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

**QUESTION #1**

**CODE:**

**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;

}

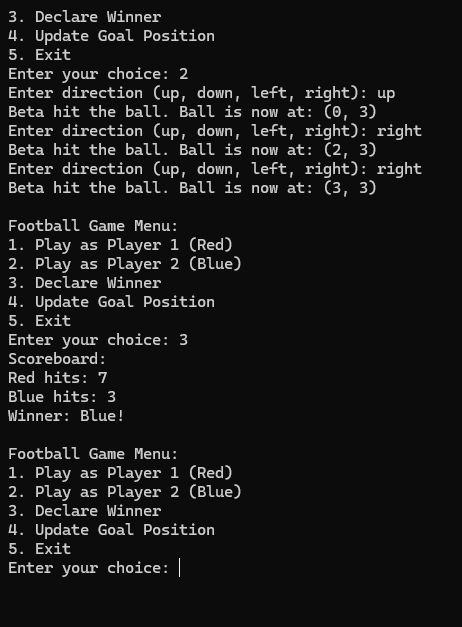
} while (choice != 5);

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;

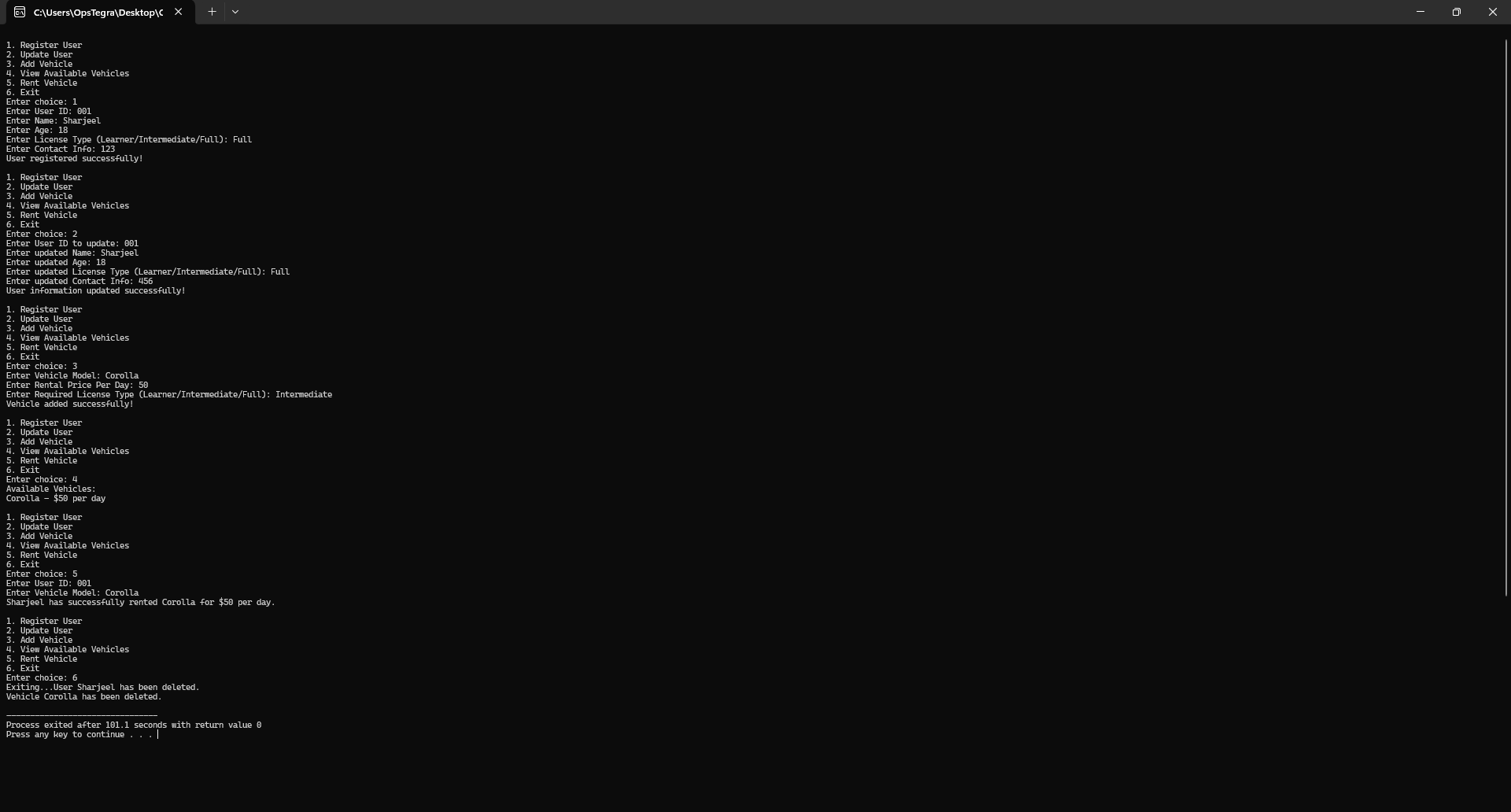
case 6: cout << "Exiting..."; return 0;

default: cout << "Invalid choice!";

}

}

}



**QUESTION #4**

**CODE:**