#include <bits/stdc++.h>

using *namespace* std;

*class* Voter {

*public*:

*int* voterID;

string name;

*int* age;

string address;

string password;

*bool* hasVoted;

Voter(*int* id, string name, *int* age, string address, string password)

: voterID(id), name(name), age(age), address(address), password(password), hasVoted(false) {}

};

*class* Candidate {

*public*:

*int* candidateID;

string name;

string party;

*int* votes;

Candidate(*int* id, string name, string party)

: candidateID(id), name(name), party(party), votes(0) {}

};

*class* Election {

*private*:

vector<Voter> voters;

vector<Candidate> candidates;

*int* totalVotes;

*bool* verifyVoter(*int* voterID, string password) {

for (*auto* &voter : voters) {

if (voter.voterID == voterID && voter.password == password && !voter.hasVoted) {

return true;

}

}

return false;

}

*public*:

Election() : totalVotes(0) {}

*void* registerVoter(*int* id, string name, *int* age, string address, string password) {

voters.push\_back(Voter(id, name, age, address, password));

}

*void* addCandidate(*int* id, string name, string party) {

candidates.push\_back(Candidate(id, name, party));

}

*void* castVote(*int* voterID, string password, *int* candidateID) {

if (verifyVoter(voterID, password)) {

for (*auto* &voter : voters) {

if (voter.voterID == voterID) {

voter.hasVoted = true;

for (*auto* &candidate : candidates) {

if (candidate.candidateID == candidateID) {

candidate.votes++;

totalVotes++;

cout << "Vote cast successfully by " << voter.name << "!!" << endl;

displayRealTimeResults();

return;

}

}

}

}

} else {

cout << "Voter verification failed for " << voterID << "!" << endl;

}

}

*void* tallyResults() {

cout << "Final Election Results:\n";

for (*auto* &candidate : candidates) {

cout << "Candidate: " << candidate.name << " | Votes: " << candidate.votes << endl;

}

cout << "Total Votes Cast: " << totalVotes << endl;

}

*void* displayRealTimeResults() {

cout << "Real-Time Election Results:\n";

for (*auto* &candidate : candidates) {

cout << "Candidate: " << candidate.name << " | Votes: " << candidate.votes << endl;

}

cout << "Total Votes So Far: " << totalVotes << endl;

}

};

*int* main() {

Election election;

election.registerVoter(1, "Tharun", 22, "Indhravathi-s9", "pass123");

election.registerVoter(2, "Harsha", 24, "Indhravathi-s1", "harsha456");

election.registerVoter(3, "Navya", 23, "Girlshostel-f10", "navya789");

election.registerVoter(4, "Santhu", 25, "Indhravathi-s3", "santhu001");

election.registerVoter(5, "Revak", 21, "Indhravathi-s4", "revak999");

election.addCandidate(1, "Chirag", "Party A");

election.addCandidate(2, "Shreya", "Party B");

election.addCandidate(3, "Ash", "Party C");

election.castVote(1, "pass123", 1);

election.castVote(2, "harsha456", 2);

election.castVote(3, "navya789", 1);

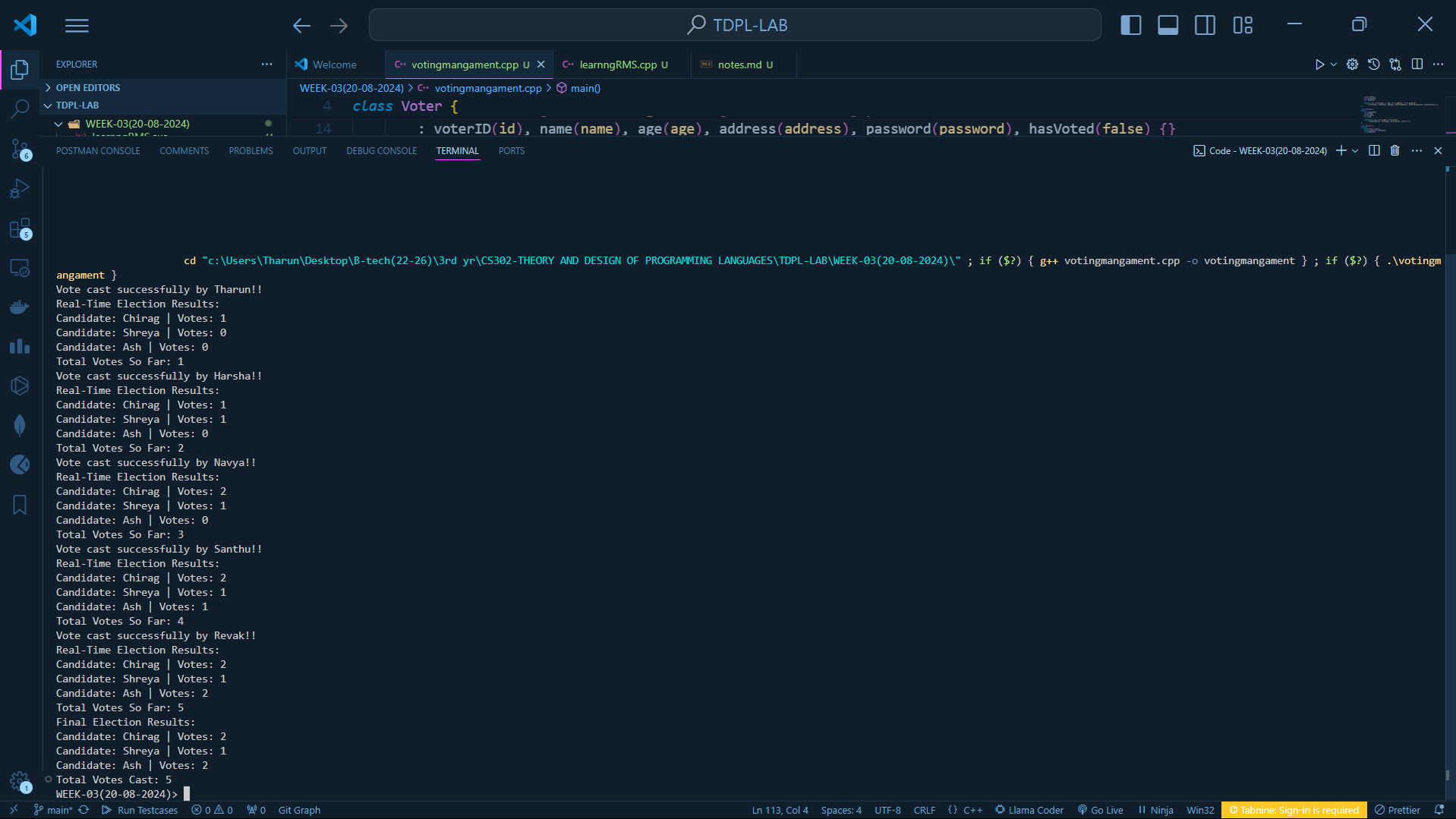
election.castVote(4, "santhu001", 3);

election.castVote(5, "revak999", 3);

election.tallyResults();

return 0;

}



#include <bits/stdc++.h>

using *namespace* std;

*class* User

{

*public*:

*int* userID;

string name;

vector<*int*> borrowedResources;

map<*int*, string> reviews;

User(*int* id, string name) : userID(id), name(name) {}

*void* addReview(*int* resourceID, *const* string *&*review)

{

reviews[resourceID] = review;

cout << name << " added a review for resource ID " << resourceID << ": " << review << endl;

}

};

*class* Resource

{

*public*:

*int* resourceID;

string title;

string author;

*bool* isBorrowed;

vector<string> userReviews;

Resource(*int* id, *const* string *&*title, *const* string *&*author)

: resourceID(id), title(title), author(author), isBorrowed(false) {}

*void* addReview(*const* string *&*review)

{

userReviews.push\_back(review);

}

*void* displayReviews() *const*

{

cout << "Reviews for " << title << ":\n";

for (*const* string &review : userReviews)

{

cout << "- " << review << endl;

}

}

};

*class* Library

{

*private*:

vector<User> users;

*const* *int* borrowDuration = 7 \* 24 \* 60 \* 60;

*public*:

vector<Resource> resources;

*void* registerUser(*int* id, *const* string *&*name)

{

users.push\_back(User(id, name));

}

*void* addResource(*int* id, *const* string *&*title, *const* string *&*author)

{

resources.push\_back(Resource(id, title, author));

}

*void* borrowResource(*int* userID, *int* resourceID)

{

for (*auto* &user : users)

{

if (user.userID == userID)

{

for (*auto* &resource : resources)

{

if (resource.resourceID == resourceID && !resource.isBorrowed)

{

resource.isBorrowed = true;

user.borrowedResources.push\_back(resourceID);

cout << user.name << " borrowed " << resource.title << " successfully!" << endl;

return;

}

}

}

}

cout << "Borrowing failed!" << endl;

}

*void* returnResource(*int* userID, *int* resourceID)

{

for (*auto* &user : users)

{

if (user.userID == userID)

{

for (*auto* &resource : resources)

{

if (resource.resourceID == resourceID && resource.isBorrowed)

{

resource.isBorrowed = false;

user.borrowedResources.erase(

remove(user.borrowedResources.begin(), user.borrowedResources.end(), resourceID),

user.borrowedResources.end());

cout << user.name << " returned " << resource.title << " successfully!" << endl;

return;

}

}

}

}

cout << "Return failed!" << endl;

}

*void* notifyOverdue()

{

*time\_t* now = time(0);

for (*const* *auto* &user : users)

{

for (*const* *auto* &borrowed : user.borrowedResources)

{

*time\_t* borrowTime = now - borrowDuration;

if (difftime(now, borrowTime) > borrowDuration)

{

cout << "Overdue notification: " << user.name << ", please return resource ID " << borrowed << " immediately!" << endl;

}

}

}

}

*void* addReview(*int* userID, *int* resourceID, *const* string *&*review)

{

for (*auto* &user : users)

{

if (user.userID == userID)

{

for (*auto* &resource : resources)

{

if (resource.resourceID == resourceID)

{

user.addReview(resourceID, review);

resource.addReview(review);

return;

}

}

}

}

}

*void* recommendResources() *const*

{

cout << "Resource Recommendations based on popular reviews:\n";

for (*const* *auto* &resource : resources)

{

if (!resource.userReviews.empty())

{

cout << resource.title << " by " << resource.author << endl;

}

}

}

};

*int* main()

{

Library library;

library.registerUser(1, "Tharun");

library.registerUser(2, "Harsha");

library.registerUser(3, "Navya");

library.registerUser(4, "Chirag");

library.addResource(1, "C++ Programming", "Bjarne Stroustrup");

library.addResource(2, "Effective Java", "Joshua Bloch");

library.addResource(3, "Clean Code", "Robert C. Martin");

library.borrowResource(1, 1);

library.borrowResource(2, 2);

library.returnResource(1, 1);

library.addReview(1, 1, "A must-read for every C++ developer!");

library.addReview(2, 2, "Excellent resource for Java enthusiasts.");

library.addReview(3, 3, "Clean Code is a great book on software craftsmanship.");

for (*const* *auto* &resource : library.resources)

{

resource.displayReviews();

}

library.recommendResources();

library.notifyOverdue();

return 0;

}

