 Computer Science and Creative Technologies

**Tasks**

**Task 1 :**

1. Create a Time class to store hours and minutes. Implement:

Overload the + operator to add two Time objects

Overload the > operator to compare two Time objects

Handle invalid time (>24 hours or >60 minutes) by throwing a custom exception

Answer:

#include <iostream>

class Time {

private:

int hours;

int minutes;

public:

Time(int h = 0, int m = 0) {

if (h < 0 || h > 24 || m < 0 || m >= 60) {

std::cout << "Invalid time!" << std::endl;

hours = 0;

minutes = 0;

} else {

hours = h;

minutes = m;

}

}

Time operator+(const Time& other) {

int totalMinutes = minutes + other.minutes;

int extraHours = totalMinutes / 60;

int remainingMinutes = totalMinutes % 60;

int totalHours = hours + other.hours + extraHours;

if (totalHours > 24) {

std::cout << "Time exceeds 24 hours!" << std::endl;

return Time(0, 0);

}

return Time(totalHours, remainingMinutes);

}

bool operator>(const Time& other) {

return (hours \* 60 + minutes) > (other.hours \* 60 + other.minutes);

}

void print() {

if (hours < 10) std::cout << "0";

std::cout << hours << ":";

if (minutes < 10) std::cout << "0";

std::cout << minutes;

}

};

int main() {

Time t1(10, 30);

Time t2(2, 45);

Time t3 = t1 + t2;

std::cout << "Time 1: ";

t1.print();

std::cout << "\nTime 2: ";

t2.print();

std::cout << "\nSum: ";

t3.print();

std::cout << "\nIs t1 > t2? " << (t1 > t2 ? "Yes" : "No") << std::endl;

return 0;

}



**Task 2: 70 marks**

1. Create a base class Vehicle and two derived classes Car and Bike:

Vehicle has registration number and color

Car adds number of seats

Bike adds engine capacity

Each class should have its own method to write its details to a file

Include proper inheritance and method overriding

Answer :

#include <iostream>

#include <fstream>

#include <string>

class Vehicle {

protected:

std::string regNumber;

std::string color;

public:

Vehicle(std::string reg = "UNKNOWN", std::string col = "UNKNOWN")

: regNumber(reg), color(col) {}

virtual void writeToFile(const std::string& filename) {

std::ofstream outFile(filename, std::ios::app);

if (outFile) {

outFile << "Vehicle Details:\n"

<< "Registration: " << regNumber << "\n"

<< "Color: " << color << "\n";

} else {

std::cout << "Unable to open file: " << filename << std::endl;

}

}

virtual ~Vehicle() {}

};

class Car : public Vehicle {

private:

int numSeats;

public:

Car(std::string reg = "UNKNOWN", std::string col = "UNKNOWN", int seats = 0)

: Vehicle(reg, col), numSeats(seats) {}

void writeToFile(const std::string& filename) override {

std::ofstream outFile(filename, std::ios::app);

if (outFile) {

outFile << "Car Details:\n"

<< "Registration: " << regNumber << "\n"

<< "Color: " << color << "\n"

<< "Number of Seats: " << numSeats << "\n";

} else {

std::cout << "Unable to open file: " << filename << std::endl;

}

}

};

class Bike : public Vehicle {

private:

int engineCapacity;

public:

Bike(std::string reg = "UNKNOWN", std::string col = "UNKNOWN", int capacity = 0)

: Vehicle(reg, col), engineCapacity(capacity) {}

void writeToFile(const std::string& filename) override {

std::ofstream outFile(filename, std::ios::app);

if (outFile) {

outFile << "Bike Details:\n"

<< "Registration: " << regNumber << "\n"

<< "Color: " << color << "\n"

<< "Engine Capacity: " << engineCapacity << "cc\n";

} else {

std::cout << "Unable to open file: " << filename << std::endl;

}

}

};

int main() {

Vehicle v("ABC123", "Blue");

Car c("XYZ789", "Red", 5);

Bike b("DEF456", "Black", 150);

v.writeToFile("vehicles.txt");

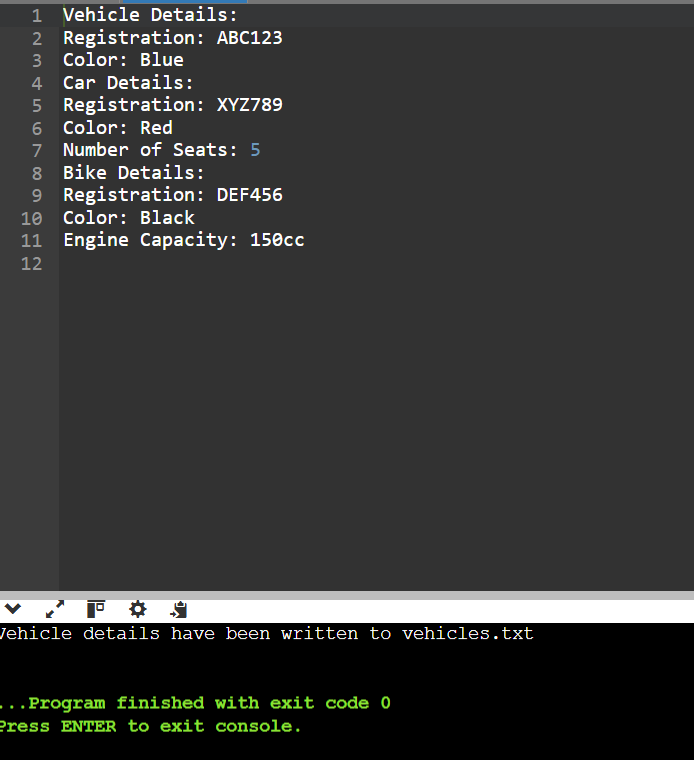
c.writeToFile("vehicles.txt");

b.writeToFile("vehicles.txt");

std::cout << "Vehicle details have been written to vehicles.txt" << std::endl;

return 0;

}



1. Create a program that:

Reads student records (roll, name, marks) from a text file

Throws an exception if marks are not between 0 and 100

Allows adding new records with proper validation

Saves modified records back to file

Answer:

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <stdexcept>

class Student {

public:

    int roll;

    std::string name;

    float marks;

    Student(int r, const std::string& n, float m) : roll(r), name(n), marks(m) {}

};

class StudentRecords {

private:

    std::vector<Student> students;

    const std::string filename;

public:

    StudentRecords(const std::string& file) : filename(file) {

        loadRecords();

    }

    void loadRecords() {

        std::ifstream inFile(filename);

        if (!inFile) {

            std::cerr << "Unable to open file: " << filename << std::endl;

            return;

        }

        int roll;

        std::string name;

        float marks;

        while (inFile >> roll >> name >> marks) {

            if (marks < 0 || marks > 100) {

                throw std::invalid\_argument("Marks must be between 0 and 100.");

            }

            students.emplace\_back(roll, name, marks);

        }

        inFile.close();

    }

    void addRecord(int roll, const std::string& name, float marks) {

        if (marks < 0 || marks > 100) {

            throw std::invalid\_argument("Marks must be between 0 and 100.");

        }

        students.emplace\_back(roll, name, marks);

    }

    void saveRecords() {

        std::ofstream outFile(filename);

        if (!outFile) {

            std::cerr << "Unable to open file: " << filename << std::endl;

            return;

        }

        for (const auto& student : students) {

            outFile << student.roll << " " << student.name << " " << student.marks << "\n";

        }

        outFile.close();

    }

    void displayRecords() const {

        for (const auto& student : students) {

            std::cout << "Roll: " << student.roll << ", Name: " << student.name << ", Marks: " << student.marks << "\n";

        }

    }

};

int main() {

    const std::string filename = "students.txt";

    StudentRecords records(filename);

    try {

        records.displayRecords();

        // Adding a new record

        int roll;

        std::string name;

        float marks;

        std::cout << "Enter new student record (roll name marks): ";

        std::cin >> roll >> name >> marks;

        records.addRecord(roll, name, marks);

        records.saveRecords();

        std::cout << "New record added successfully!\n";

        records.displayRecords();

    } catch (const std::invalid\_argument& e) {

        std::cerr << "Error: " << e.what() << std::endl;

    }

    return 0;

}

