**DATA STRUCTURE FINAL PROJECT**



**NAME:**SAQIB ALI

**ROLL NO:**22F-3116

**SECTION:**BAI-4A

**GITHUB REPOSITORY LINK:** https://github.com/Saqibali944/Github-Simulation

**HEADER FILE:**

#include<iostream> //libraries used

#include<fstream>

#include<string>

#include<sstream>

using namespace std;

struct filenode {

string filename;

filenode\* nextFile;

filenode(const string& filenm);

};

struct commitnode {

string commitmessage;

commitnode\* nextcommit;

filenode\* filehead;

commitnode(const string& commitmsg);

};

struct repo {

string name;

repo\* left;

repo\* right;

commitnode\* commithead;

repo(const string& nam);

};

struct User {

string username;

bool isLoggedIn;

int numFollowers;

repo\* repository; //added member to associate repository with user

User(const string& uname);

};

//define graph class for user relationships

class graph {

private:

static const int MAX\_USERS = 100;

string users[MAX\_USERS];

bool followMatrix[MAX\_USERS][MAX\_USERS]; // Adjacency matrix for user relationships

int numUsers;

public:

graph();

void addUser(const string& username);

void followUser(const string& follower, const string& followee);

void unfollowUser(const string& follower, const string& followee);

void printGraph();

private:

int findUserIndex(const string& username);

};

class hashtable {

private:

User\* table[100];

graph usergraph;

public:

hashtable();

int hash(const string& key);

int findUser(const string& username);

//adding a new user

void insertuser(const string& username);

//login a user

void loginuser(const string& username);

//logout a user

void logoutuser(string username);

//getting user by username

User\* getUser(string username);

//saving user data

void saveusers(string filename);

//delete repository for a user

void deleterepository(string username);

//view user profile

void viewusers(string username);

//adding new repository

void createrepository(string username, string reponame);

// Function to follow a user

void followuser(string follower, string followee);

// Function to unfollow a user

void unfollowuser(string follower, string followee);

};

**CPP FILE:**

#include"Header.h"

graph::graph() : numUsers(0) {

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

users[i] = "";

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

followMatrix[i][j] = false;

}

}

}

void graph::addUser(const string& username) {

if (numUsers < MAX\_USERS) {

users[numUsers++] = username;

}

}

void graph::followUser(const string& follower, const string& followee) {

int followerIndex = findUserIndex(follower);

int followeeIndex = findUserIndex(followee);

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

followMatrix[followerIndex][followeeIndex] = true;

}

}

void graph::unfollowUser(const string& follower, const string& followee) {

int followerIndex = findUserIndex(follower);

int followeeIndex = findUserIndex(followee);

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

followMatrix[followerIndex][followeeIndex] = false;

}

}

void graph::printGraph() {

cout << "User Graph:\n";

cout << " ";

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

cout << users[i] << " ";

}

cout << endl;

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

cout << users[i] << " ";

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

cout << (followMatrix[i][j] ? "1 " : "0 ");

}

cout << endl;

}

}

int graph::findUserIndex(const string& username) {

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

if (users[i] == username) {

return i;

}

}

return -1; // User not found

}

hashtable::hashtable() {

for (int i = 0; i < 100; ++i) {

table[i] = nullptr;

}

}

int hashtable::hash(const string& key) {

int sum = 0;

for (char c : key) {

sum += c;

}

return (sum % 100);

}

//adding a new user

int hashtable::findUser(const string& username) {

int index = hash(username);

int originalIndex = index;

while (table[index] != nullptr && table[index]->username != username) {

index = (index + 1) % 100;

if (index == originalIndex) {

return -1; // if user not found

}

}

return index;

}

void hashtable::insertuser(const string& username) {

int index = hash(username);

while (table[index] != nullptr) {

index = (index + 1) % 100; //linear probing

}

table[index] = new User(username);

}

//login a user

void hashtable::loginuser(const string& username) {

int index = findUser(username);

if (index != -1) {

table[index]->isLoggedIn = true; // Fixed typo

cout << "Login successful " << endl;

}

else {

cout << "Unable to login";

}

}

//logout a user

void hashtable::logoutuser(string username) {

int index = findUser(username);

if (index != -1) {

table[index]->isLoggedIn = false;

}

}

//getting user by username

User\* hashtable::getUser(string username) {

int index = findUser(username);

return (index != -1) ? table[index] : nullptr;

}

//saving user data

void hashtable::saveusers(string filename) {

ofstream file(filename);

if (!file.is\_open()) {

cerr << "Error opening file: " << filename << endl;

return;

}

for (int i = 0; i < 100; ++i) {

if (table[i] != nullptr) {

file << table[i]->username << "," << table[i]->isLoggedIn << "," << table[i]->numFollowers << endl;

}

}

file.close();

cout << "User data saved to " << filename << endl;

}

//delete repository for a user

void hashtable::deleterepository(string username) {

User\* user = getUser(username);

if (user != nullptr && user->repository != nullptr) {

delete user->repository;

user->repository = nullptr;

}

}

//view user profile

void hashtable::viewusers(string username) {

User\* user = getUser(username);

if (user != nullptr) {

cout << "Username: " << user->username << endl;

cout << "Is Logged In: ";

if (user->isLoggedIn) {

cout << "Yes" << endl;

}

else {

cout << "No" << endl;

}

cout << "Number of Followers: " << user->numFollowers << endl;

}

}

//adding new repository

void hashtable::createrepository(string username, string reponame) {

User\* user = getUser(username);

if (user != NULL && user->repository == NULL) {

user->repository = new repo(reponame);

}

}

// Function to follow a user

void hashtable::followuser(string follower, string followee) {

usergraph.followUser(follower, followee);

cout << "Followed user!";

}

// Function to unfollow a user

void hashtable::unfollowuser(string follower, string followee) {

usergraph.unfollowUser(follower, followee);

cout << "Unfollowed user!";

}

commitnode::commitnode(const string& commitmsg) {

commitmessage = commitmsg;

filehead = NULL;

nextcommit = NULL;

}

repo::repo(const string& nam) {

name = nam;

left = NULL;

right = NULL;

commithead = NULL;

}

filenode::filenode(const string& filenm) {

filename = filenm;

nextFile = NULL;

}

//added member to associate repository with user

User::User(const string& uname) : username(uname), isLoggedIn(false), numFollowers(0), repository(nullptr) {}

**MAIN FILE:**

#include"Header.h"

// Main part

int main() {

hashtable table;

int c;

do {

cout << "--- GITHUB SIMULATION ---" << endl;

cout << "Press 1 to add new user" << endl;

cout << "Press 2 to login" << endl;

cout << "Press 3 to logout" << endl;

cout << "Press 4 to view user profile" << endl;

cout << "Press 5 to save user data in file" << endl;

cout << "Press 6 to create repository" << endl;

cout << "Press 7 to delete repository" << endl;

cout << "Press 8 to follow user" << endl;

cout << "Press 9 to unfollow user" << endl;

cout << "Press 0 to close program" << endl;

cin >> c;

switch (c) {

case 1: {

string username;

cout << "Enter username" << endl;

cin >> username;

table.insertuser(username);

break;

}

case 2: {

string username;

cout << "Enter username" << endl;

cin >> username;

table.loginuser(username);

break;

}

case 3: {

string username;

cout << "Enter username: " << endl;

cin >> username;

table.logoutuser(username);

break;

}

case 4: {

string username;

cout << "Enter username" << endl;

cin >> username;

table.viewusers(username);

break;

}

case 5:

table.saveusers("users.csv");

break;

case 6: {

string username, repoName;

cout << "Enter username" << endl;

cin >> username;

cout << "Enter repository name" << endl;

cin >> repoName;

table.createrepository(username, repoName);

break;

}

case 7: {

string username;

cout << "Enter username" << endl;

cin >> username;

table.deleterepository(username);

break;

}

case 8: {

string follower, followee;

cout << "Enter follower's username" << endl;

cin >> follower;

cout << "Enter followee's username" << endl;

cin >> followee;

table.followuser(follower, followee);

break;

}

case 9: {

string f1, f2;

cout << "Enter follower's username" << endl;

cin >> f1;

cout << "Enter followee's username" << endl;

cin >> f2;

table.unfollowuser(f1, f2);

break;

}

case 0:

cout << "BYE!!!!!!" << endl;

break;

default:

cout << "Incorrect option entered" << endl;

}

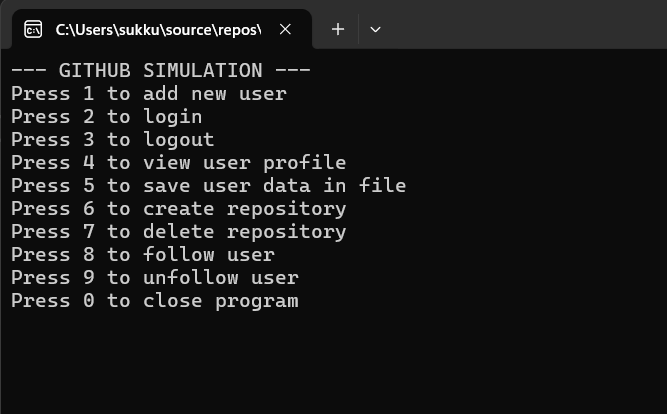
} while (c != 0);

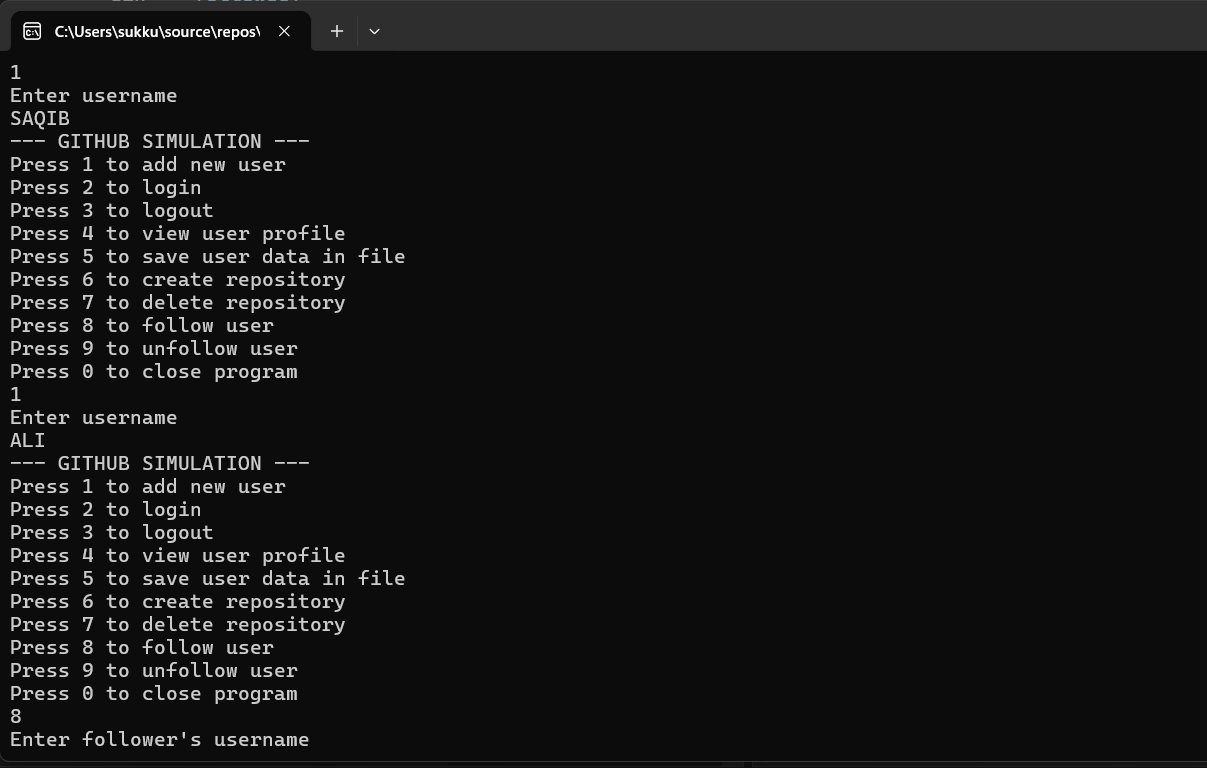
system("pause");

return 0;

}

**OUTPUT:**





A screenshot of a computer

Description automatically generated

