**GITHUB**

**JAWERIA REHMAN 22F-3352**

**AYESHA MUDASSAR 22F-8761**

**IFRAH RAUF 22F-3326**

**MAIN:**

#include"header.h"

#include<fstream>

#include<string>

#include<iostream>

using namespace std;

int main()

{

Account account;

string user, pass, repoName, fileName;

int val, val2;

Git git{};

Repos repos{};

SocialNetwork network;

bool log = false;

do

{

system("color 0B");

cout << "--------------------------------------\n";

cout << "\t<Git Simulator>\n";

cout << "--------------------------------------\n";

cout << "\n\n Please Select an Option:\n\n";

cout << "1) Register\n";

cout << "2) Login\n";

cout << "3) Exit\n";

cout << "--------------------------------------\n";

cout << "\nEnter Choice: ";

cin >> val;

system("cls");

if (val == 1)

{

cout << "--------------------------------------\n";

cout << "Registertion\n";

cout << "--------------------------------------\n";

cout << "\nEnter Username : ";

cin.ignore();

getline(cin,user);

cout << "\nEnter Password : ";

cin.ignore();

getline(cin,pass);

cout << endl;

if (account.regsiter(user, pass))

{

cout << "Registered Successfully!\n\n";

}

else

{

cout << "Invalid Username!!\n\n";

}

cout << "--------------------------------------\n";

}

else if (val == 2)

{

cout << "--------------------------------------\n";

cout << "Login\n";

cout << "--------------------------------------\n";

cout << endl << endl << "Enter Username : ";

cin.ignore();

getline(cin, user);

network.check = user;

cout << endl << endl << "Enter Password : ";

cin.ignore();

getline(cin, pass);

cout << endl;

log = git.chkUser(user, pass);

if (log)

{

cout << "Successfully Logged In!\n\n";

system("pause");

}

else

{

cout << "Invalid Username or Password!!\n\n";

system("pause");

}

cout << "--------------------------------------\n";

}

else if (val == 3)

{

cout << "Exiting\n";

break;

}

else

{

cout << "Invalid Choice\n";

}

if (log)

{

do

{

system("pause");

system("cls");

cout << "--------------------------------------\n";

cout << "\tGit Simulator\n";

cout << "--------------------------------------\n";

cout << "Please Select an Option:\n";

cout << "0) Social Features\n";

cout << "1) Create Repository\n";

cout << "2) Remove Repository\n";

cout << "3) Create File in Repository\n";

cout << "4) Remove File from Repository\n";

cout << "5) Search a Repository\n";

cout << "6) Search a File in Repository\n";

cout << "7) Commit in Repository\n";

cout << "8) Display All Repository\n";

cout << "9) Logout\n";

cout << "--------------------------------------\n";

cout << "\nEnter: ";

cin >> val2;

system("cls");

switch (val2)

{

case 0:

{

network.readUsersFromFile("Users.txt");

int action;

string username, target;

while (true)

{

system("pause");

system("cls");

cout << "--------------------------------------\n";

cout << " Social Features\n";

cout << "--------------------------------------\n";

cout << "Enter username : ";

cin.ignore();

getline(cin , username);

if (!network.userExists(username))

{

cout << "User " << username << " not found." << endl;

break;

// continue;

}

if (network.check == username)

{

cout << "Enter action:" << endl;

cout << "1) follow\n";

cout << "2) unfollow\n";

cout << "3) display\n";

cout << "4) exit\n";

}

else

{

cout << "Invalid username , Login account to use Social features.\n";

system("pause");

break;

}

cout << "\n";

cout << "action(should be an integer value):";

cin >> action;

if (action == 1 || action == 2)

{

cout << "Enter target username: ";

cin.ignore();

getline(cin , target);

if (!network.userExists(target))

{

cout << "User " << target << " not found." << endl;

}

if (action == 1)

{

network.follow(username, target);

}

else

{

network.unfollow(username, target);

}

}

else if (action == 3)

{

network.displayFollowers(username);

}

else if (action == 4)

{

cout << "Exiting social features..\n";

break; // Exit the social features submenu

}

else

{

cout << "Invalid action. Please try again." << endl;

system("pause");

break;

}

}

cout << "--------------------------------------\n";

break;

}

case 1:

cout << "--------------------------------------\n";

cout << "Respository Creation\n";

cout << "--------------------------------------\n";

cout << "Enter Repo Name:\n";

cin.ignore();

getline(cin,repoName);

repos.Repo(user, repoName, "add");

cout << "Repository Created!!!\n";

break;

case 2:

cout << "--------------------------------------\n";

cout << "Respository Deletion\n";

cout << "--------------------------------------\n";

if (repos.Repo(user, repoName, "delete"))

{

cout << "Repository Deleted\n";

}

else

{

cout << "Repository Not Deleted\n";

}

break;

case 3:

cout << "--------------------------------------\n";

cout << "File Creation in Respository\n";

cout << "--------------------------------------\n";

cout << "\n\nEnter File Name:\n";

cin.ignore();

getline(cin , fileName);

cout << "Enter Repository Name:\n";

cin.ignore();

getline(cin,repoName);

repos.File(user, repoName, fileName, "add");

cout << "File Created\n";

break;

case 4:

cout << "--------------------------------------\n";

cout << "File deletion from respository\n";

cout << "--------------------------------------\n";

cout << "\n\nEnter File Name:\n";

cin.ignore();

getline(cin ,fileName);

if (repos.File(user, repoName, fileName, "delete"))

{

cout << "File Deleted\n";

}

else

{

cout << "File Not Deleted\n";

}

break;

case 5:

cout << "--------------------------------------\n";

cout << "Searching Respository details\n";

cout << "--------------------------------------\n";

cout << "Enter Repository Name: ";

cin.ignore();

getline(cin , repoName);

if (!repos.Repo(" ", repoName, "searchRepoDetails"))

{

cout << "Repository Not Found\n";

}

else

{

cout << "Repository Found\n";

}

break;

case 6:

cout << "--------------------------------------\n";

cout << "Searching file\n";

cout << "--------------------------------------\n";

cout << "Enter File name: ";

cin.ignore();

getline( cin , fileName);

if (!repos.File(" ", repoName, fileName, "searchRepoDetails"))

{

cout << "File Not Found\n";

}

else

{

cout << "File found\n";

}

break;

case 7:

cout << "--------------------------------------\n";

cout << "Respository commits\n";

cout << "--------------------------------------\n";

repos.Repo(" ", repoName, "commit");

cout << "Committed\n";

break;

case 8:

cout << "--------------------------------------\n";

cout << "Displaying All Repositories\n";

cout << "--------------------------------------\n";

repos.displayRepo(user);

break;

case 9:

cout << "Logging Out\n";

system("pause");

system("cls");

log = false;

break;

default:

cout << "Invalid Choice\n";

break;

}

cout << endl;

} while (val2 != 9);

}

} while (val != 3);

return 0;

}

**HEADER:**

#pragma once

#ifndef GIT\_H

#define GIT\_H

#include<iostream>

#include<string>

#include <unordered\_map>

#include <vector>

using namespace std;

class Account

{

private:

string username, password;

static int totalAcc;

friend class Git;

public:

bool regsiter(string username, string password);

};

class Git

{

private:

Account\* accounts;

int hash(string username, int size) const;

public:

bool chkUser(string user, string pass);

};

class Node

{

public:

string data;

int ch;

Node\* childs;

int comits;

Node() : data(""), ch(0), childs(NULL), comits(0) {}

};

class Repos

{

private:

Node\* node;

void Tree(string x);

public:

void addToRepo(string user);

bool Repo(string user, string repoName, string x);

bool File(string user, string repoName, string fileName, string x);

void displayRepo(string user);

};

//Socials

class SocialNetwork

{

private:

unordered\_map<string, vector<string>> graph; // Key: user, Value: followers

public:

string check;

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

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

void readUsersFromFile(const string& filename);

bool userExists(const string& username) const;

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

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

bool isFollowing(const string& follower, const string& followee) const;

void displayFollowers(const string& username) const;

};

#endif

**CPP:**

#include"header.h"

#include<fstream>

//#include<vector>

#include<string>

#include <unordered\_map>

#include <vector>

#include <sstream>

#include<iostream>

using namespace std;

bool checkStringSimilarity(string x, string y);

bool Git::chkUser(string user, string pass) //User Login

{

int count = 0;

ifstream in;

in.open("Count.txt");

in >> count;

in.close();

accounts->totalAcc = count \* 2;

this->accounts = new Account[count \* 2];//dynamic memory allocation

for (int i = 0; i < count \* 2; i++)

{

accounts[i].username = " ";

}

in.open("Users.txt");

int index;

string userr, passs;

for (int i = 0; i < count; i++)

{

in >> userr >> passs;

index = hash(user, count \* 2);//to calculate an index,

accounts[index].username = userr;

accounts[index].password = passs;

}

int total = 0;

for (int i = 0; i < user.length(); i++)

{

total += int(user[i]);

}

int value = total % (count \* 2);

while (accounts[value].username != " " &&accounts[value].username != user)

{

if (value == accounts->totalAcc)

{

value = 0;

}

value++;

}

if (accounts[value].username == user && accounts[value].password == pass)

{

return true;

}

return false;

}

int Git::hash(string username, int size) const //constant hash funstion

{

int total = 0;

for (int i = 0; i < username.length(); i++)

{

total += int(username[i]);//Adds the ASCII value of the characters of the username

}

int value = total % size;

while (accounts[value].username != " ")

{

value++;

if (value == size)

{

value = 0;

}

}

return value;

}

int Account::totalAcc = 0;

bool Account::regsiter(string username, string password) //User Registeration

{

Repos rp;

ifstream in;

bool valid = true;

in.open("Users.txt");

string usrr, pss;

while (!in.eof()) //ye loop user ko file se read krne k liye hai

{

in >> usrr >> pss;

if (checkStringSimilarity(username, usrr)) //ye condition check kr rhi hai k user already exist hai ya nahi

{

valid = false;

}

}

in.close();

if (valid)

{

ofstream out;

out.open("Users.txt", ios::app);

out << username << endl << password << endl;

out.close();

int count;

ifstream in;

in.open("Count.txt");

in >> count;

in.close();

out.open("Count.txt");

count++;

out << count;

out.close();

rp.addToRepo(username);

return true;

}

return false;

}

bool checkStringSimilarity(string x, string y)

{

if (x.length() != y.length())

return false;

else

{

for (int i = 0; i < x.length(); i++)

{

if (x[i] != y[i])

{

return false;

}

}

return true;

}

}

void Repos::Tree(string x) //Tree Structure

{

if (x == "load")

{

ifstream in;

in.open("Repositories.txt");

int users, repos, files;

string str;

in >> users;

node = new Node;

node->childs = new Node[users];

node->ch = users;

for (int i = 0; i < users; i++) //Es loop mn user ko file se read kr k tree mn store kr rha hai

{

in >> str;

node->childs[i].data = str;

in >> repos;

node->childs[i].childs = new Node[repos];

node->childs[i].ch = repos;

for (int j = 0; j < repos; j++)

{

in >> str;

node->childs[i].childs[j].data = str;

in >> files;

node->childs[i].childs[j].childs = new Node[files];

node->childs[i].childs[j].ch = files;

for (int k = 0; k < files; k++)

{

in >> str;

node->childs[i].childs[j].childs[k].data = str;

}

}

}

in.close();

}

else if (x == "unload")

{

ofstream out;

out.open("Repositories.txt");

out << node->ch << endl;

for (int i = 0; i < node->ch; i++)

{

out << node->childs[i].data << endl;

out << node->childs[i].ch << endl;

for (int j = 0; j < node->childs[i].ch; j++)

{

out << node->childs[i].childs[j].data << endl;

out << node->childs[i].childs[j].ch << endl;

for (int k = 0; k < node->childs[i].childs[j].ch; k++)

{

out << node->childs[i].childs[j].childs[k].data << endl;

}

}

}

}

}

void Repos::addToRepo(string user) //Add User to Repository

{

ifstream in;

in.open("Repositories.txt");

int users = 0, repos = 0, files;

string str;

in >> users;

node = new Node();

node->childs = new Node[users];

node->ch = users;

for (int i = 0; i < users; i++)

{

in >> str;

node->childs[i].data = str;

in >> repos;

node->childs[i].childs = new Node[repos];

node->childs[i].ch = repos;

for (int j = 0; j < repos; j++)

{

in >> str;

node->childs[i].childs[j].data = str;

in >> files;

node->childs[i].childs[j].childs = new Node[files];

node->childs[i].childs[j].ch = files;

for (int k = 0; k < files; k++)

{

in >> str;

node->childs[i].childs[j].childs[k].data = str;

}

}

}

in.close();

ofstream out;

out.open("Repositories.txt");

out << node->ch + 1 << endl;

for (int i = 0; i < node->ch; i++)

{

out << node->childs[i].data << endl;

out << node->childs[i].ch << endl;

for (int j = 0; j < node->childs[i].ch; j++)

{

out << node->childs[i].childs[j].data << endl;

out << node->childs[i].childs[j].ch << endl;

for (int k = 0; k < node->childs[i].childs[j].ch; k++)

{

out << node->childs[i].childs[j].childs[k].data << endl;

}

}

}

out << user << endl << 0 << endl;

}

bool Repos::File(string user, string repoName, string fileName, string x) //File Functions

{

if (x == "add")

{

ifstream in;

in.open("Repositories.txt");

int users, repos, files;

string str;

in >> users;

node = new Node;

node->childs = new Node[users];

node->ch = users;

for (int i = 0; i < users; i++)

{

in >> str;

node->childs[i].data = str;

in >> repos;

node->childs[i].childs = new Node[repos];

node->childs[i].ch = repos;

for (int j = 0; j < repos; j++) //Es loop mn file ko add krne k liye hai

{

in >> str;

node->childs[i].childs[j].data = str;

in >> files;

if (node->childs[i].data == user && node->childs[i].childs[j].data == repoName)

{

files++;

}

node->childs[i].childs[j].childs = new Node[files];

node->childs[i].childs[j].ch = files;

for (int k = 0; k < files; k++)

{

if (node->childs[i].data == user && node->childs[i].childs[j].data == repoName && k == files - 1)

{

node->childs[i].childs[j].childs[k].data = fileName;

}

else

{

in >> str;

node->childs[i].childs[j].childs[k].data = str;

}

}

}

}

in.close();

Tree("unload");

}

else if (x == "delete")

{

Tree("load");

if (!File(user, repoName, fileName, "search"))

{

return false;

}

ofstream out;

out.open("Repositories.txt");

out << node->ch << endl;

for (int i = 0; i < node->ch; i++)

{

out << node->childs[i].data << endl;

out << node->childs[i].ch << endl;

for (int j = 0; j < node->childs[i].ch; j++)

{

out << node->childs[i].childs[j].data << endl;

if (node->childs[i].data == user && node->childs[i].childs[j].data == repoName)

{

out << node->childs[i].childs[j].ch - 1 << endl;

}

else

{

out << node->childs[i].childs[j].ch << endl;

}

for (int k = 0; k < node->childs[i].childs[j].ch; k++)

{

if (node->childs[i].data == user && node->childs[i].childs[j].data == repoName && node->childs[i].childs[j].childs[k].data == fileName)

{

}

else

{

out << node->childs[i].childs[j].childs[k].data << endl;

}

}

}

}

return true;

}

else if (x == "search")

{

for (int i = 0; i < node->ch; i++)

{

for (int j = 0; j < node->childs[i].ch; j++)

{

for (int k = 0; k < node->childs[i].childs[j].ch; k++)

{

if (node->childs[i].data != user || node->childs[i].childs[j].data != repoName)

{

break;

}

else if (node->childs[i].data == user && node->childs[i].childs[j].data == repoName && node->childs[i].childs[j].childs[k].data == fileName)

{

return true;

}

}

}

}

return false;

}

}

bool Repos::Repo(string user, string repoName, string x) //Repository Functions

{

Tree("load");

if (x == "add")

{

ifstream in;

in.open("Repositories.txt");

int users, repos, files;

string str;

in >> users;

node = new Node;

node->childs = new Node[users];

node->ch = users;

for (int i = 0; i < users; i++)

{

in >> str;

node->childs[i].data = str;

in >> repos;

if (str == user)

{

repos++;

}

node->childs[i].childs = new Node[repos];

node->childs[i].ch = repos;

for (int j = 0; j < repos; j++)

{

if (node->childs[i].data == user && j == repos - 1)

{

node->childs[i].childs[j].data = repoName;

node->childs[i].childs[j].ch = 0;

}

else

{

in >> str;

node->childs[i].childs[j].data = str;

in >> files;

node->childs[i].childs[j].childs = new Node[files];

node->childs[i].childs[j].ch = files;

for (int k = 0; k < files; k++)

{

in >> str;

node->childs[i].childs[j].childs[k].data = str;

}

}

}

}

in.close();

Tree("unload");

return true;

}

else if (x == "delete")

{

Tree("load");

if (!Repo(user, repoName, "search"))

{

return false;

}

ofstream out;

out.open("Repositories.txt");

out << node->ch << endl;

for (int i = 0; i < node->ch; i++)

{

out << node->childs[i].data << endl;

if (node->childs[i].data == user)

{

out << node->childs[i].ch - 1 << endl;

}

else

{

out << node->childs[i].ch << endl;

}

for (int j = 0; j < node->childs[i].ch; j++)

{

if (node->childs[i].data != user && node->childs[i].childs[j].data != repoName)

{

out << node->childs[i].childs[j].data << endl;

out << node->childs[i].childs[j].ch << endl;

for (int k = 0; k < node->childs[i].childs[j].ch; k++)

{

out << node->childs[i].childs[j].childs[k].data << endl;

}

}

}

}

return true;

}

else if (x == "search")//Search Repository

{

bool flag = false;

for (int i = 0; i < node->ch; i++)

{

for (int j = 0; j < node->childs[i].ch; j++)

{

if (node->childs[i].data != user)

{

break;

}

else if (node->childs[i].data == user && node->childs[i].childs[j].data == repoName)

{

return true;

}

}

}

return false;

}

else if (x == "commit") //Commit Repository

{

Tree("load");

for (int i = 0; i < node->ch; i++)

{

for (int j = 0; j < node->childs[i].ch; j++)

{

if (node->childs[i].childs[j].data == repoName)

{

node->childs[i].childs[j].comits++;

}

}

}

}

else if (x == "searchRepoDetails") //Search Repository Details

{

Tree("load");

bool flag = false;

for (int i = 0; i < node->ch; i++)

{

for (int j = 0; j < node->childs[i].ch; j++)

{

if (node->childs[i].childs[j].data == repoName)

{

cout << "\n";

cout << "\n\t-> User : " << node->childs[i].data;

cout << "\n\t-> Repo Name : " << node->childs[i].childs[j].data;

cout << "\n\t-> Total Files : " << node->childs[i].childs[j].ch;

cout << "\n\n";

return true;

}

}

}

return false;

}

}

void Repos::displayRepo(string user)

{

Tree("load");

for (int i = 0; i < node->ch; i++)

{

if (node->childs[i].data == user)

{

for (int j = 0; j < node->childs[i].ch; j++)

{

cout << "\n\t-> Repository Name: " << node->childs[i].childs[j].data;

cout << "\n\t-> Files: ";

for (int k = 0; k < node->childs[i].childs[j].ch; k++)

{

cout << node->childs[i].childs[j].childs[k].data << " ";

}

cout << "\n\n";

}

}

}

}

//Socials

void SocialNetwork::readUsersFromFile(const string& filename)

{

ifstream file(filename);

if (!file.is\_open())

{

cerr << "Error: Unable to open file " << filename << endl;

return;

}

string line;

while (getline(file, line))

{

stringstream ss(line);

string username;

ss >> username;

string follower;

vector<string> followers;

while (ss >> follower)

{

followers.push\_back(follower);

updateFollowersFile(username, follower); // Update followers file

}

graph[username] = followers;

}

file.close();

}

// Function to update followers file

void SocialNetwork::updateFollowersFile(const string& followee, const string& follower)

{

ofstream file(followee + "\_followers.txt", ios::app);

if (file.is\_open())

{

file << follower << endl;

file.close();

}

else

{

cout << "Error: Unable to update followers file for user " << followee << endl;

}

}

// Function to remove follower from followers file

void SocialNetwork::removeFollowerFromFile(const string& followee, const string& follower) {

ifstream inFile(followee + "\_followers.txt");

ofstream tempFile(followee + "\_temp.txt");

string line;

bool followerFound = false; // To check if the follower is found in the file

while (getline(inFile, line)) {

if (line != follower) {

tempFile << line << endl; // Write to the temp file if it's not the follower

}

else {

followerFound = true; // Set followerFound to true if follower is found

}

}

inFile.close();

tempFile.close();

if (followerFound) {

// Remove the original followers file

if (remove((followee + "\_followers.txt").c\_str()) == 0) { // Check if removal is successful

// Rename temp file to original followers file

if (rename((followee + "\_temp.txt").c\_str(), (followee + "\_followers.txt").c\_str()) == 0) { // Check if renaming is successful

cout << "Follower removed successfully." << endl;

return; // exit the function

}

else {

cerr << "Error: Unable to rename temp file." << endl;

}

}

else {

cerr << "Error: Unable to remove followers file." << endl;

}

}

else {

cout << "Follower not found in the file." << endl;

}

// Remove the temporary file if not successful

remove((followee + "\_temp.txt").c\_str());

}

// Function to check if user exists

bool SocialNetwork::userExists(const string& username) const

{

return graph.find(username) != graph.end();

}

// Function to follow another user

void SocialNetwork::follow(const string& follower, const string& followee)

{

if (!userExists(followee))

{

cout << "User " << followee << " not found." << endl;

return;

}

if (follower == followee)

{

cout << "You cannot follow yourself." << endl;

return;

}

if (!isFollowing(follower, followee))

{

graph[follower].push\_back(followee);

updateFollowersFile(followee, follower); // Update followers file

cout << follower << " is now following " << followee << endl;

}

else {

cout << follower << " is already following " << followee << endl;

}

}

void SocialNetwork::unfollow(const string& follower, const string& followee)

{

if (!userExists(followee))

{

cout << "User " << followee << " not found." << endl;

return;

}

if (isFollowing(follower, followee))

{

removeFollowerFromFile(followee, follower);

}

else

{

cout << follower << " is not following " << followee << endl;

}

}

void SocialNetwork::displayFollowers(const string& username) const

{

ifstream file(username + "\_followers.txt");

if (!file.is\_open())

{

cout << "No followers found for user " << username << endl;

return;

}

string follower;

cout << "Followers of " << username << ":" << endl;

while (getline(file, follower))

{

cout << follower << endl;

}

file.close();

}

// Function to check if a user is already following another user

bool SocialNetwork::isFollowing(const string& follower, const string& followee) const

{

ifstream file(followee + "\_followers.txt");

if (!file.is\_open())

{

return false;

}

string line;

while (getline(file, line))

{

if (line == follower)

{

return true;

}

}

return false;

}

**OUTPUT:**  
























