****

**worksheet 3**

****

SUBMITTED BY: Sadaf akhtar ansari

student id: 24030177

**A logo with blue and red text

AI-generated content may be incorrect.A red sign with white text

AI-generated content may be incorrect.**

# **QUESTION 1.1**

1. **Create a Time class to store hours and minutes. Implement:**
   1. **Overload the + operator to add two Time objects**
   2. **Overload the > operator to compare two Time objects**
   3. **Handle invalid time (>24 hours or >60 minutes) by throwing a custom exception**

**CODE:**

**#include <iostream>**

**#include <stdexcept>**

**using namespace std;**

**class InvalidTimeException : public exception { // Custom exception class**

**public:**

**const char\* what() const throw() {**

**return "Invalid time! Hours must be <= 24 and minutes < 60.";**

**}**

**};**

**// Time class**

**class Time {**

**private:**

**int hours, minutes;**

**void checkValid() {**

**if (hours > 24 || minutes >= 60) {**

**throw InvalidTimeException();**

**}**

**}**

**public:**

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

**hours = h;**

**minutes = m;**

**checkValid();**

**}**

**// Add two Time objects**

**Time operator+(const Time& t) {**

**int newHours = hours + t.hours;**

**int newMinutes = minutes + t.minutes;**

**if (newMinutes >= 60) {**

**newHours += newMinutes / 60;**

**newMinutes %= 60;**

**}**

**return Time(newHours, newMinutes);**

**}**

**// Compare two Time objects**

**bool operator>(const Time& t) {**

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

**}**

**void display() {**

**cout << hours << " hours " << minutes << " minutes" << endl;**

**}**

**};**

**int main() {**

**try {**

**int h1, m1, h2, m2;**

**cout << "Enter first time (hours minutes): ";**

**cin >> h1 >> m1;**

**Time t1(h1, m1);**

**cout << "Enter second time (hours minutes): ";**

**cin >> h2 >> m2;**

**Time t2(h2, m2);**

**// Add times**

**Time sum = t1 + t2;**

**cout << "Sum: ";**

**sum.display();**

**// Compare times**

**if (t1 > t2)**

**cout << "First time is greater." << endl;**

**else**

**cout << "Second time is greater or equal." << endl;**

**} catch (const exception& e) {**

**cout << "Error: " << e.what() << endl;**

**}**

**return 0;**

**}**

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

# **QUESTION 2.1**

1. **Create a base class Vehicle and two derived classes Car and Bike:**
   1. **Vehicle has registration number and color**
   2. **Car adds number of seats**
   3. **Bike adds engine capacity**
   4. **Each class should have its own method to write its details to a file**
   5. **Include proper inheritance and method overriding.**

**CODE:**

**#include <iostream>**

**#include <fstream>**

**using namespace std;**

**// Base class: Vehicle (stores common info)**

**class Vehicle {**

**protected:**

**string registrationNumber;**

**string color;**

**public:**

**Vehicle(string regNum, string clr) : registrationNumber(regNum), color(clr) {}**

**// Virtual function to save vehicle details in a file**

**virtual void writeToFile(ofstream& file) const {**

**file << "Vehicle - Reg No: " << registrationNumber << ", Color: " << color << endl;**

**}**

**// Virtual function to show vehicle details**

**virtual void display() const {**

**cout << "Vehicle - Reg No: " << registrationNumber << ", Color: " << color << endl;**

**}**

**};**

**// Derived class: Car (adds seats info)**

**class Car : public Vehicle {**

**private:**

**int numberOfSeats;**

**public:**

**Car(string regNum, string clr, int seats) : Vehicle(regNum, clr), numberOfSeats(seats) {}**

**// Override: Save car details**

**void writeToFile(ofstream& file) const override {**

**file << "Car - Reg No: " << registrationNumber << ", Color: " << color << ", Seats: " << numberOfSeats << endl;**

**}**

**// Override: Show car details**

**void display() const override {**

**cout << "Car - Reg No: " << registrationNumber << ", Color: " << color << ", Seats: " << numberOfSeats << endl;**

**}**

**};**

**// Derived class: Bike (adds engine capacity info)**

**class Bike : public Vehicle {**

**private:**

**int engineCapacity;**

**public:**

**Bike(string regNum, string clr, int engineCap) : Vehicle(regNum, clr), engineCapacity(engineCap) {}**

**// Override: Save bike details**

**void writeToFile(ofstream& file) const override {**

**file << "Bike - Reg No: " << registrationNumber << ", Color: " << color << ", Engine: " << engineCapacity << "cc" << endl;**

**}**

**// Override: Show bike details**

**void display() const override {**

**cout << "Bike - Reg No: " << registrationNumber << ", Color: " << color << ", Engine: " << engineCapacity << "cc" << endl;**

**}**

**};**

**int main() {**

**string regNum, color;**

**int choice, seats, engineCap;**

**// Open file to store vehicle details**

**ofstream file("vehicles.txt", ios::app); // app mode means add at the end**

**if (!file) {**

**cout << "File couldn't be opened!" << endl;**

**return 1;**

**}**

**// User chooses the type of vehicle**

**cout << "Choose Vehicle Type:\n1. Car\n2. Bike\nEnter choice: ";**

**cin >> choice;**

**cin.ignore(); // Clear buffer for getline**

**// Get common input**

**cout << "Enter Registration Number: ";**

**getline(cin, regNum);**

**cout << "Enter Color: ";**

**getline(cin, color);**

**// Handle Car**

**if (choice == 1) {**

**cout << "Enter Number of Seats: ";**

**cin >> seats;**

**Car car(regNum, color, seats);**

**car.writeToFile(file);**

**cout << "Car details saved successfully!\n";**

**car.display();**

**}**

**// Handle Bike**

**else if (choice == 2) {**

**cout << "Enter Engine Capacity (in CC): ";**

**cin >> engineCap;**

**Bike bike(regNum, color, engineCap);**

**bike.writeToFile(file);**

**cout << "Bike details saved successfully!\n";**

**bike.display();**

**}**

**// Invalid choice**

**else {**

**cout << "Invalid choice entered!" << endl;**

**}**

**file.close(); // Close the file**

**return 0;**

**}**

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

# **QUESTION 2.2**

1. **Create a program that:**
   1. **Reads student records (roll, name, marks) from a text file**
   2. **Throws an exception if marks are not between 0 and 100**
   3. **Allows adding new records with proper validation**
   4. **Saves modified records back to file**

**CODE:**

**#include <iostream>**

**#include <fstream>**

**#include <vector>**

**#include <sstream>**

**#include <stdexcept>**

**using namespace std;**

**// Custom exception class to handle invalid marks**

**class InvalidMarksException : public exception {**

**public:**

**const char\* what() const throw() {**

**return "Marks should be between 0 and 100!";**

**}**

**};**

**// Structure to hold student details**

**struct Student {**

**int roll;**

**string name;**

**int marks;**

**};**

**// Function to read students from file**

**vector<Student> readRecords(const string& filename) {**

**vector<Student> students;**

**ifstream file(filename);**

**// If file doesn't exist, create a new empty file**

**if (!file) {**

**cout << "File not found. Creating a new file..." << endl;**

**ofstream newFile(filename);**

**newFile.close();**

**return students;**

**}**

**string line;**

**while (getline(file, line)) {**

**stringstream ss(line);**

**Student s;**

**ss >> s.roll;**

**ss.ignore(); // Skip space**

**getline(ss, s.name, ',');**

**ss >> s.marks;**

**// Check if marks are valid**

**if (s.marks < 0 || s.marks > 100) {**

**throw InvalidMarksException();**

**}**

**students.push\_back(s);**

**}**

**file.close();**

**return students;**

**}**

**// Function to add a new student**

**void addRecord(vector<Student>& students) {**

**Student s;**

**cout << "Enter roll number: ";**

**cin >> s.roll;**

**cin.ignore(); // Ignore leftover newline**

**cout << "Enter name: ";**

**getline(cin, s.name);**

**cout << "Enter marks (0-100): ";**

**cin >> s.marks;**

**// Check if marks are within valid range**

**if (s.marks < 0 || s.marks > 100) {**

**throw InvalidMarksException();**

**}**

**students.push\_back(s);**

**}**

**// Function to save all students back to file**

**void saveRecords(const string& filename, const vector<Student>& students) {**

**ofstream file(filename);**

**if (!file) {**

**cout << "Error opening file for saving!" << endl;**

**return;**

**}**

**for (const Student& s : students) {**

**file << s.roll << " " << s.name << "," << s.marks << endl;**

**}**

**file.close();**

**cout << "Records saved!" << endl;**

**}**

**int main() {**

**string filename = "students.txt";**

**vector<Student> students;**

**try {**

**students = readRecords(filename);**

**} catch (const exception& e) {**

**cout << "Problem reading file: " << e.what() << endl;**

**return 1;**

**}**

**// Show existing student records**

**cout << "Student Records:" << endl;**

**for (const Student& s : students) {**

**cout << s.roll << " " << s.name << " " << s.marks << endl;**

**}**

**char choice;**

**cout << "Do you want to add a new student? (y/n): ";**

**cin >> choice;**

**if (choice == 'y' || choice == 'Y') {**

**try {**

**addRecord(students);**

**saveRecords(filename, students);**

**} catch (const exception& e) {**

**cout << "Error: " << e.what() << endl;**

**}**

**}**

**return 0;**

**}**

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**