Worksheet 3

-Utsab Poudel

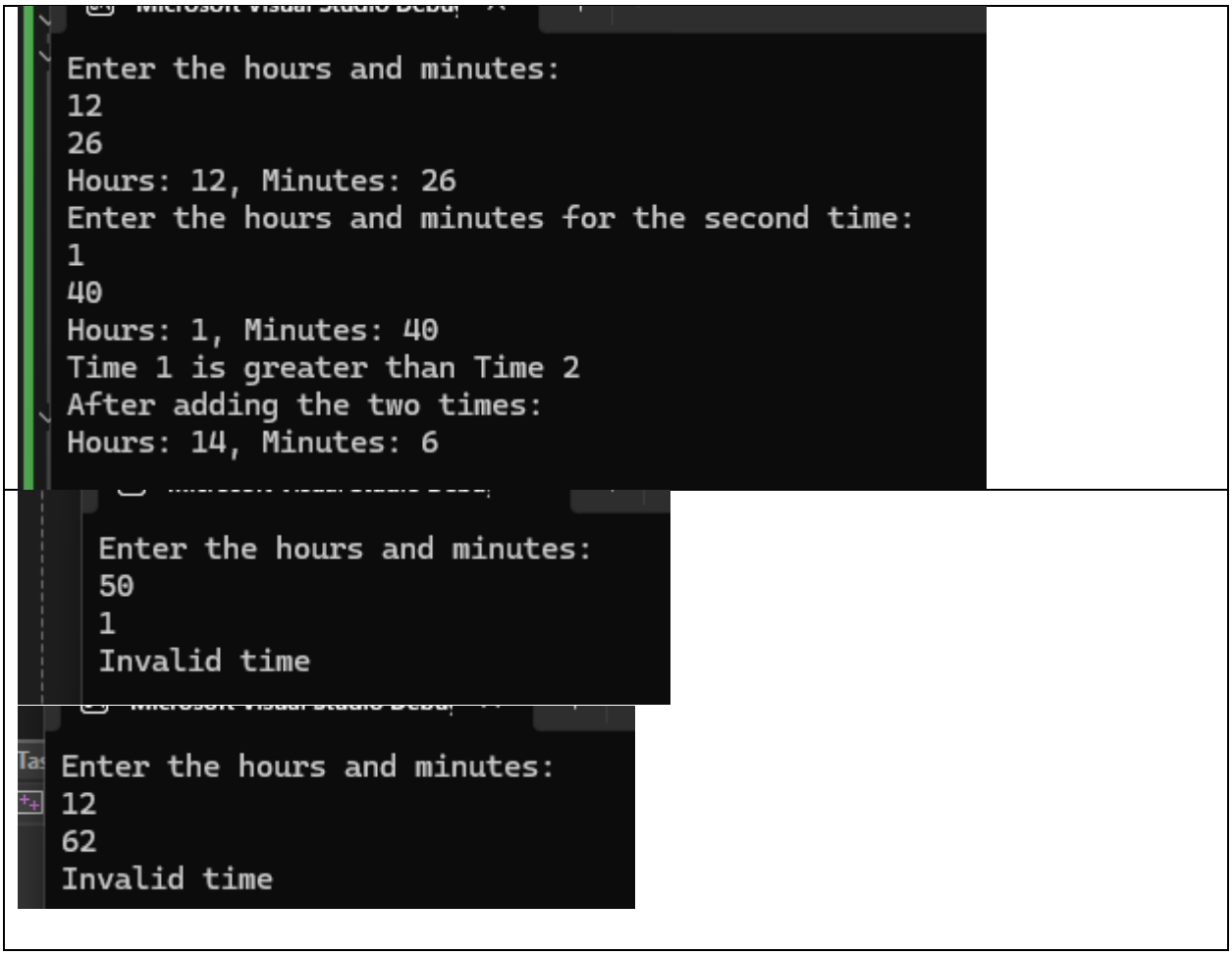
Task 1 : 30 marks

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

```
#include <iostream>
using namespace std;

class Measure {
private:
	int hours;
	int minutes;
public:
	Measure() {
		hours = 0;
		minutes = 0;
	}

	Measure(int h, int m) {
		if (h < 0 || m < 0 || h > 24 || m > 60) {
			int a = 1;
			throw (a);
		}
		hours = h;
		minutes = m;
	}

	Measure operator+(Measure m) {
		Measure temp;
		temp.minutes = minutes + m.minutes;
		int ff = temp.minutes / 60;
		temp.minutes = temp.minutes % 60;
		temp.hours = hours + m.hours + ff;

		if (temp.hours > 24) {
			int a = 1;
			throw (a);
		}
		return temp;
	}
	bool operator>(Measure m) {
```

```
            return (hours > m.hours) || (hours == m.hours && minutes >
m.minutes);
      }



      void display() {
            cout << "Hours: " << hours << ", Minutes: " << minutes << endl;
      }

      void menu() {
            try {
                  int hours, minutes;
                  cout << "Enter the hours and minutes: " << endl;
                  cin >> hours >> minutes;
                  Measure t1(hours, minutes);
                  t1.display();
                  cout << "Enter the hours and minutes for the second time: " <<
endl;
                  cin >> hours >> minutes;
                  Measure t2(hours, minutes);
                  t2.display();
                  if (t1 > t2) {
                        cout << "Time 1 is greater than Time 2" << endl;
                  }
                  else {
                        cout << "Time 2 is greater than Time 1" << endl;
                  }
                  Measure t3 = t1 + t2;
                  cout << "After adding the two times: " << endl;
                  t3.display();
            }
            catch (int a) {
                  cout << "Invalid time" << endl;
            }
      }
};

      int main()
      {
            Measure t1;
            t1.menu();
            return 0;
      }
```

Output:



Task 2: 70 marks

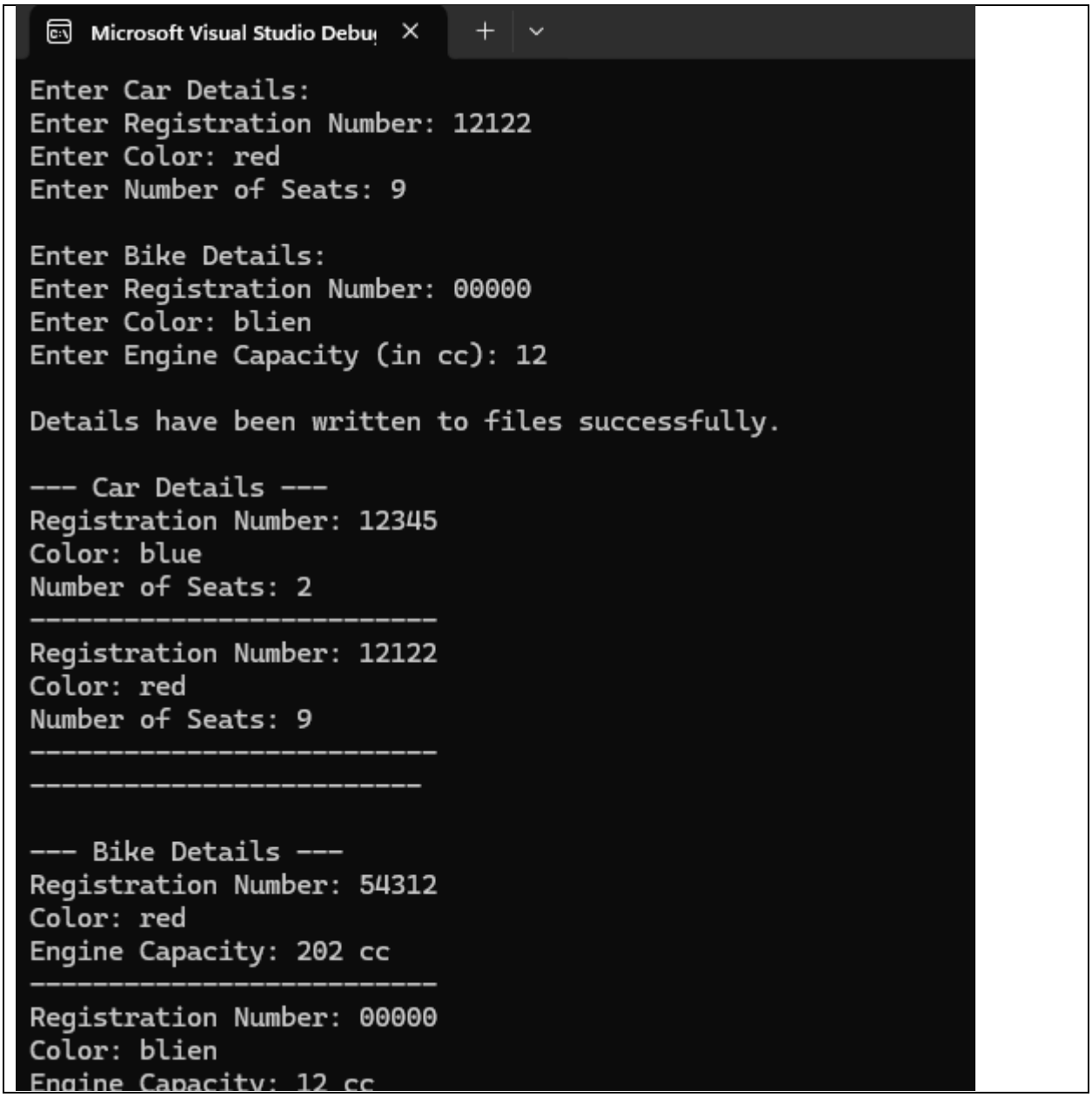
1. Create a base class Vehicle and two derived classes Car and Bike:
    1. Vehicle has registration number and color
    2. Car adds number of seats
    3. Bike adds engine capacity
    4. Each class should have its own method to write its details to a file
    5. Include proper inheritance and method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

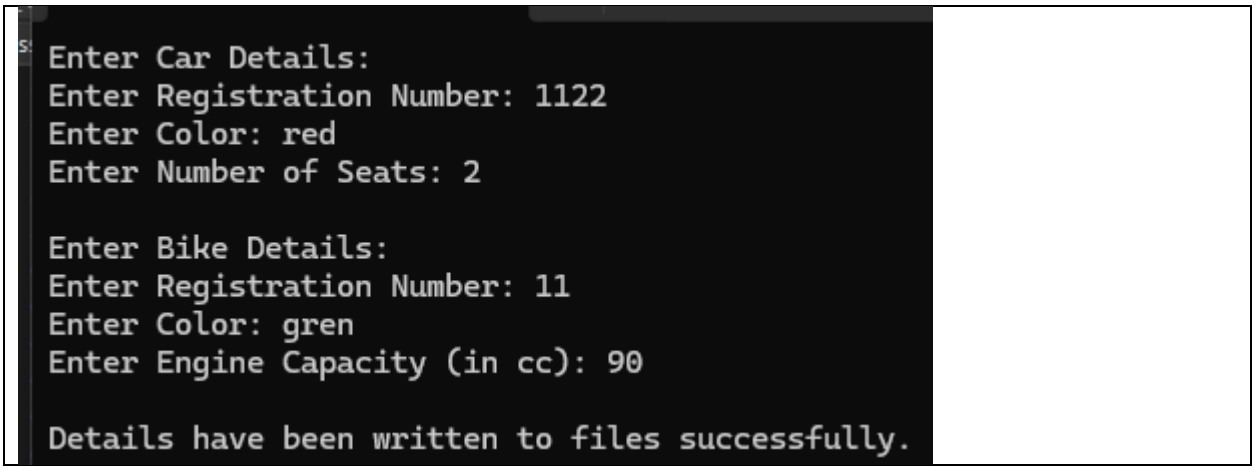
```
#include <iostream>
#include <fstream>
#include <string>
```

```cpp
using namespace std;

class Vehicle {
protected:
    string registrationNumber;
    string color;

public:
    void inputDetails() {
        cout << "Enter Registration Number: ";
        cin >> registrationNumber;
        cout << "Enter Color: ";
        cin >> color;
    }

    virtual void writeDetailsToFile() {
        ofstream outFile("VehicleDetails.txt", ios::app);
        outFile << "Registration Number: " << registrationNumber << endl;
        outFile << "Color: " << color << endl;
        outFile << "-------------------------" << endl;
        outFile.close();
    }

    virtual ~Vehicle() {}
};

class Car : public Vehicle {
protected:
    int numberOfSeats;

public:
    void inputDetails() {
        Vehicle::inputDetails();
        cout << "Enter Number of Seats: ";
        cin >> numberOfSeats;
    }

    void writeDetailsToFile() override {
        ofstream outFile("CarDetails.txt", ios::app);
        outFile << "Registration Number: " << registrationNumber << endl;
        outFile << "Color: " << color << endl;
        outFile << "Number of Seats: " << numberOfSeats << endl;
        outFile << "-------------------------" << endl;
        outFile.close();
    }
};

class Bike : public Vehicle {
protected:
    int engineCapacity;

public:
    void inputDetails() {
        Vehicle::inputDetails();
        cout << "Enter Engine Capacity (in cc): ";
        cin >> engineCapacity;
    }
```

```
    void writeDetailsToFile() override {
        ofstream outFile("BikeDetails.txt", ios::app);
        outFile << "Registration Number: " << registrationNumber << endl;
        outFile << "Color: " << color << endl;
        outFile << "Engine Capacity: " << engineCapacity << " cc" << endl;
        outFile << "-------------------------" << endl;
        outFile.close();
    }
};

class Menu {
public:
    void displayMenu() {
        Car car;
        Bike bike;

        cout << "Enter Car Details:\n";
        car.inputDetails();
        car.writeDetailsToFile();

        cout << "\nEnter Bike Details:\n";
        bike.inputDetails();
        bike.writeDetailsToFile();

        cout << "\nDetails have been written to files successfully.\n";
    }

    void displayFileContent() {
        displayFile("CarDetails.txt", "Car Details");
        displayFile("BikeDetails.txt", "Bike Details");
    }

private:
    void displayFile(const string& fileName, const string& header) {
        ifstream inFile(fileName);
        if (!inFile) {
            cout << "Could not open " << fileName << endl;
            return;
        }
        string line;
        cout << "\n--- " << header << " ---\n";
        while (getline(inFile, line)) {
            cout << line << endl;
        }
        cout << "------------------------\n";
        inFile.close();
    }
};

int main() {
    Menu m1;
    m1.displayMenu();
    m1.displayFileContent();
    return 0;
}
```

Output:


Microsoft Visual Studio Debu
Enter Car Details:
Enter Registration Number: 12122
Enter Color: red
Enter Number of Seats: 9
Enter Bike Details:
Enter Registration Number: 00000
Enter Color: blien
Enter Engine Capacity (in cc): 12
Details have been written to files successfully.
--- Car Details ---
Registration Number: 12345
Color: blue
Number of Seats: 2
Registration Number: 12122
Color: red
Number of Seats: 9
--- Bike Details ---
Registration Number: 54312
Color: red
Engine Capacity: 202 cc
Registration Number: 00000
Color: blien
Engine Capacity: 12 cc



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

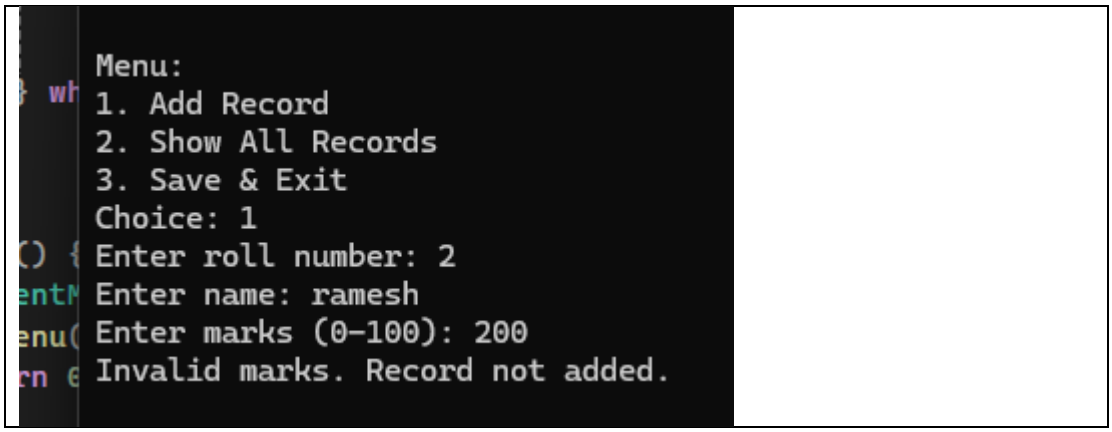
```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class Student {
public:
    int roll;
    string name;
    int marks;
};

class StudentManagement {
    Student students[100];
    int count;
    string filename;

public:
    StudentManagement() {
        count = 0;
        filename = "students.txt";
    }

    void readRecords() {
        ifstream file(filename);
        if (!file) {
            cout << "File not found. A new one will be created.\n";
```

```
            return;
        }

        while (file >> students[count].roll >> students[count].name >>
students[count].marks) {
            if (students[count].marks >= 0 && students[count].marks <=
100) {
                count++;
            }
            else {
                cout << "Invalid marks in file for student " <<
students[count].name << ". Skipping.\n";
            }
        }

        file.close();
    }

    void addRecord() {
        cout << "Enter roll number: ";
        cin >> students[count].roll;
        cout << "Enter name: ";
        cin >> students[count].name;
        cout << "Enter marks (0-100): ";
        cin >> students[count].marks;

        if (students[count].marks >= 0 && students[count].marks <= 100)
{
            cout << "Record added successfully.\n";
            count++;
        }
        else {
            cout << "Invalid marks. Record not added.\n";
        }
    }

    void writeRecords() {
        ofstream file(filename);
        for (int i = 0; i < count; i++) {
            file << students[i].roll << " " << students[i].name << " "
<< students[i].marks << endl;
        }
        file.close();
    }

    void showRecords() {
        for (int i = 0; i < count; i++) {
            cout << "Roll: " << students[i].roll << ", Name: " <<
students[i].name << ", Marks: " << students[i].marks << endl;
        }
    }

    void menu() {
        readRecords();

        int choice;
        do {
```

```
            cout << "\nMenu:\n1. Add Record\n2. Show All Records\n3.
Save & Exit\nChoice: ";
            cin >> choice;

            if (choice == 1) {
                addRecord();
            }
            else if (choice == 2) {
                showRecords();
            }
            else if (choice == 3) {
                writeRecords();
                cout << "Records saved.\n";
            }
            else {
                cout << "Invalid choice.\n";
            }
        } while (choice != 3);
    }
};

int main() {
    StudentManagement s1;
    s1.menu();
    return 0;
}
```

Output:

```
Menu:
1. Add Record
2. Show All Records
3. Save & Exit
Choice: 1
Enter roll number: 1
Enter name: Ram
Enter marks (0-100): 30
Record added successfully.

Menu:
1. Add Record
2. Show All Records
3. Save & Exit
Choice: 2
Roll: 1, Name: Ram, Marks: 30
```



Task 3

- Check and commit all your solutions.
- This task carries no marks but it is mandatory. Ensure that your solution is visible to us.