WORKSHEET 3

24030183

SUBMITTED BY: Sujal Adhikari
STUDENT ID: 24030183

INTRODUCTION

So this is an assessment I need to perform to show my understanding of Object-oriented programming (OOP) in C++ and/ or the use of the topics covered during weeks 5 and 6 of the course. The first was to create a Time class with operator overloading to make addition and comparison work. Additional aspects of programming that are critical such as exception/error handling for invalid input, file I/O for reading and writing student records, and data validation techniques are also integrated in this assignment. In summary, these assignments drive me to connect abstract computer science concepts with practical coding, which helps solidify a crucial tenet of OOP as implemented in C++: ambiguity and uncertainty of what needs to be done on the part of the programmer.

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 Implementation

```
#include <stdexcept> //For exception handling

using namespace std;

class InvalidTimeException : public exception {

public:

  const char* what() const throw() {

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

  }

};

class Time {

private:

  int hours, minutes;


  void validate() {

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

      throw InvalidTimeException();

    }

  }
```

```
}

  Time operator+(const Time& t) {

    int totalMinutes = (hours + t.hours) * 60 + (minutes + t.minutes);

    return Time(totalMinutes / 60, totalMinutes % 60);

  }

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

```
cout << "Second time is greater than or equal to first time." << endl;

} catch (const exception& e) {
cout << "Error: " << e.what() << endl;
}
return 0;
}
```

Output

```
"E:\C++\C-_CourseWorks\Wo

Enter first time (hours minutes): 5 6
Enter second time (hours minutes): 7 8
Sum: 12 hours 14 minutes
Second time is greater than or equal to first time.

Process returned 0 (0x0)   execution time : 8.354 s
Press any key to continue.
```

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 Implementation

```
#include <iostream>
#include <fstream>
using namespace std;
// Base class: Vehicle
class Vehicle {
protected:
   string registrationNumber;
   string color;
public:
   Vehicle(string regNum, string clr) : registrationNumber(regNum), color(clr) {}
   // Virtual function to write details to a file
   virtual void writeToFile(ofstream& file) const {
      file << "Vehicle: " << registrationNumber << ", " << color << endl;
   }
   virtual void display() const {
      cout << "Vehicle - Reg No: " << registrationNumber << ", Color: " << color << endl;
   }
};
// Derived class: Car
class Car : public Vehicle {
private:
   int numberOfSeats;
public:
   Car(string regNum, string clr, int seats) : Vehicle(regNum, clr), numberOfSeats(seats) {}
```

```
void writeToFile(ofstream& file) const override {
    file << "Car: " << registrationNumber << ", " << color << ", Seats: " << numberOfSeats << endl;
  }
  void display() const override {
    cout << "Car - Reg No: " << registrationNumber << ", Color: " << color << ", Seats: " << numberOfSeats
<< endl;
  }
};
// Derived class: Bike
class Bike : public Vehicle {
private:
  int engineCapacity; // in CC
public:
  Bike(string regNum, string clr, int engineCap) : Vehicle(regNum, clr), engineCapacity(engineCap) {}
  // Overriding the writeToFile method
  void writeToFile(ofstream& file) const override {
    file << "Bike: " << registrationNumber << ", " << color << ", Engine Capacity: " << engineCapacity <<
"cc" << endl;
  }
  void display() const override {
    cout << "Bike - Reg No: " << registrationNumber << ", Color: " << color << ", Engine: " <<
engineCapacity << "cc" << endl;
  }
};
int main() {
  string regNum, color;
  int choice, seats, engineCap;
  ofstream file("vehicles.txt", ios::app); // Open file in append mode
  if (!file) {
    cout << "Error opening file!" << endl;
    return 1;
```

```
cout << "Choose Vehicle Type:\n1. Car\n2. Bike\nEnter choice: ";
cin >> choice;
cin.ignore(); // To clear newline character from input buffer
cout << "Enter Registration Number: ";
getline(cin, regNum);
cout << "Enter Color: ";
getline(cin, color);
if (choice == 1) {
cout << "Enter Number of Seats: ";
cin >> seats;
Car car(regNum, color, seats);
car.writeToFile(file);
cout << "Car details saved!\n";
car.display();
} else if (choice == 2) {
cout << "Enter Engine Capacity (in CC): ";
cin >> engineCap;
Bike bike(regNum, color, engineCap);
bike.writeToFile(file);
cout << "Bike details saved!\n";
bike.display();
} else {
cout << "Invalid choice!" << endl;
}
file.close();
return 0;
}
```

Output

```
"E:\C++\C-_CourseWorks\Wo

Choose Vehicle Type:
1. Car
2. Bike
Enter choice: 1
Enter Registration Number: 9856
Enter Color: red
Enter Number of Seats: 5
Car details saved!
Car - Reg No: 9856, Color: red, Seats: 5

Process returned 0 (0x0)   execution time : 20.467 s
Press any key to continue.
```

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 Implementation

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <stdexcept>
using namespace std;
// Custom exception for invalid marks
class InvalidMarksException : public exception {
public:
   const char* what() const throw() {
      return "Invalid marks! Marks should be between 0 and 100.";
   }
};
// Structure to store student record
struct Student {
   int roll;
   string name;
   int marks;
};
// Function to read student records from file
vector<Student> readRecords(const string& filename) {
   vector<Student> students;
   ifstream file(filename);
   // If file does not exist, create it and display a message
   if (!file) {
      cout << "File not found! Creating a new file..." << endl;
```

```
ofstream newFile(filename); // Create the file
newFile.close(); // Close after creation
return students; // Return empty list
}
string line;
while (getline(file, line)) {
stringstream ss(line);
Student s;
ss >> s.roll;
ss.ignore(); // Ignore space
getline(ss, s.name, ',');
ss >> s.marks;

// Validate marks
if (s.marks < 0 || s.marks > 100) {
throw InvalidMarksException();
}
students.push_back(s);
}
file.close();
return students;
}
// Function to add a new student record
void addRecord(vector<Student>& students) {
Student s;
cout << "Enter roll number: ";
cin >> s.roll;
cin.ignore(); // Ignore newline character
cout << "Enter name: ";
```

```
getline(cin, s.name);
cout << "Enter marks (0-100): ";
cin >> s.marks;

// Validate marks
if (s.marks < 0 || s.marks > 100) {
throw InvalidMarksException();
}

students.push_back(s);
}
// Function to save student records back to file
void saveRecords(const string& filename, const vector<Student>& students) {
ofstream file(filename);
if (!file) {
cout << "Error opening file for writing!" << endl;
return;
}
for (const Student& s : students) {
file << s.roll << " " << s.name << "," << s.marks << endl;
}
file.close();
cout << "Records saved successfully!" << endl;
}
int main() {
string filename = "students.txt";
vector<Student> students;
try {
students = readRecords(filename);
```

```
} catch (const exception& e) {
cout << "Error reading file: " << e.what() << endl;
return 1; // Exit if there's an issue
}
cout << "Existing 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

```
"E:\C++\Cpp-CourseWorks-m

Existing student records:
50 sujal adhikari 50
Do you want to add a new student? (y/n): y
Enter roll number: 1982
Enter name: ranjan
Enter marks (0-100): 77
Records saved successfully!

Process returned 0 (0x0)   execution time : 15.963 s
Press any key to continue.
```

CONCLUSION

I have learnt a lot more about C++ and Object-Oriented Programming through this project. I was introduced to designing classes, implementing operator overloading, inheritance, etc. Furthermore, I learned how to implement error handling in my programs to prevent failure and write and read data using file I/O operations. The importance of this assignment is that it has helped me improve my C++ technical skills not just that, but it has improved my problem-solving skills as well, so I can be more confident while heading towards a more complex programming world.