#include<iostream>

#include <limits>

#include<fstream>

#include <cctype>

using namespace std;

const int MAX\_FACILITIES = 5;

bool isValidString(const string& input) {

for (char ch : input) {

if (!isalpha(ch) && ch != ' ') {

return false;

}

}

return true;

}

template <typename T>

T getValidatedInput(const string& prompt) {

T input;

bool valid = false;

while (!valid) {

cout << prompt;

if constexpr (is\_same<T, string>::value) {

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

getline(cin, input);

if (isValidString(input)) {

valid = true;

} else {

cout << "Invalid input. Please enter a valid string (letters and spaces only): " << endl;

}

} else if constexpr (is\_same<T, int>::value) {

if (cin >> input) {

valid = true;

} else {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input. Please enter a valid integer: " << endl;

}

}

}

return input;

}

class Patient {

public:

int id;

string name;

int age;

string disease;

int phone;

string assignedDoctor;

int bedNumber;

Patient\* next;

Patient(int id, string name, int age, string disease, int phone, string doctor, int bed) {

this->id = id;

this->name = name;

this->age = age;

this->disease = disease;

this->phone = phone;

this->assignedDoctor = doctor;

this->bedNumber = bed;

this->next = nullptr;

}

};

class Doctor {

public:

int id;

string name;

int age;

int phone;

Doctor(int id = 0, string name = "", int age = 0, int phone = 0) {

this->id = id;

this->name = name;

this->age = age;

this->phone = phone;

}

};

class HospitalGraph {

private:

string facilities[MAX\_FACILITIES];

int adjMatrix[MAX\_FACILITIES][MAX\_FACILITIES];

int facilityCount;

int getFacilityIndex(string facility) {

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

if (facilities[i] == facility) {

return i;

}

}

return -1;

}

public:

HospitalGraph() {

facilityCount = 0;

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

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

adjMatrix[i][j] = (i == j) ? 0 : INT\_MAX;

}

}

}

void addFacility(string facility) {

if (facilityCount >= MAX\_FACILITIES) {

cout << "Maximum facility limit reached.\n";

return;

}

facilities[facilityCount++] = facility;

}

void addConnection(string facility1, string facility2, int distance) {

int index1 = getFacilityIndex(facility1);

int index2 = getFacilityIndex(facility2);

if (index1 == -1 || index2 == -1) {

cout << "One or both facilities not found.\n";

return;

}

adjMatrix[index1][index2] = distance;

adjMatrix[index2][index1] = distance;

}

void displayGraph() {

cout << "\nHospital Facilities and Connections:\n";

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

cout << facilities[i] << " -> ";

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

if (adjMatrix[i][j] != INT\_MAX && i != j) {

cout << facilities[j] << " (" << adjMatrix[i][j] << "m), ";

}

}

cout << "\n";

}

}

void findShortestPath(string start, string end) {

int startIndex = getFacilityIndex(start);

int endIndex = getFacilityIndex(end);

if (startIndex == -1 || endIndex == -1) {

cout << "One or both facilities not found.\n";

return;

}

bool visited[MAX\_FACILITIES] = {false};

int distance[MAX\_FACILITIES];

int previous[MAX\_FACILITIES];

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

distance[i] = INT\_MAX;

previous[i] = -1;

}

distance[startIndex] = 0;

for (int count = 0; count < facilityCount - 1; count++) {

int minDist = INT\_MAX, minIndex;

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

if (!visited[i] && distance[i] < minDist) {

minDist = distance[i];

minIndex = i;

}

}

visited[minIndex] = true;

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

if (!visited[i] && adjMatrix[minIndex][i] != INT\_MAX &&

distance[minIndex] != INT\_MAX &&

distance[minIndex] + adjMatrix[minIndex][i] < distance[i]) {

distance[i] = distance[minIndex] + adjMatrix[minIndex][i];

previous[i] = minIndex;

}

}

}

if (distance[endIndex] == INT\_MAX) {

cout << "No path exists between " << start << " and " << end << ".\n";

return;

}

cout << "\nShortest Path from " << start << " to " << end << " is " << distance[endIndex] << " meters.\n";

cout << "Path: ";

int path[MAX\_FACILITIES];

int pathCount = 0;

for (int i = endIndex; i != -1; i = previous[i]) {

path[pathCount++] = i;

}

for (int i = pathCount - 1; i >= 0; i--) {

cout << facilities[path[i]];

if (i > 0) cout << " -> ";

}

cout << "\n";

}

};

class Hospital {

private:

Patient\* head;

Doctor doctors[5];

int totalBeds;

bool beds[20];

int minHeap[100];

int heapSize;

public:

Hospital() {

head = nullptr;

heapSize = 0;

totalBeds = 20;

for (int i = 0; i < 20; i++) beds[i] = false;

for (int i = 0; i < 5; i++) {

doctors[i] = Doctor(i + 1, "", 0, 0);

}

loadData();

}

~Hospital() {

Patient\* temp;

while (head != nullptr) {

temp = head;

head = head->next;

delete temp;

}

}

void saveData() {

ofstream patientFile("patients.txt");

Patient\* temp = head;

while (temp) {

patientFile << temp->id << " " << temp->name << " " << temp->age << " " << temp->disease << " "

<< temp->phone << " " << temp->assignedDoctor << " " << temp->bedNumber << endl;

temp = temp->next;

}

patientFile.close();

ofstream doctorFile("doctors.txt");

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

if (!doctors[i].name.empty()) {

doctorFile << doctors[i].id << " " << doctors[i].name << " " << doctors[i].age << " " << doctors[i].phone << endl;

}

}

doctorFile.close();

ofstream bedFile("beds.txt");

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

bedFile << beds[i] << " ";

}

bedFile.close();

cout<<"\n ''''''''''''''''''''''''''''''''''\n"<<" Data Saved Successfully\n"<<" ''''''''''''''''''''''''''''''''''\n";

}

void loadData() {

ifstream patientFile("patients.txt");

if (patientFile.is\_open()) {

int id, age, bed,phone;

string name, disease,doctor;

while (patientFile >> id >> name >> age >> disease >> phone >> doctor >> bed) {

addPatient(id, name, age, disease, phone, doctor, bed);

}

patientFile.close();

}

ifstream doctorFile("doctors.txt");

if (doctorFile.is\_open()) {

int id, age,phone;

string name;

while (doctorFile >> id >> name >> age >> phone) {

doctors[id - 1] = Doctor(id, name, age, phone);

}

doctorFile.close();

}

ifstream bedFile("beds.txt");

if (bedFile.is\_open()) {

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

bedFile >> beds[i];

}

bedFile.close();

}

cout<<"\n ''''''''''''''''''''''''''''''''''\n"<<" Data Loaded Successfully\n"<<" ''''''''''''''''''''''''''''''''''\n";

}

void addPatient(int id, string name, int age, string disease, int phone, string doctor, int bed) {

Patient\* newPatient = new Patient(id, name, age, disease, phone, doctor, bed);

newPatient->next = head;

head = newPatient;

}

void displayPatients() {

if (!head) {

cout << "No patients available.\n";

return;

}

Patient\* temp = head;

while (temp) {

cout << "\nPatient Details\n";

cout << "ID: " << temp->id << "\nName: " << temp->name << "\nAge: " << temp->age;

cout << "\nDisease: " << temp->disease << "\nPhone: " << temp->phone;

cout << "\nDoctor: " << temp->assignedDoctor << "\nBed: " << temp->bedNumber << "\n";

temp = temp->next;

}

}

Patient\* searchPatientByID(int id) {

Patient\* temp = head;

while (temp) {

if (temp->id == id) return temp;

temp = temp->next;

}

return nullptr;

}

void searchPatientByName(string name) {

Patient\* temp = head;

bool found = false;

while (temp) {

if (temp->name == name) {

cout << "\nPatient Found:\n";

cout << "ID: " << temp->id << ", Name: " << temp->name << ", Age: " << temp->age << "\n";

found = true;

}

temp = temp->next;

}

if (!found) {

cout << "No patient found with name: " << name << "\n";

}

}

void addDoctor(int id, string name, int age, int phone) {

if (id < 1 || id > 5) {

cout << "Invalid Doctor ID." << endl;

return;

}

doctors[id - 1] = Doctor(id, name, age, phone);

}

void displayDoctors() {

for (int i = 0; i < 5; i++) {

if (doctors[i].name != "") {

cout << "ID: " << doctors[i].id << ", Name: " << doctors[i].name;

cout << ", Age: " << doctors[i].age << ", Phone: " << doctors[i].phone << endl;

}

}

}

int assignBed() {

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

if (!beds[i]) {

beds[i] = true;

return i + 1;

}

}

return -1;

}

void releaseBed(int bedNumber) {

if (bedNumber > 0 && bedNumber <= totalBeds) {

beds[bedNumber - 1] = false;

}

}

void displayBedStatus() {

cout << "\nBed Status:\n";

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

cout << "Bed " << i + 1 << ": " << (beds[i] ? "Occupied" : "Available") << "\n";

}

}

void addEmergencyPatient(int priority) {

minHeap[heapSize++] = priority;

int i = heapSize - 1;

while (i != 0 && minHeap[(i - 1) / 2] > minHeap[i]) {

swap(minHeap[i], minHeap[(i - 1) / 2]);

i = (i - 1) / 2;

}

}

void viewEmergencyPatients() {

if (heapSize == 0) {

cout << "No emergency patients.\n";

return;

}

cout << "Emergency Patients (Priorities):\n";

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

cout << minHeap[i] << " ";

}

cout << "\n";

}

void ownerMenu() {

int choice;

do {

cout << "\n------------Welcome to the Owner Menu------------\n";

cout << "1. Add Doctor\n2. Display Doctors\n3. Display All Patients\n4. Display Bed Status\n5. Exit\n";

choice = getValidatedInput<int>("Enter your Desired Menu: ");

switch (choice) {

case 1: {

cout<<"\n------------Adding Doctor------------\n";

int id, age, phone;

string name;

id = getValidatedInput<int>("Enter ID: ");

name = getValidatedInput<string>("Enter name: ");

age = getValidatedInput<int>("Enter age: ");

phone = getValidatedInput<int>("Enter phone: ");

addDoctor(id, name, age, phone);

saveData();

break;

}

case 2:

loadData();

cout<<"------------Displaying Doctors------------\n";

displayDoctors();

break;

case 3:

loadData();

cout<<"------------Displaying Patients------------\n";

displayPatients();

break;

case 4:

loadData();

cout<<"------------Displaying Bed Status------------\n";

displayBedStatus();

break;

case 5:

return;

default:

cout << "Invalid choice!\n";

}

} while (choice != 5);

}

void doctorMenu() {

int choice;

do {

cout << "\n------------Welcome to the Doctor Menu------------\n";

cout << "1. View Patients\n2. Search Patient by ID\n3. Search Patient by Name\n4. Exit\n";

choice = getValidatedInput<int>("Enter your Desired Menu: ");

switch (choice) {

case 1:

loadData();

cout<<"------------Displaying Patients------------\n";

displayPatients();

break;

case 2: {

int id;

id = getValidatedInput<int>("Enter Patient ID: ");

loadData();

cout<<"------------Searching Patient By ID------------\n";

Patient\* patient = searchPatientByID(id);

if (patient) {

cout << "Patient Found: " << patient->name << ", Age: " << patient->age << "\n";

} else {

cout << "Patient not found!\n";

}

break;

}

case 3: {

string name;

name = getValidatedInput<string>("Enter Patient Name: ");

loadData();

cout<<"------------Searching Patient By Name------------\n";

searchPatientByName(name);

break;

}

case 4:

cout<<"\nExiting Doctor Menu! Good Bye";

return;

default:

cout << "Invalid choice!\n";

}

} while (choice != 4);

}

void receptionistMenu() {

int choice;

do {

cout << "\n------------Welcome to the Receptionist Menu------------\n";

cout << "1. Add Patient\n2. Assign Emergency Priority\n3. View Emergency Patients\n4. Display Bed Status\n5. Exit\n";

choice = getValidatedInput<int>("Enter your choice: ");

switch (choice) {

case 1: {

cout<<"------------Adding a Patient------------\n";

int id, age, phone;

string name, disease, doctor;

id = getValidatedInput<int>("Enter Patient ID: ");

name = getValidatedInput<string>("Enter Patient Name: ");

age = getValidatedInput<int>("Enter Patient Age: ");

disease = getValidatedInput<string>("Enter Patient Disease: ");

phone = getValidatedInput<int>("Enter Patient Phone Number: ");

loadData();

cout<<"------------Displaying Doctors------------\n";

displayDoctors();

doctor = getValidatedInput<string>("Enter Assigned Doctor's name: ");

int bed = assignBed();

if (bed == -1) {

cout << "No beds available.\n";

} else {

addPatient(id, name, age, disease, phone, doctor, bed);

cout << "Patient added with Bed Number: " << bed << "\n";

saveData();

}

break;

}

case 2: {

cout<<"------------Assign Emergency Priority------------\n";

int priority;

priority = getValidatedInput<int>("Enter Patient Emergency Priority: ");

addEmergencyPatient(priority);

break;

}

case 3:

cout<<"------------Viewing Emergecy Patients------------\n";

viewEmergencyPatients();

break;

case 4:

loadData();

cout<<"------------Displaying Bed Status------------\n";

displayBedStatus();

break;

case 5:

return;

default:

cout << "Invalid choice!\n";

}

} while (choice != 5);

}

void patientMenu() {

int choice;

do {

cout << "\n------------Welcome to the Patient Menu------------\n";

cout << "\n1. View Own Details\n2. Exit\n";

choice = getValidatedInput<int>("Enter your choice: ");

switch (choice) {

case 1: {

cout<<"------------View Own Details------------\n";

int id = getValidatedInput<int>("Enter your Patient ID: ");

loadData();

Patient\* patient = searchPatientByID(id);

if (patient) {

cout << "Patient Details: Name: " << patient->name << ", Disease: " << patient->disease;

cout << ", Assigned Doctor: " << patient->assignedDoctor << ", Bed Number: " << patient->bedNumber << "\n";

} else {

cout << "Patient not found!\n";

}

break;

}

case 2:

return;

default:

cout << "Invalid choice!\n";

}

} while (choice != 2);

}

};

int main() {

Hospital hospital;

HospitalGraph hospitalgraph;

hospitalgraph.addFacility("Reception");

hospitalgraph.addFacility("ICU");

hospitalgraph.addFacility("Ward A");

hospitalgraph.addFacility("Ward B");

hospitalgraph.addFacility("Pharmacy");

hospitalgraph.addConnection("Reception", "ICU", 50);

hospitalgraph.addConnection("Reception", "Ward A", 30);

hospitalgraph.addConnection("Ward A", "Ward B", 40);

hospitalgraph.addConnection("Ward B", "ICU", 20);

hospitalgraph.addConnection("ICU", "Pharmacy", 60);

const int numUsers = 4;

int validIds[numUsers] = {1, 232476, 232449, 232458};

int userId;

bool isAuthenticated = false;

int pass1 = 1;

int pass2 = 4321;

int pass3 = 1122;

int pass4 = 2211;

cout << "\n------------ Enter Login Credentials ------------";

cout << "\n------------ You only have 3 Tries ------------\n";

for (int attempts = 0; attempts < 3; attempts++) {

userId = getValidatedInput<int>("Enter User ID: ");

int pass = getValidatedInput<int>("Enter Password: ");

int i = 0;

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

if (userId == validIds[i] && pass == pass1 || pass == pass2 || pass == pass3 || pass == pass4) {

isAuthenticated = true;

break;

}

}

if (isAuthenticated) {

break;

} else {

cout<<"----------------Invalid ID----------------\n";

}

}

if (!isAuthenticated) {

cout << "Too many failed attempts. Exiting the program.\n";

return 0;

}

if(userId==232506 && isAuthenticated)

{

cout << "\n --------------------Welcome Haseeb--------------------\n --------------------Login successful!--------------------";

}

else if(userId==1 && isAuthenticated)

{

cout << "\n --------------------Welcome Abdullah--------------------\n --------------------Login successful!--------------------";

}

else if(userId==232476 && isAuthenticated)

{

cout << "\n --------------------Welcome Munawar--------------------\n --------------------Login successful!--------------------";

}

else if(userId==232458 && isAuthenticated)

{

cout << "\n --------------------Welcome Touqeer--------------------\n ------------Login successful!--------------------";

}

int userType;

string start, end;

do {

cout << "\n ------- Welcome to the Hospital Management System -------\n";

cout << "1- Owner Menu \n2- Doctor Menu \n3- Receptionist Menu \n4- Patient Menu \n";

cout << "5- Display Ward Connections \n6- Path Distance \n7- Exit Program\n";

userType = getValidatedInput<int>("Enter your choice: ");

switch (userType) {

case 1:

hospital.ownerMenu();

break;

case 2:

hospital.doctorMenu();

break;

case 3:

hospital.receptionistMenu();

break;

case 4:

hospital.patientMenu();

break;

case 5:

cout<<"------------Display Ward Connections------------";

hospitalgraph.displayGraph();

break;

case 6:

start = getValidatedInput<string>("Enter the starting facility: ");

cout << "\nEnter the destination facility: ";

cin >> end;

hospitalgraph.findShortestPath(start, end);

break;

case 7:

cout << "\nExiting the Program! Goodbye!\n";

break;

default:

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

}

} while (userType != 7);

return 0;

}

OutPut ScreenShots:

