf(", ");

}

printf("\n");

}

// Display comments with threading

if (post->comments) {

printf("\nComments:\n");

printComments(post->comments, 0);

}

printf("================\n");

}

void viewFeed(int showAll) {

if (!head) {

printf("No posts available.\n");

return;

}

printf("\n=== Social Media Feed ===\n");

Post \*current = head;

int userIdx = findUserIndex(loggedInUser);

while (current) {

if (showAll || strcmp(current->author, loggedInUser) == 0) {

printPost(current);

} else if (userIdx != -1) {

// Check if following this user

User \*user = &users[userIdx];

for (int i = 0; i < user->followingCount; i++) {

if (strcmp(user->following[i], current->author) == 0) {

printPost(current);

break;

}

}

}

current = current->next;

}

}

void searchPosts() {

char query[MAX\_LENGTH];

printf("Enter search term: ");

fgets(query, MAX\_LENGTH, stdin);

trimWhitespace(query);

if (strlen(query) == 0) {

printf("Please enter a search term.\n");

return;

}

printf("\n=== Search Results ===\n");

Post \*current = head;

int found = 0;

while (current) {

if (strstr(current->message, query) != NULL ||

strstr(current->author, query) != NULL) {

printPost(current);

found = 1;

} else {

// Search in comments

Comment \*cmt = current->comments;

while (cmt) {

if (strstr(cmt->message, query) != NULL ||

strstr(cmt->author, query) != NULL) {

printPost(current);

found = 1;

break;

}

cmt = cmt->next;

}

}

current = current->next;

}

if (!found) {

printf("No matching posts found.\n");

}

}

// Data persistence

void saveToFile() {

FILE \*file = fopen(DATA\_FILE, "wb");

if (!file) {

printf("Failed to open file for saving.\n");

return;

}

// Save user data

fwrite(&userCount, sizeof(int), 1, file);

fwrite(users, sizeof(User), userCount, file);

// Save post count

fwrite(&postCount, sizeof(int), 1, file);

// Save posts

Post \*current = head;

while (current) {

fwrite(current, sizeof(Post), 1, file);

// Save likedBy array

fwrite(current->likedBy, sizeof(char) \* 50, current->likeCount, file);

// Save comments recursively

int commentCount = 0;

Comment \*cmt = current->comments;

while (cmt) {

commentCount++;

cmt = cmt->next;

}

fwrite(&commentCount, sizeof(int), 1, file);

cmt = current->comments;

while (cmt) {

fwrite(cmt, sizeof(Comment), 1, file);

cmt = cmt->next;

}

current = current->next;

}

fclose(file);

printf("Data saved successfully!\n");

}

void loadFromFile() {

FILE \*file = fopen(DATA\_FILE, "rb");

if (!file) {

printf("No saved data found. Starting fresh.\n");

return;

}

// Load user data

fread(&userCount, sizeof(int), 1, file);

fread(users, sizeof(User), userCount, file);

// Load post count

fread(&postCount, sizeof(int), 1, file);

// Load posts

Post \*prev = NULL;

int postNum;

fread(&postNum, sizeof(int), 1, file);

for (int i = 0; i < postNum; i++) {

Post \*newPost = (Post \*)malloc(sizeof(Post));

fread(newPost, sizeof(Post), 1, file);

// Load likedBy array

fread(newPost->likedBy, sizeof(char) \* 50, newPost->likeCount, file);

// Load comments

int commentCount;

fread(&commentCount, sizeof(int), 1, file);

Comment \*cmtPrev = NULL;

for (int j = 0; j < commentCount; j++) {

Comment \*newComment = (Comment \*)malloc(sizeof(Comment));

fread(newComment, sizeof(Comment), 1, file);

if (cmtPrev) {

cmtPrev->next = newComment;

} else {

newPost->comments = newComment;

}

cmtPrev = newComment;

}

if (cmtPrev) cmtPrev->next = NULL;

if (prev) {

prev->next = newPost;

} else {

head = newPost;

}

prev = newPost;

}

if (prev) prev->next = NULL;

fclose(file);

printf("Data loaded successfully!\n");

}

// Main menu

void displayMenu() {

printf("\n=== Social Media Platform ===\n");

if (strlen(loggedInUser)) {

printf("Logged in as: %s\n", loggedInUser);

}

printf("1. Register\n");

printf("2. Login\n");

printf("3. Logout\n");

printf("4. View Profile\n");

printf("5. Follow User\n");

printf("6. Add Post\n");

printf("7. Edit Post\n");

printf("8. Delete Post\n");

printf("9. View Feed (Following)\n");

printf("10. View All Posts\n");

printf("11. Like Post\n");

printf("12. Add Comment\n");

printf("13. Edit Comment\n");

printf("14. Reply to Comment\n");

printf("15. Search Posts\n");

printf("16. Save & Exit\n");

printf("Enter choice: ");

}

int main() {

loadFromFile();

int choice;

while (1) {

displayMenu();

if (scanf("%d", &choice) != 1) {

printf("Invalid input. Please enter a number.\n");

clearInputBuffer();

continue;

}

clearInputBuffer();

switch (choice) {

case 1: registerUser(); break;

case 2: loginUser(); break;

case 3: logoutUser(); break;

case 4: {

char username[50];

printf("Enter username (leave blank for your profile): ");

fgets(username, 50, stdin);

trimWhitespace(username);

if (strlen(username) == 0 && strlen(loggedInUser)) {

strcpy(username, loggedInUser);

}

if (strlen(username)) viewProfile(username);

else printf("Please log in or specify a username.\n");

break;

}

case 5: followUser(); break;

case 6: addPost(); break;

case 7: editPost(); break;

case 8: deletePost(); break;

case 9: viewFeed(0); break;

case 10: viewFeed(1); break;

case 11: likePost(); break;

case 12: addComment(); break;

case 13: editComment(); break;

case 14: addReply(); break;

case 15: searchPosts(); break;

case 16: saveToFile(); exit(0);

default: printf("Invalid choice!\n");

}

}

return 0;

}