#include <iostream>

#include <string>

using namespace std;

const int MAX\_USERS = 100;

const int MAX\_FOLLOWERS = 100;

const int MAX\_REPOSITORIES = 100;

const int MAX\_COMMITS = 100;

const int MAX\_FILES = 100;

class Commit {

private:

string message;

public:

Commit(string msg) {

message = msg;

}

string getMessage() const {

return message;

}

};

class File {

private:

string name;

public:

File(string filename) {

name = filename;

}

string getName() const {

return name;

}

};

class Repository {

private:

string name;

Commit\* commits[MAX\_COMMITS];

File\* files[MAX\_FILES];

int commitCount;

int fileCount;

public:

Repository(string repoName) {

name = repoName;

commitCount = 0;

fileCount = 0;

}

string getName() const {

return name;

}

bool addCommit(Commit\* commit) {

if (commitCount < MAX\_COMMITS) {

commits[commitCount++] = commit;

return true;

}

return false;

}

bool addFile(File\* file) {

if (fileCount < MAX\_FILES) {

files[fileCount++] = file;

return true;

}

return false;

}

const Commit\*\* getCommits() const {

return (const Commit\*\*)commits;

}

const File\*\* getFiles() const {

return (const File\*\*)files;

}

int getCommitCount() const {

return commitCount;

}

int getFileCount() const {

return fileCount;

}

};

class Tree {

private:

struct TreeNode {

Repository\* repository;

TreeNode\* left;

TreeNode\* right;

TreeNode(Repository\* repo) {

repository = repo;

left = nullptr;

right = nullptr;

}

};

TreeNode\* root;

void deleteTree(TreeNode\* node) {

if (node) {

deleteTree(node->left);

deleteTree(node->right);

delete node->repository;

delete node;

}

}

public:

Tree() {

root = nullptr;

}

~Tree() {

deleteTree(root);

}

bool addRepository(Repository\* repository) {

TreeNode\* newNode = new TreeNode(repository);

if (!root) {

root = newNode;

return true;

}

TreeNode\* current = root;

TreeNode\* parent = nullptr;

while (current) {

parent = current;

if (repository->getName() < current->repository->getName()) {

current = current->left;

}

else if (repository->getName() > current->repository->getName()) {

current = current->right;

}

else {

delete newNode;

return false;

}

}

if (repository->getName() < parent->repository->getName()) {

parent->left = newNode;

}

else {

parent->right = newNode;

}

return true;

}

Repository\* searchRepository(const string& repoName) const {

TreeNode\* current = root;

while (current) {

if (repoName == current->repository->getName()) {

return current->repository;

}

else if (repoName < current->repository->getName()) {

current = current->left;

}

else {

current = current->right;

}

}

return nullptr;

}

};

class User {

private:

string username;

string password;

string followers[MAX\_FOLLOWERS];

int followerCount;

public:

User() {

username = "";

password = "";

followerCount = 0;

}

User(string uname, string pwd) {

username = uname;

password = pwd;

followerCount = 0;

}

string getUsername() const {

return username;

}

string getPassword() const {

return password;

}

const string\* getFollowers() const {

return followers;

}

int getFollowerCount() const {

return followerCount;

}

bool addFollower(const string& follower) {

if (followerCount < MAX\_FOLLOWERS) {

followers[followerCount++] = follower;

return true;

}

return false;

}

bool removeFollower(const string& follower) {

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

if (followers[i] == follower) {

for (int j = i; j < followerCount - 1; ++j) {

followers[j] = followers[j + 1];

}

followerCount--;

return true;

}

}

return false;

}

};

class UserRepository {

private:

string name;

string commits[MAX\_FOLLOWERS];

string files[MAX\_FOLLOWERS];

int commitCount;

int fileCount;

public:

UserRepository(string repoName) {

repoName = name;

commitCount = 0;

fileCount = 0;

}

string getName() const {

return name;

}

const string\* getCommits() const {

return commits;

}

const string\* getFiles() const {

return files;

}

int getCommitCount() const {

return commitCount;

}

int getFileCount() const {

return fileCount;

}

bool addCommit(const string& commit) {

if (commitCount < MAX\_FOLLOWERS) {

commits[commitCount++] = commit;

return true;

}

return false;

}

bool addFile(const string& file) {

if (fileCount < MAX\_FOLLOWERS) {

files[fileCount++] = file;

return true;

}

return false;

}

};

class SocialGraph {

private:

string users[MAX\_USERS];

int userCount;

int followMatrix[MAX\_USERS][MAX\_USERS];

public:

SocialGraph() : userCount(0) {

for (int i = 0; i < MAX\_USERS; ++i) {

for (int j = 0; j < MAX\_USERS; ++j) {

followMatrix[i][j] = 0;

}

}

}

bool addUser(const string& username) {

if (userCount < MAX\_USERS) {

users[userCount++] = username;

return true;

}

return false;

}

bool followUser(int followerIndex, int followeeIndex) {

if (followerIndex < userCount && followeeIndex < userCount) {

followMatrix[followerIndex][followeeIndex] = 1;

return true;

}

return false;

}

bool unfollowUser(int followerIndex, int followeeIndex) {

if (followerIndex < userCount && followeeIndex < userCount) {

followMatrix[followerIndex][followeeIndex] = 0;

return true;

}

return false;

}

const string\* getUsernames() const {

return users;

}

int getUserCount() const {

return userCount;

}

const int\* getFollowMatrix() const {

return \*followMatrix;

}

};

class UserManager {

private:

User users[MAX\_USERS];

int userCount;

public:

UserManager() {

userCount = 0;

for (int i = 0; i < MAX\_USERS; ++i) {

users[i] = User();

}

}

bool registerUser(const string& username, const string& password) {

if (userCount < MAX\_USERS) {

users[userCount++] = User(username, password);

return true;

}

return false;

}

User\* loginUser(const string& username, const string& password) {

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

if (users[i].getUsername() == username && users[i].getPassword() == password) {

return &users[i];

}

}

return nullptr;

}

User\* getUser(const string& username) {

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

if (users[i].getUsername() == username) {

return &users[i];

}

}

return nullptr;

}

};

int main() {

UserManager userManager;

SocialGraph socialGraph;

Tree repositoryTree;

while (true) {

system("CLS");

cout << "1. Create Account" << endl;

cout << "2. Login" << endl;

cout << "3. View Profile" << endl;

cout << "4. Follow User" << endl;

cout << "5. Unfollow User" << endl;

cout << "6. Create Repository" << endl;

cout << "7. Add Commit" << endl;

cout << "8. Add File" << endl;

cout << "9. View Repository Details" << endl;

cout << "10. Exit" << endl;

cout << "Choose an option: ";

int choice;

cin >> choice;

if (choice == 1) {

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

if (userManager.registerUser(username, password)) {

cout << "ACCOUNT CREATED!" << endl;

socialGraph.addUser(username);

}

else {

cout << "UNABLE TO CREATE AN ACCOUNT!" << endl;

}

}

else if (choice == 2) {

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

User\* loggedInUser = userManager.loginUser(username, password);

if (loggedInUser) {

cout << "LOGIN SUCCESSFUL!" << endl;

}

else {

cout << "INVALID USERNAME OR PASSWORD! TRY AGAIN" << endl;

}

}

else if (choice == 3) {

string username;

cout << "Enter username: ";

cin >> username;

User\* userProfile = userManager.getUser(username);

if (userProfile) {

cout << "Username: " << userProfile->getUsername() << endl;

cout << "Followers:" << endl;

const string\* followers = userProfile->getFollowers();

for (int i = 0; i < userProfile->getFollowerCount(); ++i) {

cout << "- " << followers[i] << endl;

}

}

else {

cout << "USER NOT FOUND" << endl;

}

}

else if (choice == 4) {

string followerName, followeeName;

cout << "Enter your username: ";

cin >> followerName;

cout << "Enter the username of the user you want to follow: ";

cin >> followeeName;

int followerIndex = -1, followeeIndex = -1;

for (int i = 0; i < socialGraph.getUserCount(); ++i) {

if (socialGraph.getUsernames()[i] == followerName) {

followerIndex = i;

}

if (socialGraph.getUsernames()[i] == followeeName) {

followeeIndex = i;

}

}

if (followerIndex != -1 && followeeIndex != -1) {

if (socialGraph.followUser(followerIndex, followeeIndex)) {

cout << "You are now following " << followeeName << "." << endl;

userManager.getUser(followeeName)->addFollower(followerName);

}

else {

cout << "FAILED TO FOLLOW USER! TRY AGAIN" << endl;

}

}

else {

cout << "INVALID USERNAME! TRY AGAIN" << endl;

}

}

else if (choice == 5) {

string followerName, followeeName;

cout << "Enter your username: ";

cin >> followerName;

cout << "Enter the username of the user you want to unfollow: ";

cin >> followeeName;

int followerIndex = -1, followeeIndex = -1;

for (int i = 0; i < socialGraph.getUserCount(); ++i) {

if (socialGraph.getUsernames()[i] == followerName) {

followerIndex = i;

}

if (socialGraph.getUsernames()[i] == followeeName) {

followeeIndex = i;

}

}

if (followerIndex != -1 && followeeIndex != -1) {

if (socialGraph.unfollowUser(followerIndex, followeeIndex)) {

cout << "You have unfollowed " << followeeName << "." << endl;

userManager.getUser(followeeName)->removeFollower(followerName);

}

else {

cout << "FAILED TO UNFOLLOW USER! TRY AGAIN" << endl;

}

}

else {

cout << "INVALID USERNAME! TRY AGAIN" << endl;

}

}

else if (choice == 6) {

string repoName;

cout << "Enter repository name: ";

cin >> repoName;

Repository\* newRepo = new Repository(repoName);

if (repositoryTree.addRepository(newRepo)) {

cout << "REPOSITORY CREATED!" << endl;

}

else {

cout << "SAME NAME ALREADY EXISTS!" << endl;

delete newRepo;

}

}

else if (choice == 7) {

string repoName, commitMessage;

cout << "Enter repository name: ";

cin >> repoName;

Repository\* repo = repositoryTree.searchRepository(repoName);

if (repo) {

cout << "Enter commit message: ";

cin.ignore();

getline(cin, commitMessage);

Commit\* newCommit = new Commit(commitMessage);

if (repo->addCommit(newCommit)) {

cout << "COMMIT ADDED!" << endl;

}

else {

cout << "FAILED TO ADD COMMIT! TRY AGAIN" << endl;

delete newCommit;

}

}

else {

cout << "REPOSITORY NOT FOUND!" << endl;

};

}

else if (choice == 8) {

string repoName, fileName;

cout << "Enter repository name: ";

cin >> repoName;

Repository\* repo = repositoryTree.searchRepository(repoName);

if (repo) {

cout << "Enter file name: ";

cin >> fileName;

File\* newFile = new File(fileName);

if (repo->addFile(newFile)) {

cout << "FILE ADDED!" << endl;

}

else {

cout << "FAILED TO ADD FILE!" << endl;

delete newFile;

}

}

else {

cout << "REPOSITORY NOT FOUND!" << endl;

}

}

else if (choice == 9) {

string repoName;

cout << "Enter repository name: ";

cin >> repoName;

Repository\* repo = repositoryTree.searchRepository(repoName);

if (repo) {

cout << "Repository Name: " << repo->getName() << endl;

cout << "Commits:" << endl;

const Commit\*\* commits = repo->getCommits();

for (int i = 0; i < repo->getCommitCount(); ++i) {

cout << "- " << commits[i]->getMessage() << endl;

}

cout << "Files:" << endl;

const File\*\* files = repo->getFiles();

for (int i = 0; i < repo->getFileCount(); ++i) {

cout << "- " << files[i]->getName() << endl;

}

}

else {

cout << "REPOSITORY NOT FOUND!" << endl;

}

}

else if (choice == 10) {

cout << "EXITING" << endl;

system("PAUSE");

return 0;

}

else {

cout << "INVALID OPTION! TRY AGAIN" << endl;

}

system("PAUSE");

}

}