GitHub: <https://github.com/thetalhamirza/OOP-Theory-Assignment>

# Task 1

## Code:

#include <iostream>

#include <string>

using namespace std;

class Student;

class Mentor;

class Sport;

class Skill;

class Skill

{

private:

int ID;

string name;

string description;

public:

Skill() {}

Skill(int ID, string name, string d)

{

this->ID = ID;

this->name = name;

this->description = d;

}

void showSkillDetails()

{

cout << "\nSkill ID: " << ID << endl;

cout << "Skill Name: " << name << endl;

cout << "Description: " << description << endl;

}

void updateSkillDescription(string d)

{

description = d;

cout << "\nSkill description updated.\n";

}

int getSkillID()

{

return ID;

}

string getName()

{

return name;

}

string getDescription()

{

return description;

}

};

class Sport

{

private:

int ID;

string name;

string description;

Skill requiredSkills[3];

int skillCount;

public:

Sport(int id, string n, string d)

{

ID = id;

name = n;

description = d;

skillCount = 0;

}

Sport() {}

string get\_name()

{

return name;

}

void addSkill(Skill s)

{

if (skillCount < 3) {

requiredSkills[skillCount] = s;

skillCount++;

cout << "\nSkill added to " << name << endl;

} else {

cout << "\nCannot add more skills" << endl;

}

}

void removeSkill(int sID)

{

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

{

if (requiredSkills[i].getSkillID() == sID)

{

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

{

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

}

skillCount--;

cout << "Skill removed." << endl;

return;

}

}

cout << "\nSkill not found." << endl;

}

void showSportDetails()

{

cout << "\nSport ID: " << ID << endl;

cout << "Name: " << name << endl;

cout << "Description: " << description << endl;

cout << "Required Skills: " << endl;

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

{

requiredSkills[i].showSkillDetails();

}

}

int getSportID()

{

return ID;

}

};

class Mentor

{

private:

int mentorID;

string name;

Sport sportsExpertise[3];

int expertiseCount;

int maxLearners;

Student\* assignedLearners[5];

int learnerCount;

public:

Mentor() {

mentorID = 0;

name = "Unknown Mentor";

maxLearners = 5;

learnerCount = 0;

expertiseCount = 0;

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

{

assignedLearners[i] = 0;

}

}

Mentor(int id, string N, Sport expertise[], int max, int e, int l)

{

mentorID = id;

name = N;

maxLearners = max;

learnerCount = l;

expertiseCount = e;

for (int i = 0; i < expertiseCount; i++)

{

sportsExpertise[i] = expertise[i];

}

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

{

assignedLearners[i] = 0;

}

}

void display()

{

cout << "\nMentor Id: " << mentorID << endl << "Mentor Name: " << name << endl;

for (int i = 0; i < expertiseCount; i++)

{

cout << "Mentor expertise " << i + 1 << ":" << sportsExpertise[i].get\_name() << endl;

}

}

void assignLearner(Student\* s)

{

if (learnerCount < maxLearners)

{

assignedLearners[learnerCount++] = s;

cout << "Student assigned to mentor." << endl;

}

else

{

cout << "Cannot assign more learners." << endl;

}

}

void 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--;

cout << "Student removed!" << endl;

return;

}

}

cout << "Student not found!" << endl;

}

void provideGuidance()

{

cout << "Mentor " << name << " provides guidance." << endl;

}

string get\_name()

{

return name;

}

void viewLearners();

};

class Student

{

private:

int studentID;

string name;

int age;

Sport sportsInterest[3];

int interestCount;

Mentor\* mentorAssigned;

public:

Student()

{

studentID = 0;

name = "Unknown Student";

age = 0;

interestCount = 0;

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

{

sportsInterest[i] = Sport();

}

}

Student(int id, string n, int a, int c, Mentor \*m)

{

studentID = id;

name = n;

age = a;

interestCount = c;

mentorAssigned = m;

for (int i = 0; i < interestCount; i++)

{

sportsInterest[i] = Sport();

}

}

void display()

{

cout << "\nStudent's ID: " << studentID << endl;

cout << "Student's Name: " << name << endl;

cout << "Age: " << age << endl;

}

void registerForMentorship(Mentor \*m)

{

cout << "registering for mentorship under " << m->get\_name() << endl;

mentorAssigned = m;

}

void viewMentorDetails()

{

mentorAssigned->display();

}

string getName()

{

return name;

}

};

void Mentor::viewLearners()

{

if (learnerCount == 0)

{

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

return;

}

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

{

if (assignedLearners[i] != nullptr)

{

assignedLearners[i]->display();

}

else

{

cout << "Learner at index " << i << " is nullptr." << endl;

}

}

}

int main()

{

cout << "\nName: Talha Mirza" << endl;

cout << "24K-0973\n" << endl;

Skill skill1(101, "Dribbling", "Essential for basketball.");

Skill skill2(102, "Running", "Improves endurance.");

Skill skill3(103, "Passing", "Important in team games.");

cout << "Skills created." << endl;

skill1.showSkillDetails();

skill2.showSkillDetails();

skill3.showSkillDetails();

Sport basketball(1, "Basketball", "A team sport played with a ball.");

Sport football(2, "Football", "A game played with a round ball.");

Sport tennis(3, "Tennis", "A sport played with rackets and a ball.");

cout << "\nSports created." << endl;

basketball.showSportDetails();

football.showSportDetails();

tennis.showSportDetails();

basketball.addSkill(skill1);

basketball.addSkill(skill2);

football.addSkill(skill2);

football.addSkill(skill3);

cout << "\nUpdated Sports Details:" << endl;

basketball.showSportDetails();

football.showSportDetails();

Sport mentorExpertise[] = {basketball, football};

Mentor mentor1(201, "Coach Ali", mentorExpertise, 5, 2, 0);

Mentor mentor2(202, "Coach Sara", mentorExpertise, 5, 2, 0);

cout << "\nMentors created: " << endl;

mentor1.display();

mentor2.display();

Student student1(301, "Sami", 18, 2, &mentor1);

Student student2(302, "Nusrat", 19, 1, &mentor2);

cout << "\nStudents created: " << endl;

student1.display();

student2.display();

mentor1.assignLearner(&student1);

mentor2.assignLearner(&student2);

mentor1.viewLearners();

mentor2.viewLearners();

student1.registerForMentorship(&mentor2);

student2.registerForMentorship(&mentor1);

cout << "\nViewing mentor details from students: " << endl;

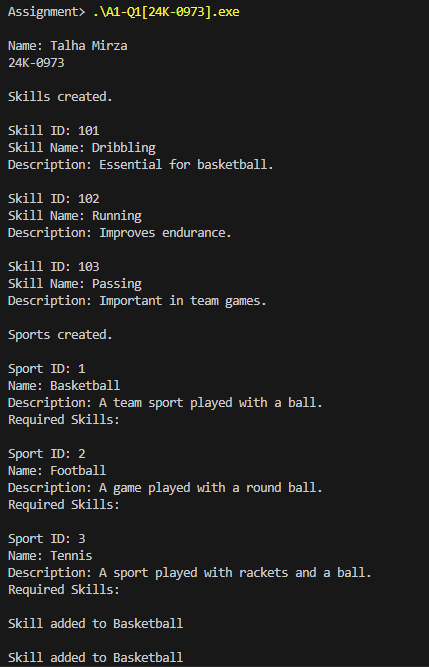
student1.viewMentorDetails();

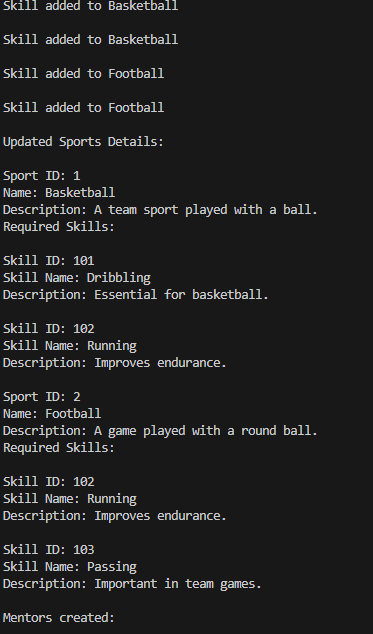
student2.viewMentorDetails();

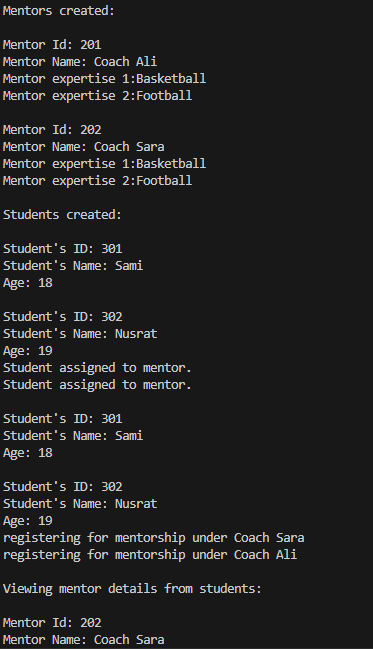
return 0;

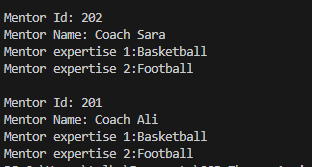
}

## Screenshots:









# Task 2

## Code:

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

class Ball {

private:

int x\_coordinate;

int y\_coordinate;

public:

Ball() {

x\_coordinate = 0;

y\_coordinate = 0;

}

int GetX()

{

return x\_coordinate;

}

int GetY()

{

return y\_coordinate;

}

int& GetXbyref()

{

return x\_coordinate;

}

int& GetYbyref()

{

return y\_coordinate;

}

void SetX(int x)

{

x\_coordinate = x;

}

void SetY(int y)

{

y\_coordinate = y;

}

void displayPosition()

{

cout << "Ball Position: (" << x\_coordinate << ", " << y\_coordinate << ")" << endl;

}

void move(int dx, int dy)

{

int newX = x\_coordinate + dx;

int newY = y\_coordinate + dy;

if(newX < 0 || newX > 3 || newY < 0 || newY > 3)

{

cout << "Ball cannot move outside the field boundaries!" << endl;

}

else

{

x\_coordinate = newX;

y\_coordinate = newY;

}

}

void getPosition()

{

cout << "The position of the ball: (" << x\_coordinate << ", " << y\_coordinate << ")" << endl;

}

};

class Robot

{

private:

string name;

int hits;

public:

Robot()

{

name = "unknown";

hits = 0;

}

Robot(string n, int h)

{

name = n;

hits = h;

}

void hitBall(int &ballX, int &ballY, const string &direction)

{

string d = (direction);

cout << "Robot " << name << " is hitting the ball " << d << "!" << endl;

if (d == "up") {

if (ballY == 3)

{

cout << "Cannot move up. Reached end of field." << endl;;

}

else

{

cout << "Moving Up!" << endl;

ballY++;

hits++;

}

}

else if (d == "down")

{

if (ballY == 0)

{

cout << "Cannot move down. Reached end of field." << endl;

}

else

{

cout << "Moving Down!" << endl;

ballY--;

hits++;

}

}

else if (d == "left")

{

if (ballX == 0)

{

cout << "Cannot move left. Reached end of field." << endl;

}

else

{

cout << "Moving Left!" << endl;

ballX--;

hits++;

}

}

else if (d == "right")

{

if (ballX == 3)

{

cout << "Cannot move right. Reached end of field." << endl;

}

else

{

cout << "Moving Right!" << endl;

ballX++;

hits++;

}

}

else

{

cout << "Invalid direction!" << endl;

}

}

int gethits()

{

return hits;

}

};

class Team

{

private:

string Name;

Robot \*r;

public:

Team(string n, string robotName)

{

Name = n;

r = new Robot(robotName, 0);

}

string getname()

{

return Name;

}

~Team()

{

delete r;

}

Robot\* getRobot()

{

return r;

}

};

class Goal

{

private:

int x, y;

public:

Goal()

{

x = 3;

y = 3;

}

bool isGoalReached(int ballX, int ballY)

{

return ballX == x && ballY == y;

}

};

class Game

{

private:

Team\* teamOne;

Team\* teamTwo;

Ball\* ball;

Goal\* goal;

public:

Game(Team\* t1, Team\* t2, Ball\* b, Goal\* g)

{

teamOne = t1;

teamTwo = t2;

ball = b;

goal = g;

}

void play(Team\* team)

{

while (!goal->isGoalReached(ball->GetX(), ball->GetY()))

{

string direction;

cout << "Enter direction (up, down, left, right) for " << team->getname() << ": ";

cin >> direction;

int& ballX = ball->GetXbyref();

int& ballY = ball->GetYbyref();

team->getRobot()->hitBall(ballX, ballY, direction);

ball->displayPosition();

}

cout << team->getname() << " reached the goal in " << team->getRobot()->gethits() << " hits." << endl;

ball->SetX(0);

ball->SetY(0);

}

void declareWinner()

{

int hits1 = teamOne->getRobot()->gethits();

int hits2 = teamTwo->getRobot()->gethits();

if (hits1 < hits2) {

cout << teamOne->getname() << " wins!" << endl;

} else if (hits1 > hits2) {

cout << teamTwo->getname() << " wins!" << endl;

} else {

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

}

}

void startGame()

{

play(teamOne);

play(teamTwo);

declareWinner();

}

};

int main()

{

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

cout << "24K-0973\n" << endl;

cout << "\n----- GAME BEGINS -----" << endl;

Team t1("Team Fast", "Ronaldo");

Team t2("Team IBA", "Shaheen Afridi");

Ball ball;

Goal goal;

Game game(&t1, &t2, &ball, &goal);

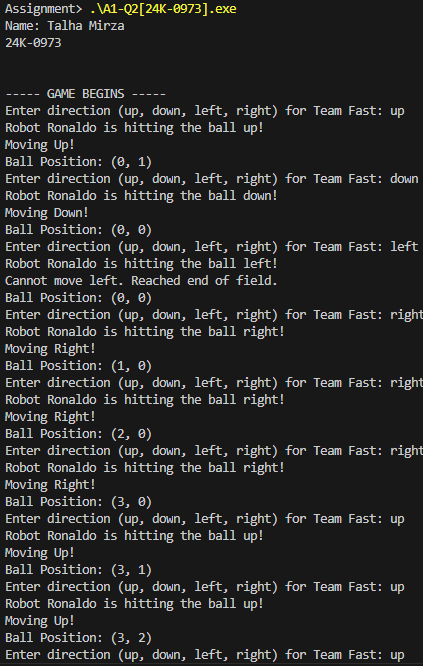
game.startGame();

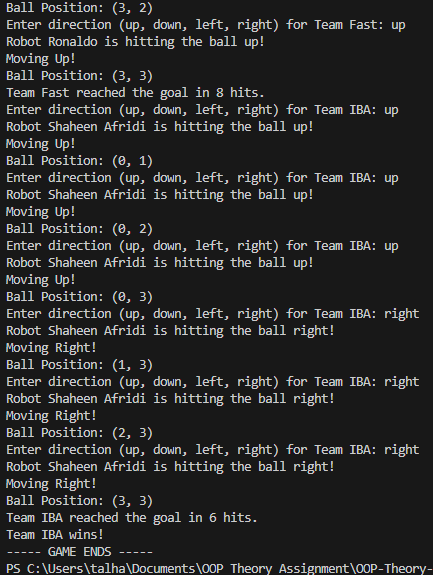
cout << "----- GAME ENDS -----" << endl;

return 0;

}

## Screenshots:





# Task 3

## Code:

# include <iostream>

# include <string>

using namespace std;

class Users

{

string name;

int age;

string licenceType;

int contact;

int id;

static int counter;

public:

Users(string n, int a, string l, int c) : name(n), age(a), licenceType(l), contact(c), id(counter++){}

Users(){}

~Users(){}

void updateDetails()

{

cout << "Enter the updated age: " << endl;

cin >> age;

cin.ignore();

cout << "Enter the updated licence type: " << endl;

getline(cin, licenceType);

cout << "Enter your updated contact number: " << endl;

cin >> contact;

};

string getLicenceType()

{

return licenceType;

}

void display()

{

cout << "---------- User Details ----------"<< endl;

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

cout << "Name: " << name << endl;

cout << "Age: " << age << endl;

cout << "Licence Type: " << licenceType << endl;

cout << "Contact Number: " << contact << endl;

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

}

};

class Vehicle

{

int id;

static int counter;

string model;

float rentalPerDay;

int learner; // 0: not eligible, 1: eligible

int intermediate; // 0: not eligible, 1: eligible

int full; // 0: not eligible, 1: eligible

int available; // 0: not available, 1: available

public:

Vehicle(string m, float r, int l, int i, int f, int a) : id(counter++), model(m), rentalPerDay(r), learner(l), intermediate(i), full(f), available(a){}

Vehicle(){}

~Vehicle(){}

void display()

{

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

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

cout << "Vehicle Model: " << model << endl;

cout << "Rental per Day is: " << rentalPerDay << " rupees" << endl;

if (learner == 1)

{

cout << "This Vehicle can be hired by drivers with a licence type of 'learner'!" << endl;

}

else

{

cout << "This Vehicle cannot be hired by drivers with a licence type of 'learner'!" << endl;

}

if (intermediate == 1)

{

cout << "This Vehicle can be hired by drivers with a licence type of 'intermediate'!" << endl;

}

else

{

cout << "This Vehicle cannot be hired by drivers with a licence type of 'intermediate'!" << endl;

}

if (full == 1)

{

cout << "This Vehicle can be hired by drivers with a licence type of 'full'!" << endl;

}

else

{

cout << "This Vehicle cannot be hired by drivers with a licence type of 'full'!" << endl;

}

}

int getLearner()

{

return learner;

}

int getIntermediate()

{

return intermediate;

}

int getFull()

{

return full;

}

int getAvailable()

{

return available;

}

void setAvailable(int i)

{

available = i;

}

int getID()

{

return id;

}

};

class RentalApplication

{

string name;

int max;

static int counter;

Vehicle \*registeredVehicles;

Vehicle \*v;

Users \*u;

public:

RentalApplication(string name) : name(name)

{

cout << "Enter the maximum number of vehicles which can be registered: " << endl;

cin >> max;

registeredVehicles = new Vehicle[max];

cin.ignore();

}

~RentalApplication()

{

delete[] registeredVehicles;

}

void AddVehicle(Vehicle &v)

{

if (counter < max)

{

registeredVehicles[counter] = v;

cout << "Vehicle has been added!" << endl;

counter++;

}

else

{

cout << "The maximum capacity has been reached. Cannot register more vehicles." << endl;

}

}

void rentAcar(Users &u)

{

int eligibleVehicles[max];

int c = 0;

string l = u.getLicenceType();

if (l == "learner")

{

for (int i = 0; i < counter; i++)

{

if ((registeredVehicles[i].getLearner() == 1) && (registeredVehicles[i].getAvailable() == 1))

{

eligibleVehicles[c] = i;

cout << "Option " << c+1 << endl;

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

registeredVehicles[i].display();

c++;

}

}

}

else if (l == "intermediate")

{

for (int i = 0; i < counter; i++)

{

if ((registeredVehicles[i].getIntermediate() == 1) && (registeredVehicles[i].getAvailable() == 1))

{

eligibleVehicles[c] = i;

cout << "Option " << c+1 << endl;

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

registeredVehicles[i].display();

c++;

}

}

}

else if (l == "full")

{

for (int i = 0; i < counter; i++)

{

if ((registeredVehicles[i].getFull() == 1) && (registeredVehicles[i].getAvailable() == 1))

{

eligibleVehicles[c] = i;

cout << "Option " << c+1 << endl;

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

registeredVehicles[i].display();

c++;

}

}

}

else

{

cout << "Invalid Licence Type!" << endl;

}

if (c == 0)

{

cout << "We're Sorry, right now you're not eligible for any vehicle." << endl;

}

else

{

int choice;

cout << "Select a car number from " << 1 << " to " << c << endl;

cin >> choice;

while (choice < 1 || choice > c)

{

cout << "Enter a valid number: " << endl;

cin >> choice;

}

registeredVehicles[eligibleVehicles[choice-1]].setAvailable(0); // Set to unavailable

cout << "You have rented the Car!" << endl;

}

}

void removeVehicle()

{

if (counter > 0)

{

int ID;

cout << "Enter the ID of the vehicle you want to remove: " << endl;

cin >> ID;

int found = 0;

int i = 0;

while ((found == 0) && (i < counter))

{

if (registeredVehicles[i].getID() == ID)

{

found = 1;

break;

}

i++;

}

if (found == 1)

{

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

{

registeredVehicles[j] = registeredVehicles[j+1];

}

counter--;

cout << "Vehicle Removed!" << endl;

}

else

{

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

}

}

else

{

cout << "No Vehicles Registered." << endl;

}

}

void printDetails()

{

cout << "-------- Cars Available & their details ----------------" << endl;

for (int i = 0; i < counter; i++)

{

registeredVehicles[i].display();

}

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

}

};

int Users :: counter = 0;

int Vehicle :: counter = 0;

int RentalApplication :: counter = 0;

int main()

{

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

cout << "24K-0973\n" << endl;

cout << "\n\*\*\*\*\*Rental A Car Application\*\*\*\*\*" << endl;

RentalApplication app("MyRentalApp");

Vehicle v1("Toyota Corolla", 1000.0, 1, 1, 1, 1);

Vehicle v2("Honda Civic", 1200.0, 0, 1, 1, 1);

Vehicle v3("Suzuki Alto", 800.0, 1, 0, 0, 1);

app.AddVehicle(v1);

app.AddVehicle(v2);

app.AddVehicle(v3);

cout << "\n----- All Available Vehicles -----" << endl;

app.printDetails();

Users user1("Talha", 25, "full", 1234567890);

cout << "\n----- Rent A Car -----" << endl;

cout << "\n----- Vehicles Eligible for you -----" << endl;

app.rentAcar(user1);

cout << "\n----- Remove Vehicle -----" << endl;

app.removeVehicle();

cout << "\n----- Update User Details -----" << endl;

user1.updateDetails();

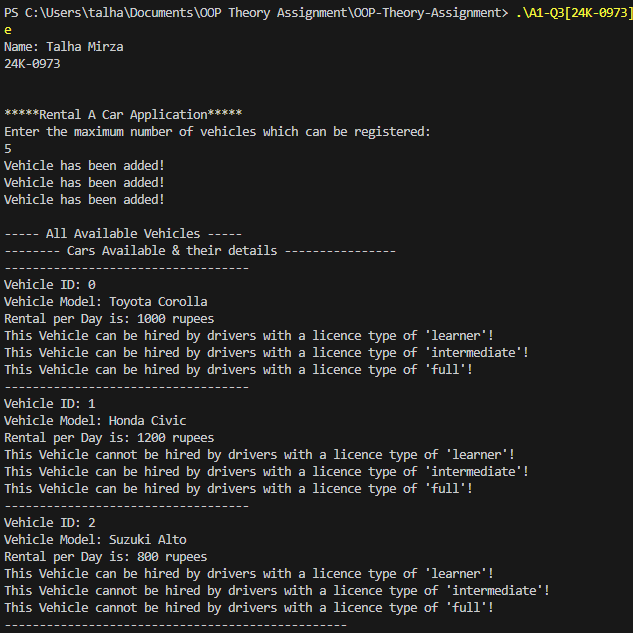
user1.display();

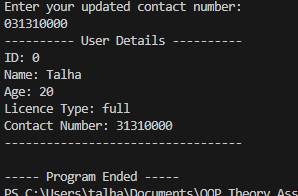
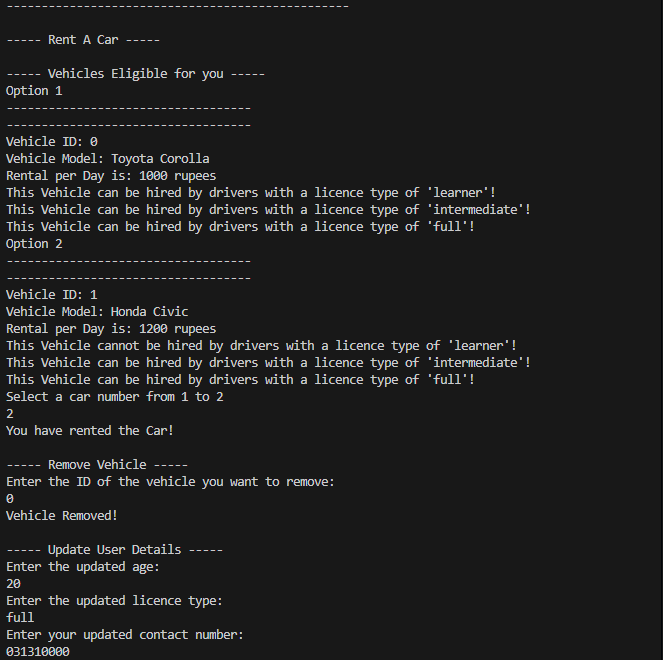
cout << "\n----- Program Ended -----" << endl;

return 0;

}

## Screenshots:





# Task 4

## Code:

# include <iostream>

# include <string>

using namespace std;

class Student

{

string name;

string ID;

int fees\_paid; // 0: Not Paid, 1: Paid

string stop;

int point;

int \*Attendance;

public:

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

{

point = -1;

cout << "Enter the days of semester: ";

int days;

cin >> days;

Attendance = new int[days];

for (int i = 0; i < days; i++)

{

Attendance[i] = 0;

}

}

void markAttendance(int i) // i is the day of the semester

{

Attendance[i-1] = 1;

cout << "Attendance marked!" << endl;

}

Student(){}

~Student(){}

int getFeesPaid()

{

return fees\_paid;

}

void setFeesPaid(int f)

{

fees\_paid = f;

}

void setPoint(int p)

{

point = p;

}

int getPoint()

{

return point;

}

string getID()

{

return ID;

}

string getStop()

{

return stop;

}

void viewDetails()

{

cout << "----- Student Details -----" << endl;

cout << "Name: " << name << endl;

cout << "FAST-NU ID: " << ID << endl;

if (fees\_paid == 0)

{

cout << "The fees is not Paid!" << endl;

}

else

{

cout << "The fees is Paid!" << endl;

}

cout << "The stop of the student is: " << stop << endl;

}

};

class Bus

{

string name;

static int counter;

int current;

int id;

int max;

int max\_students;

int current\_students;

Student\* studentsRegistered;

string\* busStops;

Student \*s;

public:

Bus(string name) : name(name)

{

current = 0;

current\_students = 0;

id = counter++;

cout << "Enter the maximum number of stops of the bus: " << endl;

cin >> max;

busStops = new string[max];

cout << "Enter the maximum number of students which can be registered: " << endl;

cin >> max\_students;

studentsRegistered = new Student[max\_students];

}

Bus(){}

~Bus()

{

delete[] busStops;

delete[] studentsRegistered;

}

int getMax()

{

return max;

}

void AddStop(string bus\_stop)

{

if (current < max)

{

busStops[current] = bus\_stop;

cout << "Bus Stop has been Added!" << endl;

current++;

}

else

{

cout << "Max Limit Reached. Cannot Add More Bus Stops." << endl;

}

}

int searchStop(Student &s)

{

int found = 0;

int i = 0;

while ((found == 0) && (i < current))

{

if (s.getStop() == busStops[i])

{

return 1;

}

i++;

}

return 0;

}

void AddStudent(Student &s)

{

if (current\_students < max\_students)

{

if (s.getFeesPaid() == 1)

{

if (searchStop(s) == 1)

{

studentsRegistered[current\_students] = s;

s.setPoint(id);

cout << "Student has been Registered!" << endl;

current\_students++;

}

else

{

cout << "This Bus does not offers the desired Bus Stop." << endl;

}

}

else

{

cout << "Fees has not been Paid!" << endl;

}

}

else

{

cout << "Max Limit Reached. No more students can be registered!" << endl;

}

}

void removeStudent(string ID)

{

if (counter > 0)

{

int found = 0;

int i = 0;

while ((found == 0) && (i < current\_students))

{

if (studentsRegistered[i].getID() == ID)

{

found = 1;

break;

}

i++;

}

if (found == 1)

{

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

{

studentsRegistered[j] = studentsRegistered[j+1];

}

current\_students--;

cout << "Student Removed!" << endl;

}

else

{

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

}

}

else

{

cout << "No Students Registered." << endl;

}

}

void initializeFeesPaid()

{

for (int i = 0; i < current\_students; i++)

{

studentsRegistered[i].setFeesPaid(0);

}

}

void viewDetails()

{

cout << "----- Bus Details -----" << endl;

cout << "Route Number: " << id << endl;

cout << "\nThe stops of the bus: " << endl;

for (int i = 0; i < counter; i++)

{

cout << i+1 << ". " << busStops[i] << endl;

}

// cout << "\nStudents Registered: " << endl;

// for (int i = 0; i < current\_students; i++)

// {

// studentsRegistered[i].viewDetails();

// }

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

}

};

class PointSystem

{

string name;

static int counter;

int max;

Student\* s;

Bus\* b;

Bus \*busSystem;

public:

PointSystem(string name) : name(name)

{

cout << "Enter the maximum number of buses which can be registered in the system: " << endl;

cin >> max;

busSystem = new Bus[max];

}

~PointSystem()

{

delete[] busSystem;

}

void registerBus(Bus &b)

{

if (counter < max)

{

busSystem[counter] = b;

counter++;

cout << "Bus has been registered!" << endl;

}

else

{

cout << "Maximum capacity has been reached. No more buses can be registered!" << endl;

}

}

void viewDetails()

{

cout << "----- All Point Services Details -----" << endl;

for (int i = 0; i < counter; i++)

{

busSystem[i].viewDetails();

}

}

void initializeFees()

{

for (int i = 0; i < counter; i++)

{

busSystem[i].initializeFeesPaid(); // new semester

}

}

void AddStudent(Bus &b, Student &s)

{

b.AddStudent(s);

}

void removeStudent(Bus &b, Student &s)

{

b.removeStudent(s.getID());

}

};

int Bus :: counter = 0;

int PointSystem :: counter = 0;

int main()

{

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

cout << "24K-0973\n" << endl;

cout << "----- Point System -----" << endl;

cout << "\nBus 1: " << endl;

Bus bus1("Jadoon");

bus1.AddStop("A");

bus1.AddStop("B");

bus1.AddStop("C");

cout << "\nBus 2: " << endl;

Bus bus2("Nadeem");

bus2.AddStop("D");

bus2.AddStop("E");

bus2.AddStop("F");

PointSystem ps("University Transport");

ps.registerBus(bus1);

ps.registerBus(bus2);

Student student1("Talha", "k240973", 1, "A");

Student student2("Ayesha", "k240900", 1, "B");

Student student3("Safi", "k219870", 1, "C");

Student student4("Khanum", "k240840", 1, "D");

Student student5("Khan", "k240900", 1, "E");

Student student6("Shahid", "k219860", 0, "F");

ps.AddStudent(bus1, student1);

ps.AddStudent(bus1, student2);

ps.AddStudent(bus1, student3);

ps.AddStudent(bus2, student4);

ps.AddStudent(bus2, student5);

ps.AddStudent(bus2, student6);

student1.markAttendance(1);

student4.markAttendance(1);

student5.markAttendance(1);

bus1.removeStudent("k240973");

ps.viewDetails();

ps.initializeFees();

return 0;

}

## Screenshots:

