 Computer Science and Creative Technologies

**Coursework or Assessment Specification**

## Module Details

|  |  |
| --- | --- |
| **Module Code** | UFCFGL-30-1 |
| **Module Title** | Programming in C++ |
| **Module Leader** | Benedict Gaster |
| **Module Tutors** | Aashish Acharya |
| **Year** | 2024-2025 |
| **Component/Element Number** | Coursework |
| **Weighting** | 10% |
| **Element Description** |  |

## Dates

|  |  |
| --- | --- |
| **Date issued to students** | 10/02/2025 |
| **Submission Date** | 10/04/2025 |
| **Submission Place** | Backboard |
| **Submission Time** | 00:00 |
| **Submission Notes** | Submit Gitlab URL |

## Feedback

|  |  |
| --- | --- |
| **Feedback provision will be** | Via Blackboard |

**Coursework-3: OPP in C++**

This coursework covers the topics we have studied in the week5 and week6. So you are required to go through the learning materials of week5 and week6 to complete this coursework.

As you know that the basics of Object oriented programming concepts have been covered in these weeks, the questions in this assignment are based on those concepts.

The marking scheme for this session is as follows:

* Task 1: 30 marks
* Task 2: 70 marks
* Task 3: 0 marks

**Submission Details in the VLE:**

Submit your solution of worksheet-6[named as “yourid\_task\_6.docx”] along with a text file containing URL of your GIT REPO zipped in file named “YourID.zip” in the link provided on the VLE of UWE.

**NOTE:** Simply submitting your solution will not ensure any marks. You have to attend the demo in the specified date and time [**which will announced later**] and you should be able to explain or answer any questions asked during the demo to get marks.

Necessary steps to complete this worksheet.

* Make sure you have gone through all the contents of lecture slides, tutorial files, Lab exercises and the handout notes of week-11 and week-12 folders.
* Make sure you have already created your repo in GIT to upload your solution documents.

**Tasks**

**Task 1 : 30 marks**

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

Answer:

#include <iostream>

#include <stdexcept>

using namespace std;

// Way Exception Class

class InvalidTimeException : public exception {

public:

const char\* what() const noexcept override {

return "Invalid Time: Hours must be <= 24 and Minutes must be <= 60";

}

};

class Time {

private:

int hours;

int minutes;

void validateTime() {

if (hours > 24 || minutes >= 60 || hours < 0 || minutes < 0) {

throw InvalidTimeException();

}

}

public:

// Constructor

Time(int h = 0, int m = 0) : hours(h), minutes(m) {

validateTime();

}

// Overload + operator

Time operator+(const Time& t) const {

int totalMinutes = minutes + t.minutes;

int totalHours = hours + t.hours + totalMinutes / 60;

totalMinutes %= 60;

if (totalHours > 24) {

throw InvalidTimeException();

}

return Time(totalHours, totalMinutes);

}

// Overload > operator

bool operator>(const Time& t) const {

if (hours > t.hours) return true;

if (hours == t.hours && minutes > t.minutes) return true;

return false;

}

// Display function

void display() const {

cout << hours << "h " << minutes << "m" << endl;

}

};

// --- Main Program ---

int main() {

try {

Time t1(10, 45);

Time t2(12, 30);

cout << "Time 1: ";

t1.display();

cout << "Time 2: ";

t2.display();

Time sum = t1 + t2;

cout << "Sum of Time 1 and Time 2: ";

sum.display();

if (t1 > t2) {

cout << "Time 1 is greater than Time 2\n";

} else {

cout << "Time 2 is greater than or equal to Time 1\n";

}

// Time invalidTime(25, 10); // Will throw exception

} catch (const InvalidTimeException& e) {

cerr << "Exception: " << e.what() << endl;

}

return 0;

}

**Task 2: 70 marks**

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
2. 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

Answer:

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <stdexcept>

using namespace std;

// construct class Vehicle

class Vehicle {

protected:

string registration\_number;

string color;

public:

Vehicle(string reg\_num, string col) : registration\_number(reg\_num), color(col) {}

virtual void writeToFile(const string &filename) {

ofstream file(filename, ios::app);

if (file.is\_open()) {

file << "Vehicle Registration: " << registration\_number

<< ", Color: " << color << endl;

file.close();

}

}

virtual ~Vehicle() {}

};

// Extract class Car

class Car : public Vehicle {

private:

int seats;

public:

Car(string reg\_num, string col, int seat\_count)

: Vehicle(reg\_num, col), seats(seat\_count) {}

void writeToFile(const string &filename) override {

ofstream file(filename, ios::app);

if (file.is\_open()) {

file << "Car Registration: " << registration\_number

<< ", Color: " << color

<< ", Seats: " << seats << endl;

file.close();

}

}

};

// Extract class Bike

class Bike : public Vehicle {

private:

int engine\_capacity;

public:

Bike(string reg\_num, string col, int engine\_cap)

: Vehicle(reg\_num, col), engine\_capacity(engine\_cap) {}

void writeToFile(const string &filename) override {

ofstream file(filename, ios::app);

if (file.is\_open()) {

file << "Bike Registration: " << registration\_number

<< ", Color: " << color

<< ", Engine Capacity: " << engine\_capacity << "cc" << endl;

file.close();

}

}

};

// Student class to store student details

class Student {

public:

string roll;

string name;

int marks;

Student(string r, string n, int m) : roll(r), name(n), marks(m) {}

void display() const {

cout << "Roll: " << roll << ", Name: " << name << ", Marks: " << marks << endl;

}

};

class StudentRecord {

private:

vector<Student> records;

string filename;

void readFromFile() {

ifstream file(filename);

string roll, name;

int marks;

if (file.is\_open()) {

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

records.push\_back(Student(roll, name, marks));

}

file.close();

} else {

cout << "File not found, starting with empty records." << endl;

}

}

void writeToFile() {

ofstream file(filename);

if (file.is\_open()) {

for (const auto &record : records) {

file << record.roll << " " << record.name << " " << record.marks << endl;

}

file.close();

}

}

public:

StudentRecord(string fname) : filename(fname) {

readFromFile();

}

void displayRecords() {

for (const auto &student : records) {

student.display();

}

}

void addStudent(string roll, string name, int marks) {

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

throw invalid\_argument("Marks should be between 0 and 100.");

}

records.push\_back(Student(roll, name, marks));

}

void saveRecords() {

writeToFile();

}

};

int main() {

// Task 1: Vehicle Management (Car and Bike)

cout << "Task 1: Vehicle Management\n";

Car car("ABC123", "Red", 5);

Bike bike("XYZ456", "Black", 150);

car.writeToFile("vehicle\_details.txt");

bike.writeToFile("vehicle\_details.txt");

cout << "Vehicle details have been written to vehicle\_details.txt\n\n";

// Task 2: Student Record Management

cout << "Task 2: Student Record Management\n";

StudentRecord record("student\_records.txt");

// Display existing records

record.displayRecords();

// Add a new student record

try {

string roll, name;

int marks;

cout << "\nEnter Roll Number: ";

cin >> roll;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Marks: ";

cin >> marks;

record.addStudent(roll, name, marks);

cout << "Student record added successfully.\n";

} catch (const invalid\_argument &e) {

cout << e.what() << endl;

}

// Save the new records to file

record.saveRecords();

cout << "Student records have been saved to student\_records.txt\n";

return 0;

}