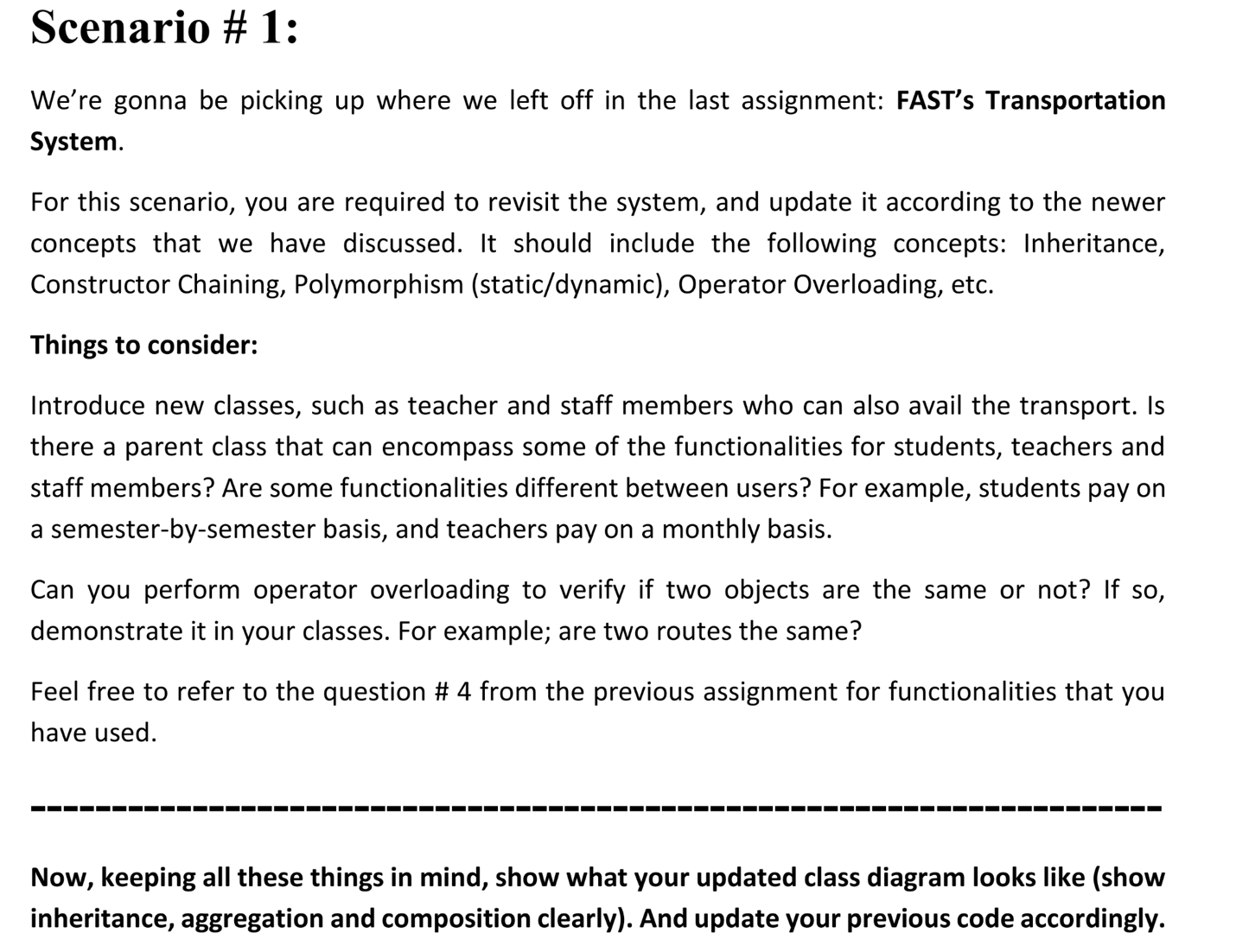
# Task 1



Code:

# include <iostream>

# include <string>

using namespace std;

class Person {

protected:

string name;

string ID;

bool fees\_paid;

string stop;

public:

Person(string name, string ID, bool fees\_paid, string stop)

: name(name), ID(ID), fees\_paid(fees\_paid), stop(stop) {}

virtual void viewDetails() const {

cout << "Name: " << name << "\nID: " << ID << "\nStop: " << stop << "\n";

}

virtual bool payFees() = 0;

string getStop() const { return stop; }

string getID() const { return ID; }

};

class Student : public Person {

int \*Attendance;

int days;

public:

Student(string name, string ID, bool fees\_paid, string stop, int days)

: Person(name, ID, fees\_paid, stop), days(days) {

Attendance = new int[days];

}

~Student() { delete[] Attendance; }

void markAttendance(int day) { Attendance[day - 1] = 1; }

bool payFees() override { return fees\_paid; }

void viewDetails() const override {

cout << "---- Student Details ----\n";

Person::viewDetails();

cout << "Fees Paid: " << (fees\_paid ? "Yes" : "No") << endl;

}

};

class Teacher : public Person {

public:

Teacher(string name, string ID, bool fees\_paid, string stop)

: Person(name, ID, fees\_paid, stop) {}

bool payFees() override { return fees\_paid; }

void viewDetails() const override {

cout << "---- Teacher Details ----\n";

Person::viewDetails();

cout << "Fees Paid: " << (fees\_paid ? "Yes" : "No") << endl;

}

};

class StaffMember : public Person {

public:

StaffMember(string name, string ID, bool fees\_paid, string stop)

: Person(name, ID, fees\_paid, stop) {}

bool payFees() override { return fees\_paid; }

void viewDetails() const override {

cout << "---- Staff Details ----\n";

Person::viewDetails();

cout << "Fees Paid: " << (fees\_paid ? "Yes" : "No") << "\n";

}

};

class Bus {

string name;

static int counter;

int id;

int max\_stops, max\_passengers, current\_passengers;

string\* busStops;

Person\*\* passengers;

public:

Bus(string name, int max\_stops, int max\_passengers)

: name(name), max\_stops(max\_stops), max\_passengers(max\_passengers) {

id = counter++;

busStops = new string[max\_stops];

passengers = new Person\*[max\_passengers];

current\_passengers = 0;

}

~Bus() {

delete[] busStops;

delete[] passengers;

}

void addStop(string stop, int index) {

if (index < max\_stops)

busStops[index] = stop;

}

void addPassenger(Person\* p) {

if (current\_passengers < max\_passengers) {

passengers[current\_passengers++] = p;

cout << p->getID() << " registered on bus " << id << "\n";

} else {

cout << "Bus full! Cannot register " << p->getID() << "\n";

}

}

bool operator==(const Bus& other) const {

if (max\_stops != other.max\_stops) return false;

for (int i = 0; i < max\_stops; i++) {

if (busStops[i] != other.busStops[i]) return false;

}

return true;

}

void viewDetails() {

cout << "Bus: " << name << " (ID: " << id << ")\nStops: ";

for (int i = 0; i < max\_stops; i++) cout << busStops[i] << " ";

cout << "\n";

}

};

int Bus::counter = 0;

int main() {

cout << "\nStudent ID: K240973" << endl;

cout << "Name: Talha Mirza\n" << endl;

Bus bus1("Route A", 3, 5);

bus1.addStop("A", 0);

bus1.addStop("B", 1);

bus1.addStop("C", 2);

Student s1("Talha", "K240973", true, "A", 100);

Teacher t1("Prof. Ahmed", "T120", true, "B");

StaffMember st1("Mr. Ali", "S340", true, "C");

bus1.addPassenger(&s1);

bus1.addPassenger(&t1);

bus1.addPassenger(&st1);

bus1.viewDetails();

s1.viewDetails();

t1.viewDetails();

st1.viewDetails();

return 0;

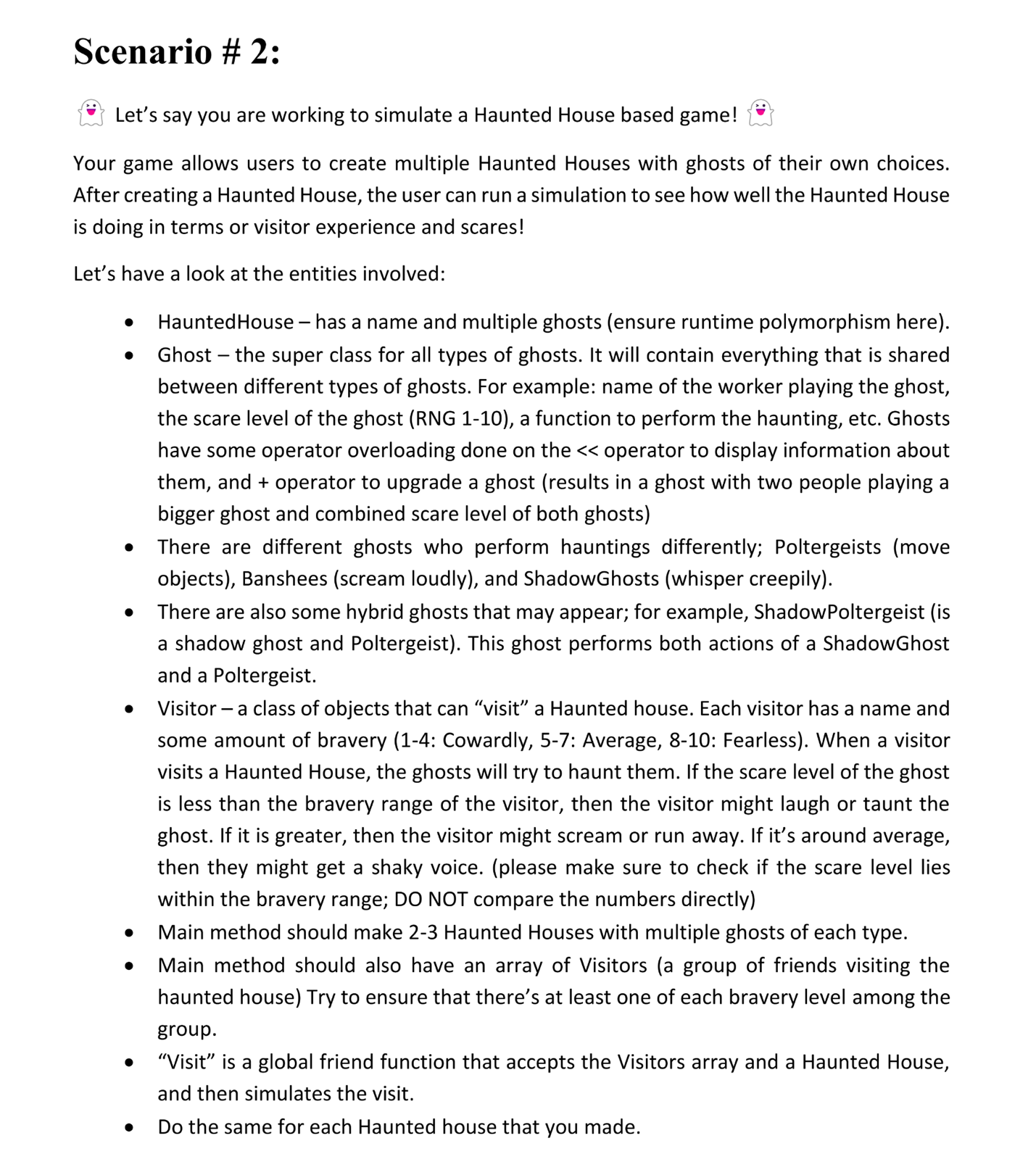
}

Screenshots:

A computer screen shot of a black screen

AI-generated content may be incorrect.

# Task 2



Code:

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

class Ghost {

protected:

string workerName;

int scareLevel;

public:

Ghost(string w, int s) : workerName(w), scareLevel(s) {}

virtual void haunt() = 0;

friend ostream& operator<<(ostream& os, Ghost& obj) {

os << "Worker Name: " << obj.workerName << ", Scare Level: " << obj.scareLevel << endl;

return os;

}

int getScareLevel() {return scareLevel;}

// \*\*\*

};

class Poltergeist : virtual public Ghost {

public:

Poltergeist(string w, int s) : Ghost(w,s) {}

void haunt() override {cout << "Poltergeist is moving objects!" << endl;}

Poltergeist\* operator+(Poltergeist& other) {

return new Poltergeist(workerName + " and " + other.workerName, scareLevel + other.getScareLevel());

}

};

class Banshee : public Ghost {

public:

Banshee(string w, int s) : Ghost(w,s) {}

void haunt() override {cout << "Banshee is screaming loudly!" << endl;}

Banshee\* operator+(Banshee& other) {

return new Banshee(workerName + " and " + other.workerName, scareLevel + other.getScareLevel());

}

};

class ShadowGhost : virtual public Ghost {

public:

ShadowGhost(string w, int s) : Ghost(w,s) {}

void haunt() override {cout << "ShadowGhost is whispering creepily!" << endl;}

ShadowGhost\* operator+(ShadowGhost& other) {

return new ShadowGhost(workerName + " and " + other.workerName, scareLevel + other.getScareLevel());

}

};

class ShadowPoltergeist : public ShadowGhost, public Poltergeist {

public:

ShadowPoltergeist(string w, int s) : ShadowGhost(w,s), Poltergeist(w,s), Ghost(w,s) {}

void haunt() override {

cout << "ShadowPoltergeist is whispering creepily and moving objects!" << endl;

}

ShadowPoltergeist\* operator+(ShadowPoltergeist& other) {

return new ShadowPoltergeist(workerName + " and " + other.workerName, scareLevel + other.getScareLevel());

}

};

class HauntedHouse {

string name;

Ghost\* ghosts[10];

int ghostCount = 0;

public:

HauntedHouse(string n) : name(n) {}

void addGhost(Ghost\* ghost) {

if (ghostCount < 10) {

ghosts[ghostCount++] = ghost;

}

}

void startHaunting() {

cout << "\nHaunted House: " << name << " begins haunting!" << endl;

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

ghosts[i]->haunt();

}

cout << endl;

}

string getName() {return name;}

Ghost\* getGhost(int index) {return ghosts[index];}

int getGhostCount() {return ghostCount;}

};

class Visitor {

string name;

int bravery;

public:

Visitor(string n, int b) : name(n), bravery(b) {}

void reactToHaunting(int scareLevel) {

if (scareLevel < bravery - 2) {cout << name << " laughs at the ghosts!" << endl;}

else if (scareLevel > bravery + 2) {cout << name << " screams and runs away!" << endl;}

else {cout << name << " is shaken but stays!" << endl;}

}

int getBravery() {return bravery;}

};

void Visit(Visitor\* visitors, int numV, HauntedHouse& house) {

cout << "\nVisitors are entering " << house.getName() << "..." << endl;

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

int totalScare = 0;

for (int j=0; j < house.getGhostCount(); j++) {

totalScare += house.getGhost(j)->getScareLevel();

}

visitors[i].reactToHaunting(totalScare);

}

}

int main() {

srand(time(0));

cout << "\nStudent ID: K240973" << endl;

cout << "Name: Talha Mirza\n" << endl;

HauntedHouse house1("Banshee's Lair");

HauntedHouse house2("Ghostly Manor");

house1.addGhost(new Poltergeist("Asad", rand() % 10 + 1));

house1.addGhost(new Banshee("Sarah", rand() % 10 + 1));

house2.addGhost(new ShadowGhost("Laiba", rand() % 10 + 1));

house2.addGhost(new ShadowPoltergeist("Izaan", rand() % 10 + 1));

Visitor visitors[3] = {Visitor("Ali", 9), Visitor("Saeed", 6), Visitor("Affan", 3)};

house1.startHaunting();

Visit(visitors, 3, house1);

house2.startHaunting();

Visit(visitors, 3, house2);

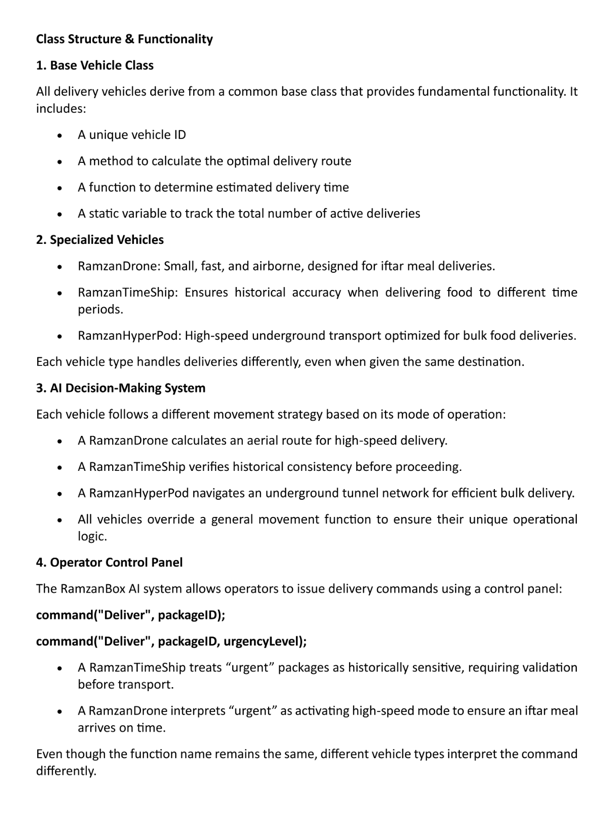
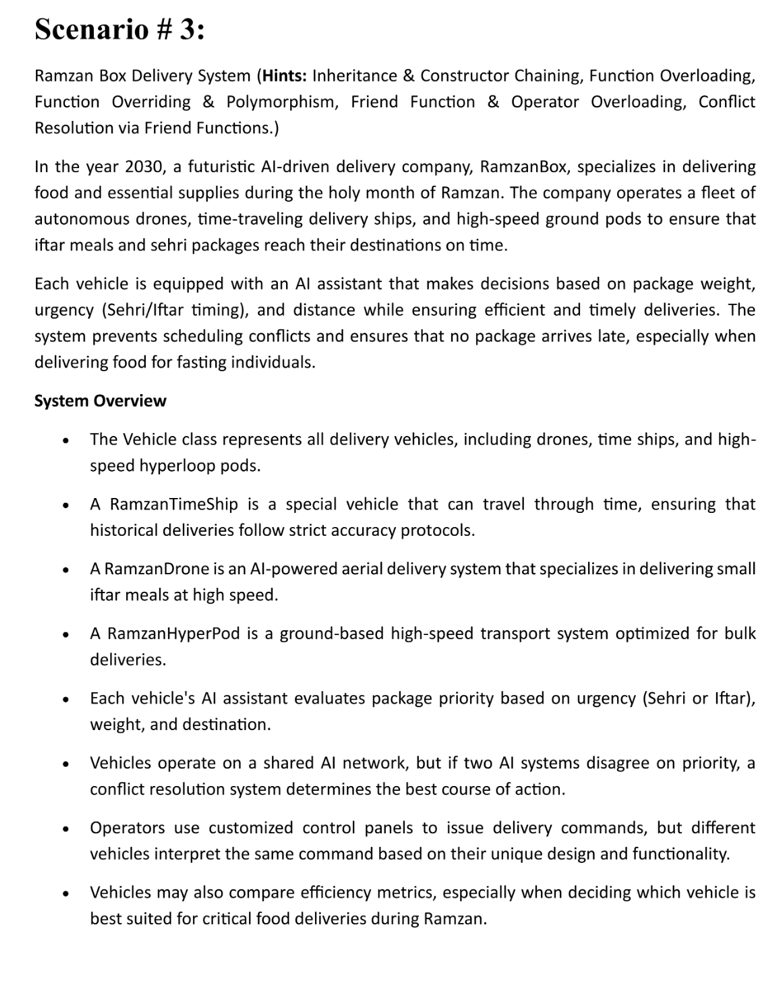
}

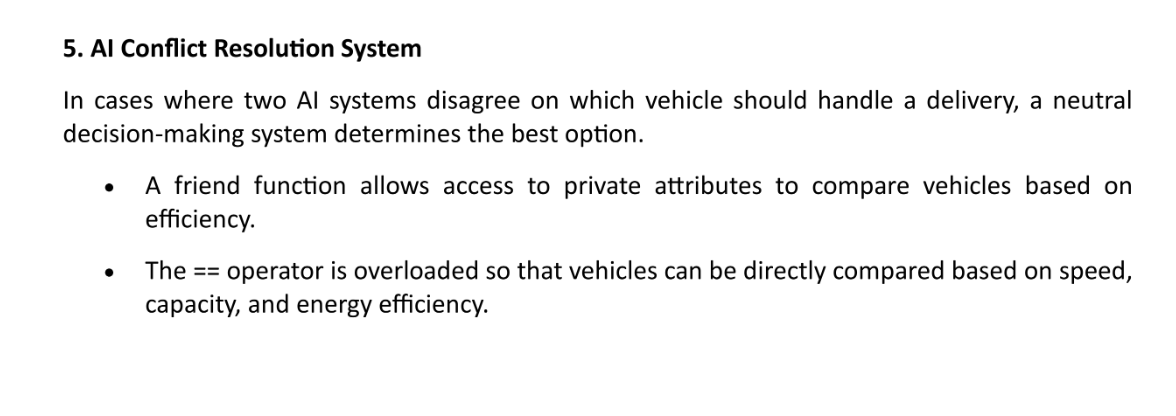
Screenshots:

A computer screen with white text

AI-generated content may be incorrect.

# Task 3





Code:

#include <iostream>

using namespace std;

class Vehicle {

protected:

int vehicleID;

float speed, capacity, energyEfficiency;

static int totalDeliveries;

public:

Vehicle(float s, float c, float e) : speed(s), capacity(c), energyEfficiency(e), vehicleID(++totalDeliveries) {}

virtual void optimalDeliveryRoute() = 0;

virtual void estimateDeliveryTime() = 0;

virtual void command(string cmd, int packageID) = 0;

virtual void command(string cmd, int packageID, string urgency) = 0;

friend bool operator==(const Vehicle& A, const Vehicle& B);

friend void AIConflictResolution(Vehicle& A, Vehicle& B);

int getID() {return vehicleID;}

};

bool operator==(const Vehicle& A, const Vehicle& B) {

return (A.speed == B.speed && A.capacity == B.capacity && A.energyEfficiency == B.energyEfficiency);

}

void AIConflictResolution(Vehicle& A, Vehicle& B) {

if (A == B) {

cout << "Order assigned twice to the same vehicle!" << endl;

} else {

int a = 0, b = 0;

if (A.speed > B.speed) {

cout << "In terms of speed, Vehicle " << A.vehicleID << "is better!" << endl;

a++;

} else {

cout << "In terms of speed, Vehicle " << B.vehicleID << "is better!" << endl;

b++;

}

if (A.capacity > B.capacity) {

cout << "In terms of capacity, Vehicle " << A.vehicleID << "is better!" << endl;

a++;

} else {

cout << "In terms of capacity, Vehicle " << B.vehicleID << "is better!" << endl;

b++;

}

if (A.energyEfficiency > B.energyEfficiency) {

cout << "In terms of Energy Efficiency, Vehicle " << A.vehicleID << "is better!" << endl;

a++;

} else {

cout << "In terms of Energy Efficiency, Vehicle " << B.vehicleID << "is better!" << endl;

b++;

}

if (a > b) {

cout << "According to AI Conflict Resolution System, Vehicle " << A.vehicleID << " is more suitable for executing the task!" << endl;

} else {

cout << "According to AI Conflict Resolution System, Vehicle " << B.vehicleID << " is more suitable for executing the task!" << endl;

}

}

}

int Vehicle::totalDeliveries = 0;

class RamzanDrone : public Vehicle {

public:

RamzanDrone(float s, float c, float e) : Vehicle(s,c,e) {}

void optimalDeliveryRoute() override {

cout << "Calculating optimal aerial route..." << endl;

}

void estimateDeliveryTime() override {

cout << "Estimating delivery time when using aerial route..." << endl;

}

void command(string cmd, int packageID) override {

cout <<"Vehicle " << vehicleID << " executing command '" << cmd << "' for package " << packageID << "using Drone Mode!" << endl;

}

void command(string cmd, int packageID, string urgency) override {

cout << "Vehicle " << vehicleID << " executing command '" << cmd << "' for package " << packageID << " with Urgency: " << urgency << " using Drone Mode!" << endl;

cout << "High-Speed Mode Activated!" << endl;

}

};

class RamzanTimeShip : public Vehicle {

public:

RamzanTimeShip(float s, float c, float e) : Vehicle(s, c, e) {}

void optimalDeliveryRoute() override {

cout << "Calculating optimal Time-Travel route..." << endl;

}

void estimateDeliveryTime() override {

cout << "Estimating delivery time when using Time-Travel..." << endl;

}

void command(string cmd, int packageID) override {

cout << "Vehicle " << vehicleID << " executing command '" << cmd << "' for package " << packageID << "using Time-Travel Mode!" << endl;

}

void command(string cmd, int packageID, string urgency) override {

cout << "Requiring Validation ! for Vehicle " << vehicleID << " to execute command '" << cmd << "' for package " << packageID << " with Urgency: " << urgency << " using Time-Travel Mode!" << endl;

}

};

class RamzanHyperPod : public Vehicle {

public:

RamzanHyperPod(float s, float c, float e) : Vehicle(s, c, e) {}

void optimalDeliveryRoute() override {

cout << "Calculating optimal Underground Transport Route..." << endl;

}

void estimateDeliveryTime() override {

cout << "Estimating delivery time when using Underground Transport route..." << endl;

}

void command(string cmd, int packageID) override {

cout << "Vehicle "<< vehicleID << " executing command '" << cmd << "' for package " << packageID << " using Underground Transport Mode!" << endl;

}

void command(string cmd, int packageID, string urgency) override {

cout << "Vehicle " << vehicleID << " executing commmand '" << cmd << "' for package " << packageID << " with Urgency: " << urgency << " using Hyper Pod Mode!" << endl;

cout << "Navigating through an underground tunnel network!" << endl;

}

};

int main() {

cout << "\nStudent ID: K240973" << endl;

cout << "Name: Talha Mirza\n" << endl;

RamzanDrone drone(95, 25, 75);

RamzanTimeShip timeShip(75, 80, 40);

RamzanHyperPod hyperPod(140, 150, 75);

Vehicle\* arr[3];

arr[0] = &drone;

arr[1] = &timeShip;

arr[2] = &hyperPod;

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

cout << "Vehicle ID: " << arr[i]->getID() << endl;

arr[i]->estimateDeliveryTime();

arr[i]->optimalDeliveryRoute();

arr[i]->command("Deliver", i+200);

}

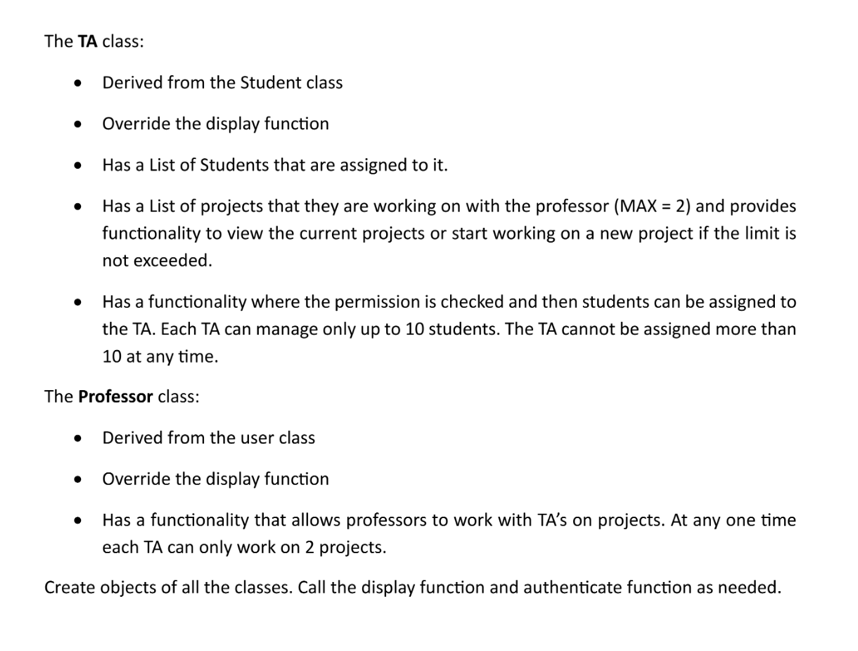
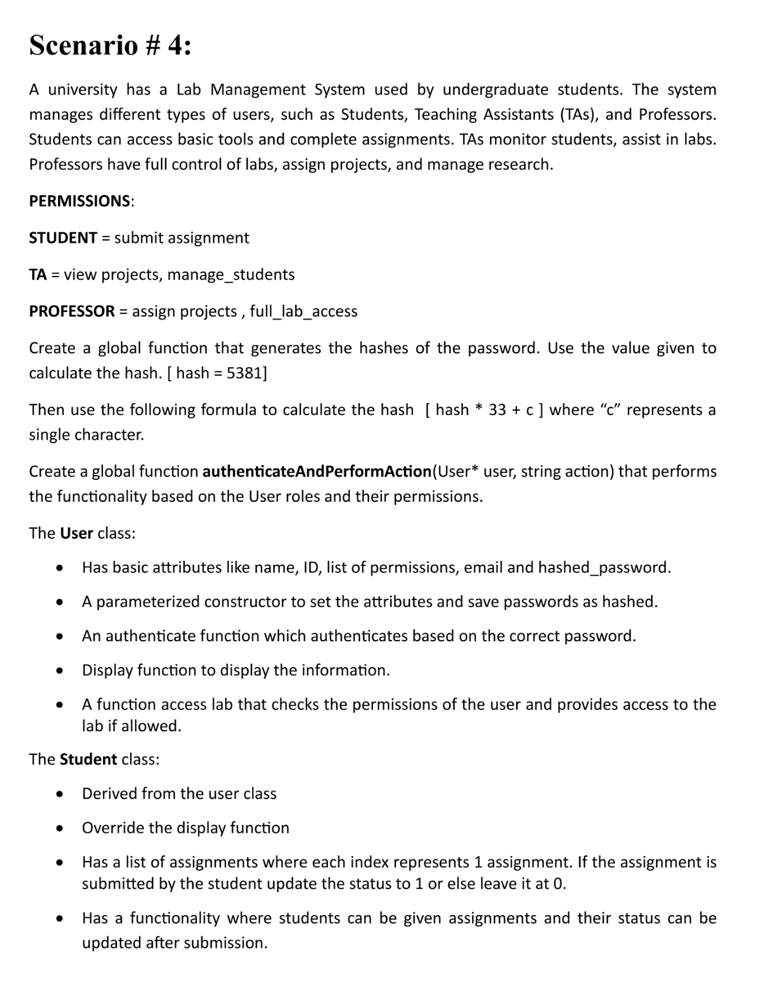
}

Screenshots:

A screenshot of a computer screen

AI-generated content may be incorrect.

# Task 4



Code:

#include <iostream>

#include <vector>

using namespace std;

class User;

const int hashValue = 5381;

int generateHash(string pwd) {

int hash = hashValue;

for (char c : pwd) {

hash = (hash \* 33) + c;

}

return hash;

}

class User {

protected:

string name, id, email;

int hashedPwd;

vector<string> permissionList;

public:

User() {}

User(string n, string id, string e, int h) : name(n), id(id), email(e), hashedPwd(h) {

permissionList.resize(2, ""); // Properly initializing the vector

}

bool authenticateUser(string pwd) {

return (hashedPwd == generateHash(pwd));

}

virtual void display() {

cout << "\nUsername: " << name << endl;

cout << "User ID: " << id << endl;

cout << "Email: " << email << endl;

cout << "Permissions given: " << endl;

bool hasPermissions = false;

for (const string &perm : permissionList) {

if (!perm.empty()) {

cout << "- " << perm << endl;

hasPermissions = true;

}

}

if (!hasPermissions) {

cout << "No Permissions Granted" << endl;

}

}

bool accessLab() {

for (const string &perm : permissionList) {

if (perm == "Lab is accessible") {

cout << "User " << id << " has access to the Lab" << endl;

return true;

}

}

cout << "User " << id << " does not have access to the Lab" << endl;

return false;

}

vector<string>& getList() { return permissionList; }

string getID() { return id; }

};

class Student : public User {

protected:

int assignmentList[10] = { -1 };

public:

Student() {}

Student(string n, string id, string e, int h) : User(n, id, e, h) {

permissionList[0] = "Submit Assignment";

}

void display() override {

cout << "The user is a student" << endl;

User::display();

}

void assignAssignment(int index) {

if (index >= 0 && index < 10) {

assignmentList[index] = 0;

cout << "Assignment " << index << " assigned successfully." << endl;

} else {

cout << "Invalid assignment index." << endl;

}

}

void submitAssignment(int index) {

string pwd;

cout << "Enter your password: ";

getline(cin, pwd);

if (authenticateUser(pwd)) {

if (index >= 0 && index < 10) {

if (assignmentList[index] == 0) {

assignmentList[index] = 1;

cout << "Assignment " << index << " submitted successfully." << endl;

} else if (assignmentList[index] == 1) {

cout << "The assignment has already been submitted." << endl;

} else {

cout << "No work assigned." << endl;

}

} else {

cout << "Invalid assignment index!" << endl;

}

} else {

cout << "Incorrect password!" << endl;

}

}

};

class TA : public Student {

private:

vector<Student\*> studentList; // max 10 students

vector<string> projectList; // can work on max 2 projects

public:

TA(string n, string id, string e, int h) : Student(n, id, e, h) {}

void display() override {

cout << "The user is a TA!" << endl;

User::display();

cout << "List of projects: ";

if (projectList.empty()) {

cout << "None" << endl;

} else {

for (size\_t i = 0; i < projectList.size(); i++) {

cout << "\n" << i + 1 << ": " << projectList[i];

}

cout << endl;

}

cout << "List of assigned students: ";

if (studentList.empty()) {

cout << "None" << endl;

} else {

for (size\_t i = 0; i < studentList.size(); i++) {

cout << "\n" << i + 1 << ": " << studentList[i]->getID();

}

cout << endl;

}

}

void newProject() {

string pwd;

cout << "Enter your password: ";

getline(cin, pwd);

if (authenticateUser(pwd)) {

if (projectList.size() < 2) {

string projectName;

cout << "Enter project name: ";

getline(cin, projectName);

projectList.push\_back(projectName);

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

} else {

cout << "TA is already working on two projects, cannot add more." << endl;

}

} else {

cout << "Incorrect password!" << endl;

}

}

void assignStudent(Student\* s) {

string password;

cout << "Enter your password: ";

cin >> password;

if (authenticateUser(password)) {

if (studentList.size() < 10) {

studentList.push\_back(s);

cout << "Student " << s->getID() << " assigned to TA " << name << endl;

} else {

cout << "TA already has 10 students assigned, cannot assign more." << endl;

}

} else {

cout << "Incorrect password!" << endl;

}

}

};

class Professor : public User {

public:

Professor(string n, string id, string e, int h) : User(n, id, e, h) {}

void display() override {

cout << "The user is a Professor" << endl;

User::display();

}

};

int main() {

cout << "\nStudent ID: K240973" << endl;

cout << "Name: Talha Mirza\n" << endl;

Student s1("Alice", "S123", "alice@example.com", generateHash("password123"));

TA ta1("Bob", "TA456", "bob@example.com", generateHash("ta\_pass"));

Professor prof1("Dr. Smith", "P789", "smith@example.com", generateHash("prof123"));

cout << "\nTesting Student Authentication:\n";

s1.authenticateUser("password123") ? cout << "Success\n" : cout << "Failure\n";

s1.assignAssignment(2);

s1.submitAssignment(2);

ta1.newProject();

ta1.assignStudent(&s1);

s1.display();

ta1.display();

prof1.display();

return 0;

}

Screenshots:

A screenshot of a computer

AI-generated content may be incorrect.