**24K-0555 Sharjeel M. Assingment#1**

**QUESTION 1**

#include<iostream>

#include<string>

using namespace std;

class Skill {

public:

int skillID;

string skillName, description;

Skill(int id, string name, string desc) : skillID(id), skillName(name), description(desc) {}

void showSkillDetails() {

cout << "Skill ID: " << skillID << "\nSkill Name: " << skillName << "\nDescription: " << description << endl;

}

void updateSkillDescription(string newDescription) {

description = newDescription;

}

};

class Sport {

public:

int sportID;

string name, description;

Skill\*\* requiredSkills;

int skillCount;

Sport(int id, string n, string desc) : sportID(id), name(n), description(desc), skillCount(0) {

requiredSkills = new Skill\*[10]; // Assuming max 10 skills per sport

}

~Sport() {

delete[] requiredSkills;

}

void addSkill(Skill\* s) {

requiredSkills[skillCount++] = s;

}

void removeSkill(int skillID) {

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

if (requiredSkills[i]->skillID == skillID) {

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

requiredSkills[j] = requiredSkills[j + 1];

}

skillCount--;

break;

}

}

}

};

class Student;

class Mentor {

public:

int mentorID;

string mentorName;

string\* sportsExpertise;

int expertiseCount;

int maxLearners;

Student\*\* assignedLearners;

int learnerCount;

Mentor(string name, int id, int maxL) : mentorName(name), mentorID(id), maxLearners(maxL), learnerCount(0), expertiseCount(0) {

assignedLearners = new Student\*[maxLearners];

sportsExpertise = new string[10]; // Assuming max 10 sports expertise

}

~Mentor() {

delete[] assignedLearners;

delete[] sportsExpertise;

}

bool assignLearner(Student\* s);

void removeLearner(Student\* s);

void viewLearners();

void provideGuidance();

void display();

};

class Student {

public:

string studentName;

int studentID;

string\* sportsInterests;

int interestCount;

Mentor\* assignedMentor;

Student(string name, int id) : studentName(name), studentID(id), assignedMentor(nullptr), interestCount(0) {

sportsInterests = new string[10]; // Assuming max 10 sports interests

}

~Student() {

delete[] sportsInterests;

}

void registerForMentorship(Mentor\* mentor) {

if (mentor->assignLearner(this)) {

assignedMentor = mentor;

cout << "Mentor " << mentor->mentorName << " assigned to " << studentName << endl;

} else {

cout << "Mentor has reached maximum capacity." << endl;

}

}

void updateSportsInterest(string sport) {

sportsInterests[interestCount++] = sport;

}

void viewMentorDetails() {

if (assignedMentor) assignedMentor->display();

else cout << "No mentor assigned." << endl;

}

void display();

};

bool Mentor::assignLearner(Student\* s) {

if (learnerCount < maxLearners) {

assignedLearners[learnerCount++] = s;

return true;

}

return false;

}

void Mentor::removeLearner(Student\* s) {

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

if (assignedLearners[i] == s) {

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

assignedLearners[j] = assignedLearners[j + 1];

}

learnerCount--;

break;

}

}

}

void Mentor::viewLearners() {

cout << "Learners under mentor " << mentorName << ":\n";

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

cout << assignedLearners[i]->studentName << " (ID: " << assignedLearners[i]->studentID << ")\n";

}

}

void Mentor::provideGuidance() {

cout << "Providing guidance to assigned learners." << endl;

}

void Mentor::display() {

cout << "Mentor Name: " << mentorName << "\nMentor ID: " << mentorID << "\nMax Learners: " << maxLearners << endl;

viewLearners();

}

void Student::display() {

cout << "Student Name: " << studentName << "\nStudent ID: " << studentID << "\nSports Interests: ";

for (int i = 0; i < interestCount; i++) cout << sportsInterests[i] << " ";

cout << endl;

viewMentorDetails();

}

int main() {

int choice;

Student student("John Doe", 101);

Mentor mentor("Dr. Smith", 202, 3);

Sport sport(1, "Tennis", "A sport played with rackets and a ball.");

Skill skill(1, "Backhand Stroke", "A fundamental tennis stroke.");

sport.addSkill(&skill);

while (true) {

cout << "\nMain Menu:\n";

cout << "1. Register (Student)\n";

cout << "2. Mentor Details\n";

cout << "3. Student Details\n";

cout << "4. Add Sport (Student)\n";

cout << "5. Add Learner (Mentor)\n";

cout << "6. Remove Learner (Mentor)\n";

cout << "7. Provide Guidance (Mentor)\n";

cout << "8. Add Skill (Sport)\n";

cout << "9. Exit\n";

cin >> choice;

if (choice == 9) break;

switch (choice) {

case 1:

student.registerForMentorship(&mentor);

break;

case 2:

mentor.display();

break;

case 3:

student.display();

break;

case 4:

{

string sport;

cout << "Enter sport: ";

cin >> sport;

student.updateSportsInterest(sport);

}

break;

case 5:

mentor.assignLearner(&student);

break;

case 6:

mentor.removeLearner(&student);

break;

case 7:

mentor.provideGuidance();

break;

case 8:

sport.addSkill(&skill);

cout << "Skill added to sport." << endl;

break;

default:

cout << "Invalid choice. Try again." << endl;

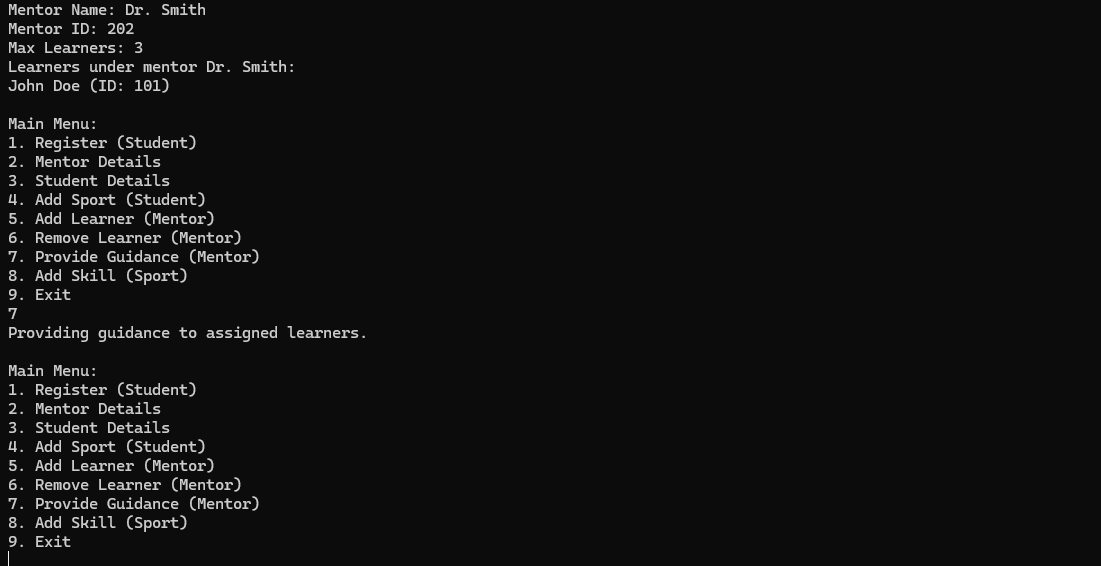
}

}

return 0;

}

**Output:**

**QUESTION #2**

**CODE:**

#include <iostream>

#include <cstdlib>

#include <algorithm>

using namespace std;

class Ball {

private:

int x, y;

static const int fieldMin = -9;

static const int fieldMax = 9;

public:

Ball() : x(0), y(0) {}

Ball(int x, int y) : x(x), y(y) {}

void move(int dx, int dy) {

int newX = x + dx;

int newY = y + dy;

if (newX >= fieldMin && newX <= fieldMax && newY >= fieldMin && newY <= fieldMax) {

x = newX;

y = newY;

} else {

cout << "Out of bounds! Ball reset to center." << endl;

x = 0;

y = 0;

}

}

int getX() const { return x; }

int getY() const { return y; }

void setX(int x) { this->x = x; }

void setY(int y) { this->y = y; }

void getPosition() const {

cout << "(" << x << ", " << y << ")";

}

};

class Goal {

private:

int goalX, goalY;

public:

Goal() : goalX(3), goalY(3) {}

bool isGoalReached(int ballX, int ballY) const {

return ballX == goalX && ballY == goalY;

}

void setGoalX(int x) { goalX = x; }

void setGoalY(int y) { goalY = y; }

int getGoalX() const { return goalX; }

int getGoalY() const { return goalY; }

};

class Robot {

private:

string name;

int hits;

public:

Robot(string n) : name(n), hits(0) {}

void hitBall(Ball &ball, string direction) {

transform(direction.begin(), direction.end(), direction.begin(), ::tolower);

int moveX = rand() % 3 + 1;

int moveY = rand() % 3 + 1;

if (direction == "up") ball.move(0, moveY);

else if (direction == "down") ball.move(0, -moveY);

else if (direction == "left") ball.move(-moveX, 0);

else if (direction == "right") ball.move(moveX, 0);

else {

cout << "Invalid direction! Please enter up, down, left, or right." << endl;

return;

}

hits++;

cout << name << " hit the ball. Ball is now at: ";

ball.getPosition();

cout << endl;

}

int getHits() const { return hits; }

string getName() const { return name; }

void setHits(int hits) { this->hits = hits; }

void setName(string name) { this->name = name; }

};

class Team {

private:

string teamName;

Robot\* player;

public:

Team(string name, Robot\* r) : teamName(name), player(r) {}

~Team() { delete player; }

Robot\* getPlayer() const { return player; }

string getTeamName() const { return teamName; }

};

class Game {

private:

Team\* teamOne;

Team\* teamTwo;

Ball ball;

Goal goal;

public:

Game(Team\* t1, Team\* t2) : teamOne(t1), teamTwo(t2) {}

~Game() { delete teamOne; delete teamTwo; }

void play(Team\* team) {

Robot\* player = team->getPlayer();

string direction;

while (!goal.isGoalReached(ball.getX(), ball.getY())) {

cout << "Enter direction (up, down, left, right): ";

cin >> direction;

player->hitBall(ball, direction);

}

}

void startGame(int playerChoice) {

ball = Ball();

if (playerChoice == 1) play(teamOne);

else play(teamTwo);

}

void declareWinner() const {

int hits1 = teamOne->getPlayer()->getHits();

int hits2 = teamTwo->getPlayer()->getHits();

cout << "Scoreboard:" << endl;

cout << teamOne->getTeamName() << " hits: " << hits1 << endl;

cout << teamTwo->getTeamName() << " hits: " << hits2 << endl;

if (hits1 < hits2)

cout << "Winner: " << teamOne->getTeamName() << "!" << endl;

else if (hits2 < hits1)

cout << "Winner: " << teamTwo->getTeamName() << "!" << endl;

else

cout << "It's a tie!" << endl;

}

void updateGoalPosition() {

int newX, newY;

cout << "Enter new goal X position: ";

cin >> newX;

cout << "Enter new goal Y position: ";

cin >> newY;

if (newX >= -9 && newX <= 9 && newY >= -9 && newY <= 9) {

goal.setGoalX(newX);

goal.setGoalY(newY);

cout << "Goal position updated to (" << newX << ", " << newY << ")." << endl;

} else {

cout << "Invalid goal position! Goal must be within (-9, -9) and (9, 9)." << endl;

}

}

};

int main() {

Robot\* r1 = new Robot("Alpha");

Robot\* r2 = new Robot("Beta");

Team\* t1 = new Team("Red", r1);

Team\* t2 = new Team("Blue", r2);

Game\* game = new Game(t1, t2);

int choice;

do {

cout << "\nFootball Game Menu:" << endl;

cout << "1. Play as Player 1 (Red)" << endl;

cout << "2. Play as Player 2 (Blue)" << endl;

cout << "3. Declare Winner" << endl;

cout << "4. Update Goal Position" << endl;

cout << "5. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

case 2:

game->startGame(choice);

break;

case 3:

game->declareWinner();

break;

case 4:

game->updateGoalPosition();

break;

case 5:

cout << "Exiting game. Goodbye!" << endl;

break;

default:

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

}

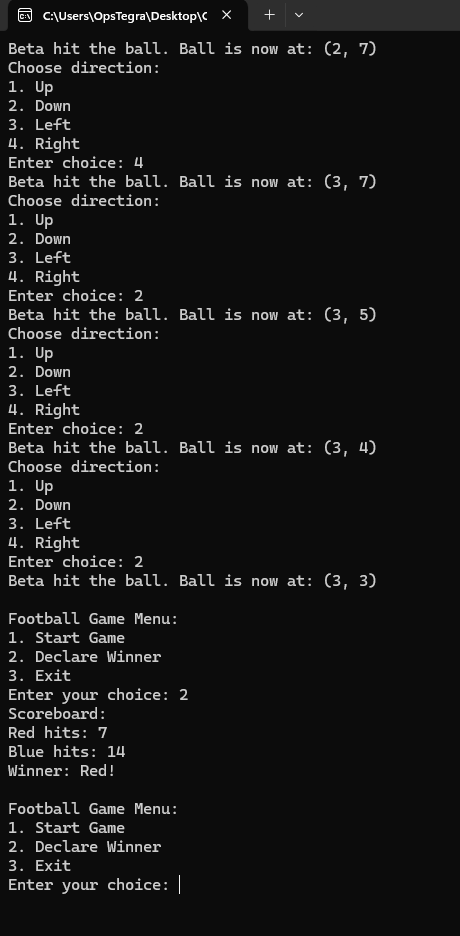
} while (choice != 5);

delete game;

return 0;

}

**Output:**



**QUESTION #3**

**CODE:**

#include <iostream>

#include <string>

using namespace std;

class Vehicle {

private:

string model;

double rentalPricePerDay;

string requiredLicenseType;

bool isRented;

public:

Vehicle(const string& model = "", double rentalPricePerDay = 0.0, const string& requiredLicenseType = "")

: model(model), rentalPricePerDay(rentalPricePerDay), requiredLicenseType(requiredLicenseType), isRented(false) {}

~Vehicle() {

cout << "Vehicle " << model << " has been deleted." << endl;

}

void setModel(const string& model) { this->model = model; }

void setRentalPricePerDay(double rentalPricePerDay) { this->rentalPricePerDay = rentalPricePerDay; }

void setRequiredLicenseType(const string& requiredLicenseType) { this->requiredLicenseType = requiredLicenseType; }

string getModel() const { return model; }

double getRentalPricePerDay() const { return rentalPricePerDay; }

string getRequiredLicenseType() const { return requiredLicenseType; }

bool getIsRented() const { return isRented; }

void rentVehicle() { isRented = true; }

void returnVehicle() { isRented = false; }

bool isEligible(const string& userLicenseType) const {

if (userLicenseType == "Full") return true;

if (userLicenseType == "Intermediate" && (requiredLicenseType == "Learner" || requiredLicenseType == "Intermediate")) return true;

return userLicenseType == requiredLicenseType;

}

};

class User {

private:

string name;

int age;

string licenseType;

string contactInfo;

string userID;

Vehicle\* rentedVehicle;

public:

User(const string& name = "", int age = 0, const string& licenseType = "", const string& contactInfo = "", const string& userID = "")

: name(name), age(age), licenseType(licenseType), contactInfo(contactInfo), userID(userID), rentedVehicle(nullptr) {}

~User() {

cout << "User " << name << " has been deleted." << endl;

}

void updateInfo() {

cout << "Enter updated Name: ";

getline(cin, name);

cout << "Enter updated Age: ";

cin >> age;

cin.ignore();

cout << "Enter updated License Type (Learner/Intermediate/Full): ";

getline(cin, licenseType);

cout << "Enter updated Contact Info: ";

getline(cin, contactInfo);

cout << "User information updated successfully!" << endl;

}

string getUserID() const { return userID; }

string getLicenseType() const { return licenseType; }

void rentVehicle(Vehicle\* vehicle) {

if (vehicle->isEligible(licenseType) && !vehicle->getIsRented()) {

rentedVehicle = vehicle;

vehicle->rentVehicle();

cout << name << " has successfully rented " << vehicle->getModel() << " for $" << vehicle->getRentalPricePerDay() << " per day." << endl;

} else {

cout << name << " is not eligible or vehicle is already rented." << endl;

}

}

void viewRentedVehicle() const {

if (rentedVehicle)

cout << name << " has rented: " << rentedVehicle->getModel() << endl;

else

cout << name << " has not rented any vehicle." << endl;

}

};

class VehicleRentalSystem {

private:

User\* users[100];

int userCount;

Vehicle\* vehicles[100];

int vehicleCount;

public:

VehicleRentalSystem() : userCount(0), vehicleCount(0) {}

~VehicleRentalSystem() {

for (int i = 0; i < userCount; ++i) delete users[i];

for (int i = 0; i < vehicleCount; ++i) delete vehicles[i];

}

void registerUser() {

string name, licenseType, contactInfo, userID;

int age;

cout << "Enter User ID: ";

getline(cin, userID);

cout << "Enter Name: ";

getline(cin, name);

cout << "Enter Age: ";

cin >> age;

cin.ignore();

cout << "Enter License Type (Learner/Intermediate/Full): ";

getline(cin, licenseType);

cout << "Enter Contact Info: ";

getline(cin, contactInfo);

users[userCount++] = new User(name, age, licenseType, contactInfo, userID);

cout << "User registered successfully!" << endl;

}

void updateUser() {

string userID;

cout << "Enter User ID to update: ";

getline(cin, userID);

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

if (users[i]->getUserID() == userID) {

users[i]->updateInfo();

return;

}

}

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

}

void addVehicle() {

string model, requiredLicenseType;

double rentalPricePerDay;

cout << "Enter Vehicle Model: ";

getline(cin, model);

cout << "Enter Rental Price Per Day: ";

cin >> rentalPricePerDay;

cin.ignore();

cout << "Enter Required License Type (Learner/Intermediate/Full): ";

getline(cin, requiredLicenseType);

vehicles[vehicleCount++] = new Vehicle(model, rentalPricePerDay, requiredLicenseType);

cout << "Vehicle added successfully!" << endl;

}

void viewAvailableVehicles() const {

cout << "Available Vehicles:" << endl;

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

if (!vehicles[i]->getIsRented())

cout << vehicles[i]->getModel() << " - $" << vehicles[i]->getRentalPricePerDay() << " per day" << endl;

}

}

void rentVehicle() {

string userID, model;

cout << "Enter User ID: ";

getline(cin, userID);

cout << "Enter Vehicle Model: ";

getline(cin, model);

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

if (users[i]->getUserID() == userID) {

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

if (vehicles[j]->getModel() == model) {

users[i]->rentVehicle(vehicles[j]);

return;

}

}

}

}

cout << "Invalid User ID or Vehicle Model." << endl;

}

};

int main() {

VehicleRentalSystem system;

int choice;

while (true) {

cout << "\n1. Register User\n2. Update User\n3. Add Vehicle\n4. View Available Vehicles\n5. Rent Vehicle\n6. Exit\nEnter choice: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1: system.registerUser(); break;

case 2: system.updateUser(); break;

case 3: system.addVehicle(); break;

case 4: system.viewAvailableVehicles(); break;

case 5: system.rentVehicle(); break;

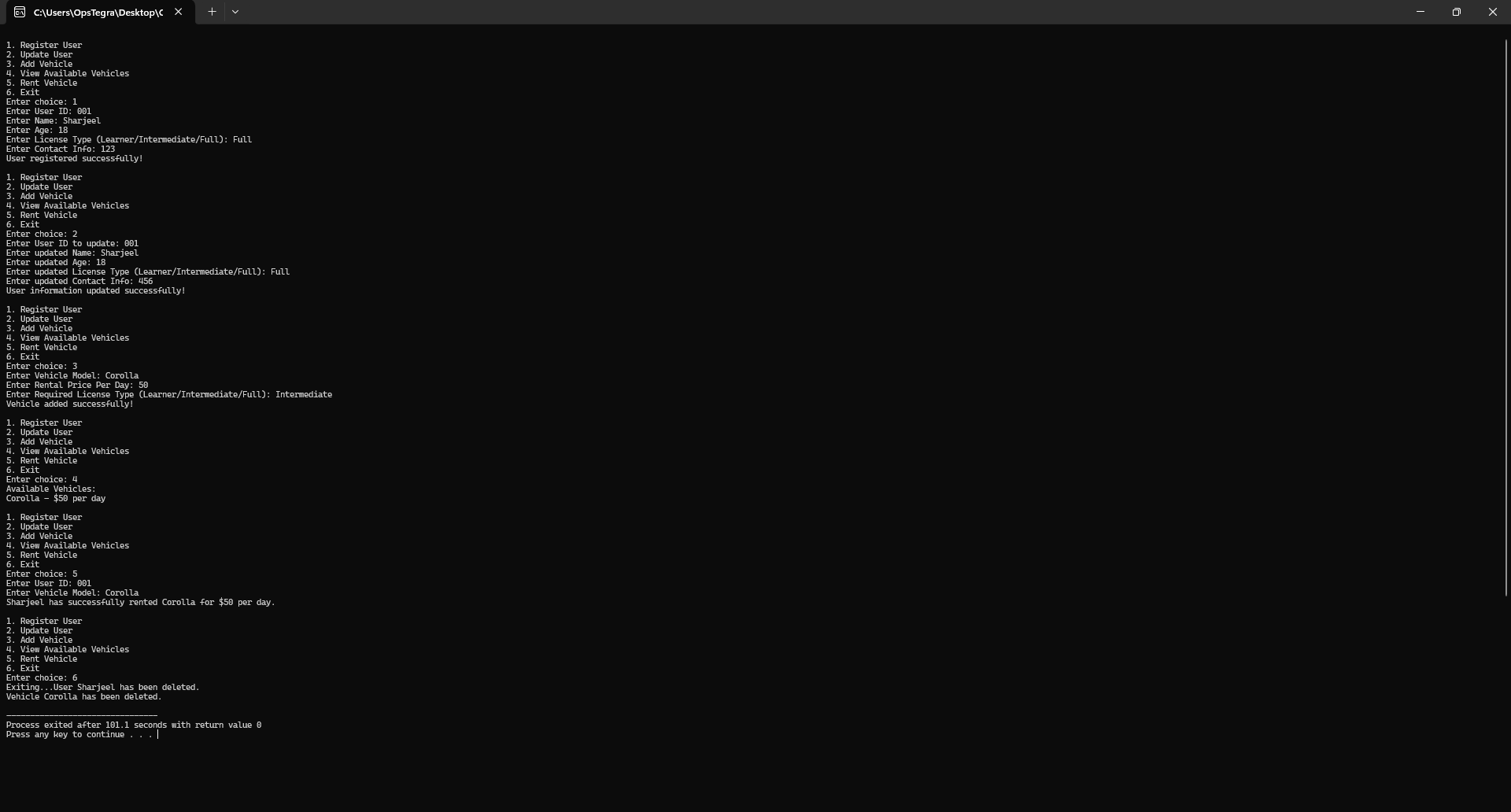
case 6: cout << "Exiting..."; return 0;

default: cout << "Invalid choice!";

}

}

}



**QUESTION #4**

#include <iostream>

#include <string>

using namespace std;

class Bus;

class Student {

static int student\_no;

int id;

string name;

bool transport\_card;

string pick\_up;

Bus \*assignedBus;

int attendance[30];

public:

Student() : id(0), transport\_card(false), assignedBus(nullptr) {

for (int &day : attendance) {

day = 0;

}

}

void payForTransportation() { transport\_card = true; }

void setValue() {

cout << "\n--- Registering a New Student ---\n";

cout << "Enter Student ID: ";

cin >> id;

cout << "Enter Student Name: ";

cin >> name;

cout << "Enter Your Pick-up Location: ";

cin >> pick\_up;

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

}

void setBus(Bus &b) { assignedBus = &b; }

int getId() const { return id; }

bool hasTransportCard() const { return transport\_card; }

string getPickUp() const { return pick\_up; }

Bus \*getBus() const { return assignedBus; }

int getAttendance() const;

void markAttendance();

~Student() {} // No need to delete assignedBus (aggregation)

};

class Bus {

static int day;

string \*stops;

int numberOfStops;

public:

Bus() : stops(nullptr), numberOfStops(0) {}

Bus(int size, string \*ptr) : numberOfStops(size) {

stops = new string[size];

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

stops[i] = ptr[i];

}

}

bool tapCard() { return true; }

int getSize() const { return numberOfStops; }

string \*getStops() const { return stops; }

void incrementDay() { day++; }

int getDay() const { return day; }

~Bus() { delete[] stops; }

};

void Student::markAttendance() {

if (assignedBus) {

attendance[assignedBus->getDay()] = 1;

}

}

int Student::getAttendance() const {

return assignedBus ? attendance[assignedBus->getDay()] : 0;

}

int Bus::day = 0;

int main() {

int total\_students = 0;

Student students[100]; // Fixed-size array to avoid dynamic allocation issues

Bus buses[3];

string r1[7] = {"University", "Hasan square", "Liaqtabad", "Nazimabad", "Johar more", "Dalmia", "Natha Khan"};

string r2[7] = {"University", "Shaheedemillat", "Guru mandir", "Garden", "Tariq route", "Baloch pull", "Jamat Khana"};

string r3[8] = {"University", "Korangi", "Landhi", "Akhtar colony", "Askari 3", "Kala pull", "Teen talwar", "Kpt bridge"};

buses[0] = Bus(7, r1);

buses[1] = Bus(7, r2);

buses[2] = Bus(8, r3);

int choice;

while (true) {

cout << "\n--- FAST Transport System ---\n";

cout << "1. Register New Student\n";

cout << "2. Mark Attendance\n";

cout << "3. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

if (total\_students < 100) {

students[total\_students].setValue();

total\_students++;

} else {

cout << "Error: Maximum student limit reached!\n";

}

}

else if (choice == 2) {

cout << "Enter Student ID: ";

int id;

cin >> id;

int i;

for (i = 0; i < total\_students; i++) {

if (id == students[i].getId()) break;

}

if (i < total\_students) {

if (students[i].hasTransportCard()) {

int a;

cout << "Tap your transport card to mark attendance (Press any number): ";

cin >> a;

if (!students[i].getAttendance()) {

buses[i % 3].incrementDay(); // Ensures days are updated

}

students[i].markAttendance();

cout << "Attendance Marked Successfully!\n";

} else {

cout << "\nYour transport card is inactive. Please pay your transport fees.\n";

cout << "Enter '1' to Pay Now: ";

int pay;

cin >> pay;

if (pay == 1) {

students[i].payForTransportation();

bool assigned = false;

for (int k = 0; k < 3 && !assigned; k++) {

for (int j = 0; j < buses[k].getSize() && !assigned; j++) {

if (students[i].getPickUp() == buses[k].getStops()[j]) {

students[i].setBus(buses[k]);

assigned = true;

}

}

}

if (assigned) {

cout << "Transport Card Activated! Bus Assigned Successfully.\n";

} else {

cout << "Error: Invalid Bus Stop Entered!\n";

}

}

}

} else {

cout << "Error: Invalid Student ID! Please try again.\n";

}

}

else if (choice == 3) {

cout << "Exiting... Thank you!\n";

break;

}

else {

cout << "Invalid option! Please try again.\n";

}

}

return 0;

}

