Name: Ahmad & Um Habiba

Class: 3D

Project:

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

using namespace std;

class User {

private:

int id;

string name;

int age;

double pct;

bool isPub;

string region;

int lastLogin;

public:

User()

: id(0), name(" "), age(0), pct(0.0), isPub(false), region(" "), lastLogin(0) {

}

User(int id, string name, int age, double pct, bool isPub, string region, int lastLogin)

: id(id), name(name), age(age), pct(pct), isPub(isPub), region(region), lastLogin(lastLogin) {

}

void setPct(double p) {

pct = p;

}

int getId() const { return id; }

string getName() const { return name; }

int getAge() const { return age; }

double getPct() const { return pct; }

bool isPublic() const { return isPub; }

string getRegion() const { return region; }

int getLastLogin() const { return lastLogin; }

// ========================= Preposition ============================

bool checkProp1() {

if (age > 30 && pct > 50.0) {

return true;

}

return false;

}

bool checkProp2(int ly, int py) const {

if (lastLogin<ly && lastLogin >py) {

return true;

}

else {

return false;

}

}

bool is\_pro() const {

return pct > 80.0;

}

};

class Pokec {

private:

User\*\* users;

bool\*\* friendship;

int size;

public:

Pokec(int numUsers) {

size = numUsers;

users = new User \* [size];

friendship = new bool\* [size];

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

users[i] = nullptr;

friendship[i] = new bool[size];

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

friendship[i][j] = false;

}

}

}

void setFriendship(int user1, int user2, bool isFriend) {

if (user1 < size && user2 < size) {

friendship[user1][user2] = isFriend;

friendship[user2][user1] = isFriend; // Friendship is mutual

}

}

////=================== Read User File For Input ===================

void readUsersFromFile(const string& filename) {

ifstream fin(filename);

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

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

return;

}

string line;

int idx = 0;

while (getline(fin, line) && idx < size) {

stringstream ss(line);

int id, age, lastLogin;

double pct;

bool isPub;

string name, region;

getline(ss, line, ','); id = stoi(line);

getline(ss, name, ',');

getline(ss, line, ','); age = stoi(line);

getline(ss, line, ','); pct = stod(line);

getline(ss, line, ','); isPub = (line == "1");

getline(ss, region, ',');

getline(ss, line, ','); lastLogin = stoi(line);

users[idx++] = new User(id, name, age, pct, isPub, region, lastLogin);

}

fin.close();

}

////==================== Write User & Friend File ==================

void write\_user(const string& file\_name, int id, string n, string r, int age, int last\_l, double pct, bool ispub) {

ofstream fout(file\_name);

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

cout << "Error opening file: " << file\_name << endl;

return;

}

else {

fout << id << "," << n << "," << age << "," << pct << "," << ispub << "," << r << "," << last\_l << endl;

}

cout << "SuccessFuly Wrote in User.txt ! \n";

fout.close();

}

void write\_friend(const string& file\_name, int id1, int id2, bool is\_friend) {

ofstream fout(file\_name);

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

cout << "Error opening file: " << file\_name << endl;

return;

}

else {

fout << id1 << ", " << id2 << ", " << is\_friend << endl;

}

cout << "SuccessFuly Wrote in Friend.txt ! \n";

fout.close();

}

////==================== Read Friendship File ==================

void readFriendFile(const string& filename) {

ifstream fin(filename);

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

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

return;

}

string line;

int idx = 0;

while (getline(fin, line) && idx < size) {

stringstream ss(line);

int u1, u2;

bool is\_friend;

getline(ss, line, ','); u1 = stoi(line);

getline(ss, line, ','); u2 = stoi(line);

getline(ss, line, ','); is\_friend = (line == "1");

this->setFriendship(u1, u2, is\_friend);

}

fin.close();

}

void displayUsers() {

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

if (users[i] != nullptr) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName()

<< ", Age: " << users[i]->getAge() << ", Pct: " << users[i]->getPct()

<< ", isPublic: " << users[i]->isPublic() << ", Region: " << users[i]->getRegion()

<< ", Last Login: " << users[i]->getLastLogin() << endl;

}

}

}

void countProfile() {

int count = 0;

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

if (users[i] != nullptr && users[i]->is\_pro()) {

count++;

}

}

cout << "\nTotal number of users with profile completion above 80%: " << count << endl;

}

User\* getUser(int idx) {

if (idx < size) {

return users[idx]; // Return user at the given index

}

return nullptr; // Return nullptr if index is out of bounds

}

void addUser(int idx, User\* user) {

if (idx < size) {

users[idx] = user;

}

}

// ========================= Quantifiers ============================

void findUsers1(int minAge, double minPct, string region) {

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

if (users[i] != nullptr && users[i]->getAge() > minAge && users[i]->getPct() > minPct && users[i]->getRegion() == region) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << endl;

}

}

}

void findUsers2(int minAge, double minPct) {

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

if (users[i] != nullptr && users[i]->getAge() > minAge && users[i]->getPct() > minPct) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << endl;

}

}

}

// ========================= Set Operations ============================

void setOps(string c) {

cout << "Union (A ∪ B):" << endl;

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

// Union: Users who have a public profile or belong to "Bratislava"

if (users[i] != nullptr && (users[i]->isPublic() || users[i]->getRegion() == c)) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << endl;

}

}

// cout << "\nIntersection (A ∩ B):" << endl;

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

// Intersection: Users who are both public and from "Bratislava"

if (users[i] != nullptr && users[i]->isPublic() && users[i]->getRegion() == c) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << endl;

}

}

cout << "\nComplement of A (Non-public profiles):" << endl;

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

// Complement: Users who have a non-public profile

if (users[i] != nullptr && !users[i]->isPublic()) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << endl;

}

}

}

//// ========================= Vens Diagram ===========================

void show\_vens(int a, string city) {

cout << "People from " << city << " at " << a << " age are : \n";

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

if (users[i]->isPublic() && users[i]->getAge() > a && users[i]->getRegion() == city) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << endl;

}

else {

cout << "No , Body of These Conditions ! " << endl;

}

}

}

//// ========================= Injective and Surjective ============================

void checkFuncProps() {

bool injective = true;

bool surjective = false;

double pct[] = { 0.0, 59.0, 77.0, 98.0 }; // Possible completion percentages

bool used[4] = { false }; // Array to track which percentages have been used

// Checking for injectivity: No two users should have the same percentage

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

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

if (users[i] != nullptr && users[j] != nullptr && users[i]->getPct() == users[j]->getPct()) {

injective = false; // Found two users with the same percentage

}

}

}

// Checking for surjectivity: Every possible percentage should be used by some user

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

if (users[i] != nullptr) {

for (int j = 0; j < 4; j++) {

if (users[i]->getPct() == pct[j]) {

used[j] = true; // Mark the percentage as used

}

}

}

}

// If any percentage was not used, it's not surjective

for (int i = 0; i < 4; i++) {

if (!used[i]) {

surjective = false;

break;

}

}

cout << "Injective: " << (injective ? "Yes" : "No") << endl;

cout << "Surjective: " << (surjective ? "Yes" : "No") << endl;

}

//// ========================== Check Mutual Friend ================================

// 1. Check if the friendship relation is symmetric

bool isSymmetric() {

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

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

if (friendship[i][j] != friendship[j][i]) {

return false; // If there is any asymmetric relation

}

}

}

return true; // All relations are symmetric

}

// 2. Check if the friendship relation is transitive

void isTransitive() {

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

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

if (friendship[i][j]) { // If A is friends with B

for (int k = 0; k < size; k++) {

if (friendship[j][k] && !friendship[i][k]) {

// If B is friends with C but A is not friends with C

cout << "Suggestion: A(" << users[i]->getName() << ") -> C(" << users[k]->getName() << ")" << endl;

}

}

}

}

}

}

// 3. Check if the friendship relation is reflexive (each user is friends with themselves)

bool isReflexive() {

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

if (!friendship[i][i]) {

return false; // If any user is not friends with themselves

}

}

return true; // All users are friends with themselves

}

// ========================= Permutations and Combinations ============================

// 1. Count the number of ways to select a subgroup of users with specific criteria

void count\_sgp(int minAge, string region) {

int count = 0;

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

if (users[i] != nullptr && users[i]->getAge() > minAge && users[i]->getRegion() == region) {

count++;

}

}

cout << "Number of users over age " << minAge << " from " << region << ": " << count << endl;

}

// 2. Calculate permutations of users based on a given property (e.g., completion percentage)

void cal\_Per() {

// Sorting users manually by completion percentage (pct) in descending order

for (int i = 0; i < size - 1; i++) {

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

if (users[i] != nullptr && users[j] != nullptr && users[i]->getPct() < users[j]->getPct()) {

// Swap the users

User\* temp = users[i];

users[i] = users[j];

users[j] = temp;

}

}

}

// Display the sorted users (permutations based on pct)

cout << "Users sorted by completion percentage (in descending order):" << endl;

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

if (users[i] != nullptr) {

cout << "ID: " << users[i]->getId() << ", Name: " << users[i]->getName() << ", Pct: " << users[i]->getPct() << "%" << endl;

}

}

}

// ========================= Counting Functions ============================

// 1. Count total number of users with profile completion above 80%

void count\_Profile(double pct) {

int count = 0;

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

if (users[i] != nullptr) {

if (users[i]->getAge() > pct) {

count++;

}

}

}

cout << "Total number of users with profile completion above " << pct << " % : " << count << endl;

}

// 2. Count the number of users in each region

void count\_Reg(string city) {

int c1 = 0, c2 = 0, c3 = 0, c4 = 0, c = 0;

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

if (users[i] != nullptr) {

if (users[i]->getRegion() == city) {

c1++;

}

}

}

cout << "Total People From " << city << " : " << c1 << endl;

}

// 3. Generate a report on the count of public profiles

void gen\_Rep() {

int countPublic = 0;

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

if (users[i] != nullptr && users[i]->isPublic()) {

countPublic++;

}

}

cout << "Report on public profiles:" << endl;

cout << "Total public profiles: " << countPublic << endl;

}

// ========================= Induction Functions ============================

void inductiveProof(int X) {

int count = 0;

// Base Case: For the first user

if (users[0] != nullptr && users[0]->is\_pro()) {

count++;

}

// Inductive Hypothesis: Assume the property holds for first 'k' users

// Inductive Step: Now prove for 'k+1'th user

for (int i = 1; i < size; i++) {

if (users[i] != nullptr && users[i]->is\_pro()) {

count++;

}

}

// Final check if count > X (Inductive Proof Conclusion)

if (count > X) {

cout << "Inductive proof holds: The number of users with profile completion above "

<< 80 << " is greater than " << X << "." << endl;

}

else {

cout << "Inductive proof fails: The number of users with profile completion above "

<< 80 << " is not greater than " << X << "." << endl;

}

}

~Pokec() {

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

delete users[i];

delete[] friendship[i];

}

delete[] users;

delete[] friendship;

}

};

////================================= Trees =============================

struct tree {

string name;

tree\* left; // Left child (friend)

tree\* right; // Right child (friend)

tree(string name) {

this->name = name;

left = right = nullptr;

}

};

// Preorder Traversal (Root -> Left -> Right)

void preorder(tree\* root) {

if (root == nullptr) return;

cout << root->name << " ";

preorder(root->left);

preorder(root->right);

}

// Postorder Traversal (Left -> Right -> Root)

void postorder(tree\* root) {

if (root == nullptr) return;

postorder(root->left);

postorder(root->right);

cout << root->name << " ";

}

// BFS Traversal (breadth-first search).

void bfsTraversal(tree\* root) {

if (root == nullptr) return;

// Level 1: Print root

cout << root->name << " ";

// Level 2: Print left and right children

if (root->left) {

cout << root->left->name << " ";

}

if (root->right) {

cout << root->right->name << " ";

}

}

int main() {

Pokec pokec(501); // Initialize the Pokec system with 100 users

pokec.readUsersFromFile("users.txt");

pokec.readFriendFile("friend.txt");

int choice;

do {

cout << "\n========== POKEC SYSTEM MENU ==========\n";

cout << "1. Display All Users\n";

cout << "2. Find Users Based on Criteria (Quantifiers)\n";

cout << "3. Count Profile Completion\n";

cout << "4. Add Profile & Set Friendship Relations\n";

cout << "5. Preposition\n";

cout << "6. Check Friendship Properties (Symmetric, Transitive, Reflexive)\n";

cout << "7. Perform Set Operations (Union, Intersection, Complement)\n";

cout << "8. Check Function Properties (Injective, Surjective)\n";

cout << "9. Check Permutations and Groups(Permutation & Combination)\n";

cout << "10. Generate Reports(Counting)\n";

cout << "11. Inductive Proof\n";

cout << "12. Details in Venn Diagram\n";

cout << "13. Tree \n";

cout << "13. Exit \n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "\nAll Users:\n";

pokec.displayUsers();

break;

case 2: {

int age;

double completion;

string city;

int quantifier;

cout << "Enter Quantifier Condition (1 or 2): ";

cin >> quantifier;

if (quantifier == 1) {

cout << "Enter age(25): ";

cin >> age;

cout << "Enter profile completion percentage(>70): ";

cin >> completion;

cout << "Enter city (Zilinsky): ";

//getline(cin, city);

cin >> city;

pokec.findUsers1(age, completion, city);

}

else if (quantifier == 2) {

cout << "Enter age: ";

cin >> age;

cout << "Enter profile completion percentage: ";

cin >> completion;

pokec.findUsers2(age, completion);

}

else {

cout << "Invalid Quantifier. Try again.\n";

}

break;

}

case 3:

cout << "\nCounting Profile Completion...\n";

pokec.countProfile();

break;

case 4: {

int user1, user2;

bool isFriend;

int id = 0, age = 0, lastLogin = 0;

double pct = 0.0;

bool isPub = false;

string name = " ", region = " ";

int c = 0;

cout << "Enter Choice (1 or 2) : ";

cin >> c;

if (c == 2) {

cout << "Enter User 1 ID: ";

cin >> user1;

cout << "Enter User 2 ID: ";

cin >> user2;

cout << "Are they friends (1 for True, 0 for False): ";

cin >> isFriend;

// pokec.setFriendship(user1, user2, isFriend);

pokec.write\_friend("friend.txt", user1, user2, isFriend);

}

else if (c == 2) {

cout << "Enter USers Name : ";

cin >> name;

cout << "Enter Id : ";

cin >> id;

cout << "Enter Age : ";

cin >> age;

cout << "Enter PCT : ";

cin >> pct;

cout << "Enter Region : ";

cin >> region;

cout << "Enter Last Login Year : ";

cin >> lastLogin;

if (lastLogin >= lastLogin - 1) {

isPub = true;

}

else {

isPub = false;

}

pokec.write\_user("users.txt", id, name, region, age, lastLogin, pct, isPub);

}

else {

cout << "Invalid Input ! ";

}

break;

}

case 5: {

int condition;

cout << "Enter Preposition Condition (1 or 2): ";

cin >> condition;

if (condition == 1) {

cout << "Preposition Condition 1: "

<< "True/False Placeholder\n";

}

else if (condition == 2) {

int present, last;

cout << "Enter Present & Last Year: ";

cin >> present >> last;

cout << "Preposition Condition 2: True/False Placeholder\n";

}

else {

cout << "Invalid Condition!\n";

}

break;

}

case 6:

cout << "\nChecking Friendship Properties...\n";

cout << "Is Symmetric? " << (pokec.isSymmetric() ? "Yes" : "No") << endl;

pokec.isTransitive();

cout << "Is Reflexive? " << (pokec.isReflexive() ? "Yes" : "No") << endl;

break;

case 7: {

string city;

cout << "\nEnter City Name (e.g., Bratislava): ";

cin >> city;

pokec.setOps(city);

break;

}

case 8:

cout << "\nChecking Function Properties...\n";

pokec.checkFuncProps();

break;

case 9: {

string c;

int a;

cout << "Enter City Name (Bratislava): ";

cin >> c;

cout << "Enter Age: ";

cin >> a;

cout << "\nChecking Permutations and Groups...\n";

pokec.cal\_Per();

pokec.count\_sgp(a, c);

break;

}

case 10: {

string c;

double p;

cout << "Enter City Name (Bratislava): ";

cin >> c;

cout << "Enter PCT : ";

cin >> p;

cout << "\nGenerating Reports...\n";

pokec.count\_Profile(p);

pokec.count\_Reg(c);

pokec.gen\_Rep();

break;

}

case 11: {

int X;

cout << "Enter the minimum number of users with > 80% completion to prove: ";

cin >> X;

pokec.inductiveProof(X);

break;

}

case 12: {

string c;

int a;

cout << "Enter City Name (Bratislava): ";

cin >> c;

cout << "Enter Age: ";

cin >> a;

pokec.show\_vens(a, c);

break;

}

case 13: {

string main, l, r;

cout << "Tree Deatails ! \n";

cout << "Enter Root Name : ";

cin >> main;

cout << "Enter Left Child Name : ";

cin >> l;

cout << "Enter Right Child Name : ";

cin >> r;

tree\* root = new tree(main);

root->left = new tree(l);

root->right = new tree(r);

cout << "SuccessFuly Set ! \n";

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

cout << "Enter Left Friend Child Name : ";

cin >> l;

root->left->left = new tree(l);

cout << "Enter Left Friend Child Name : ";

cin >> r;

root->left->right = new tree(r);

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

cout << "Enter Left Friend Child Name : ";

cin >> l;

root->right->left = new tree(l);

cout << "Enter Left Friend Child Name : ";

cin >> r;

root->right->right = new tree(r);

// Perform Postorder Traversal (Root -> Left -> Right)

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

cout << "Preorder Traversal: ";

preorder(root);

cout << endl;

// Perform Postorder Traversal (Left -> Right -> Root)

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

cout << "Postorder Traversal: ";

postorder(root);

cout << endl;

// Perform BFS Traversal

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

cout << "BFS Traversal: ";

bfsTraversal(root);

cout << endl;

break;

}

case 14:

cout << "Exiting program. Goodbye!\n";

break;

default:

cout << "Invalid choice. Please try again.\n";

break;

}

} while (choice != 14);

return 0;

}