C++ Worksheet Task_3

ASSESSMENT

WEIGHTAGE AND TYPE: 12.5%

YEAR: 2024-25

STUDENT NAME: SWOYAMRAJ SHRESTHA

STUDENT ID: 24030185

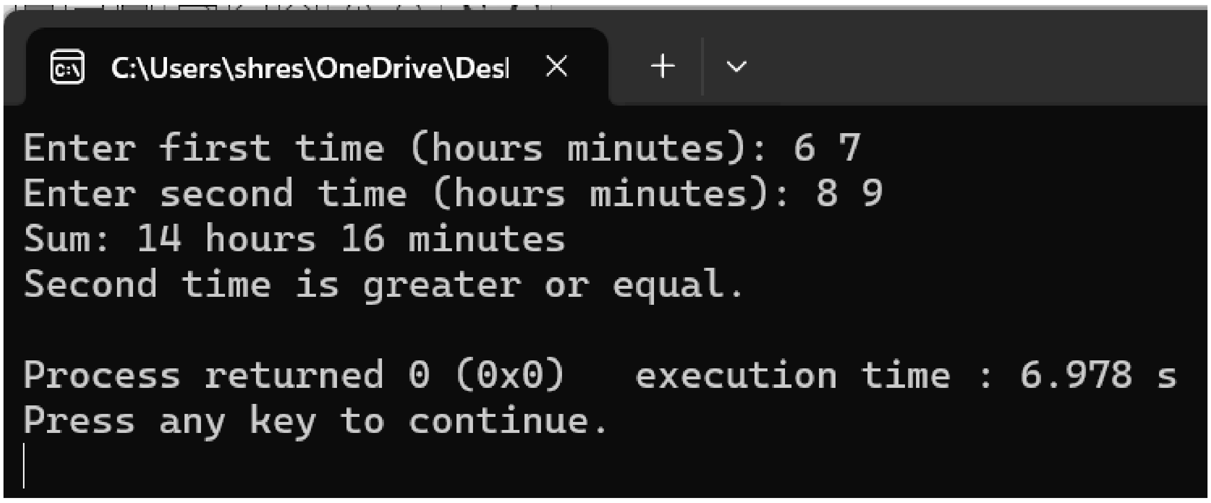
Question_1.1

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

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

class ITE : public exception
{
public:
    const char* what() const noexcept override
    {
        return "Invalid time! Time Hours must be less than or equals to 24 and minutes must be less than 60.";
    }
};

class Time
{
private:
    int hours;
    int minutes;

    void validate()
    {
        if (hours > 24 || minutes >= 60)
            {
                throw ITE();
            }
    }

public:
    Time(int h = 0, int m = 0) : hours(h), minutes(m) // Constructor with default values
    {
        validate();
    }
    Time operator+(const Time& other) const
    {
        int totalHours = hours + other.hours;
        int totalMinutes = minutes + other.minutes;

        if (totalMinutes >= 60)
            {
                totalHours += totalMinutes / 60;
                totalMinutes %= 60;
            }
        return Time(totalHours, totalMinutes);
    }

    bool operator>(const Time& other) const
    {
        return (hours * 60 + minutes) > (other.hours * 60 + other.minutes);
    }

    void display() const
    {
        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);

        Time sum = t1 + t2; //sum of times
        cout << "Sum: ";
        sum.display();

        if (t1 > t2) //comparing time
```

```
            {
                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:



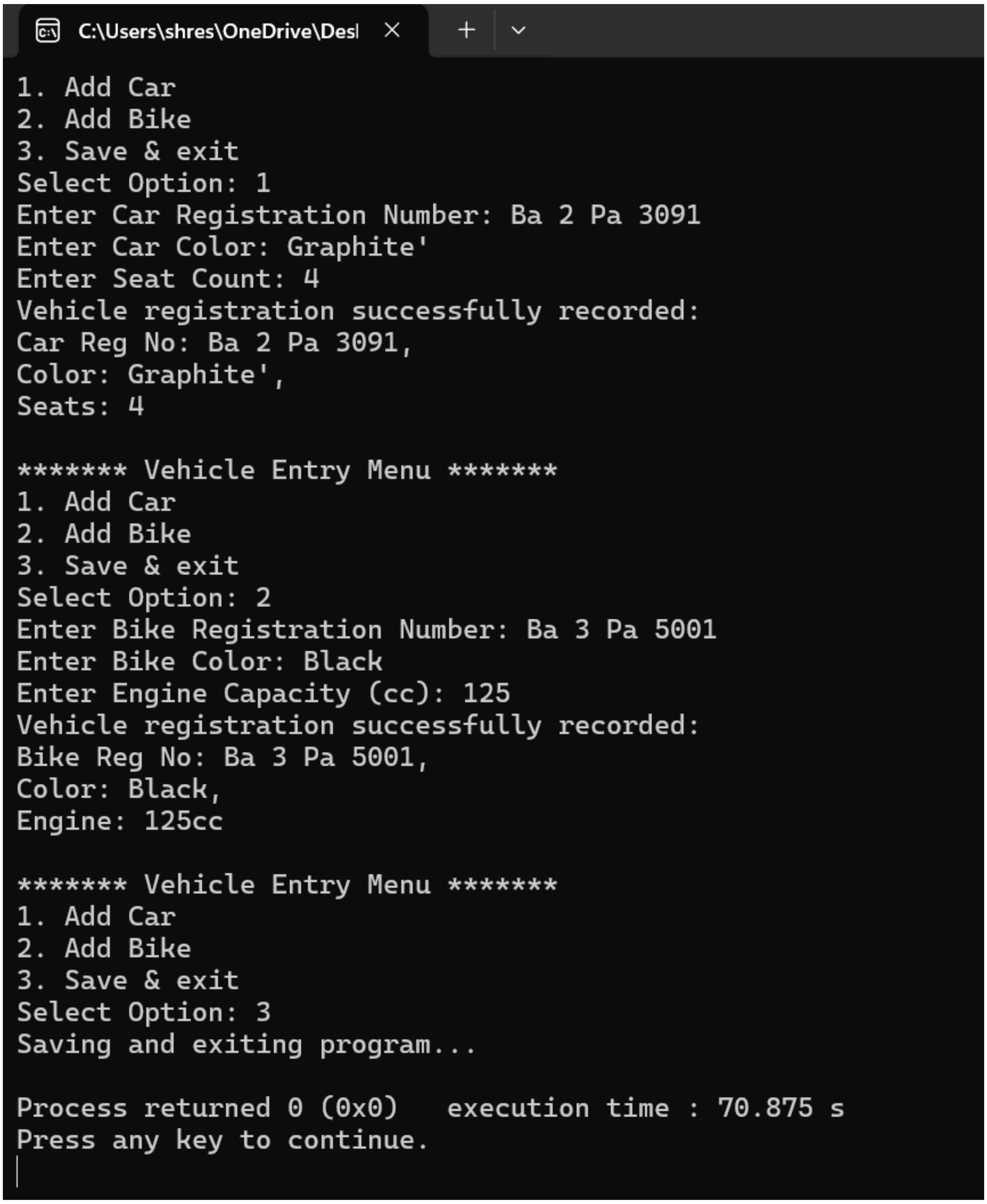
```
Enter first time (hours minutes): 6 7
Enter second time (hours minutes): 8 9
Sum: 14 hours 16 minutes
Second time is greater or equal.

Process returned 0 (0x0)   execution time : 6.978 s
Press any key to continue.
```

Question_2.1

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 <memory>
using namespace std;

class Vehicle // Base Class
{
protected:
    string regNo;
    string paint;
public:
    Vehicle(const string& reg, const string& clr) : regNo(reg), paint(clr) {}
    virtual void saveToFile(ofstream& out) const
    {
        out << "Vehicle - Reg: " << regNo << ", Color: " << paint << endl;
    }
    virtual void print() const
    {
        cout << "Vehicle -> Reg No: " << regNo << ", Color: " << paint << endl;
    }
    virtual ~Vehicle() {}
};

class FourWheeler : public Vehicle // Derived Class - Four Wheeler
{
    int seatCount;
public:
    FourWheeler(const string& reg, const string& clr, int seats) : Vehicle(reg,
clr), seatCount(seats) {}
    void saveToFile(ofstream& out) const override
    {
        out << "Car - Reg: " << regNo << ", Color: " << paint << ", Seats: " <<
seatCount << endl;
    }
    void print() const override
    {
        cout << "Car Reg No: " << regNo << ",\nColor: " << paint << ",\nSeats: "
<< seatCount << endl;
    }
};

class TwoWheeler : public Vehicle // Derived Class - Two Wheeler
{
    int cc;
public:
    TwoWheeler(const string& reg, const string& clr, int engineCC) :
Vehicle(reg, clr), cc(engineCC) {}
    void saveToFile(ofstream& out) const override
    {
        out << "Bike - Reg: " << regNo << ", Color: " << paint << ", Engine: " <<
cc << "cc" << endl;
    }
    void print() const override
    {
        cout << "Bike Reg No: " << regNo << ",\nColor: " << paint << ",\nEngine:
" << cc << "cc" << endl;
    }
};

int main()
{
    ofstream record("Vehicle's Registration.txt", ios::app);
    if (!record.is_open())
    {
        cerr << "The file does not exist!" << endl;
        return -1;
    }

    int option = 0;
    do
    {
        cout << "\n******* Vehicle Entry Menu *******" << endl;
        cout << "1. Add Car\n2. Add Bike\n3. Save & exit\nSelect Option: ";
        cin >> option;
        cin.ignore(); //clears the input buffer

        string rno, col;
        unique_ptr<Vehicle> ptr = nullptr;

    switch (option)
    {
        case 1:
        {
            int seats;
            cout << "Enter Car Registration Number: ";
            getline(cin, rno);
            cout << "Enter Car Color: ";
            getline(cin, col);
            cout << "Enter Seat Count: ";
            cin >> seats;
            cin.ignore(); //clears newline from buffer
            ptr = make_unique<FourWheeler>(rno, col, seats);
            break;
        }
        case 2:
        {
            int engine;
            cout << "Enter Bike Registration Number: ";
            getline(cin, rno);
            cout << "Enter Bike Color: ";
            getline(cin, col);
            cout << "Enter Engine Capacity (cc): ";
            cin >> engine;
            cin.ignore();
            ptr = make_unique<TwoWheeler>(rno, col, engine);
            break;
        }
            case 3:
                cout << "Saving and exiting program..." << endl;
                break;
            default:
                cout << "Invalid input! Please choose 1, 2, or 3." << endl;
        }

        if (ptr)
        {
            ptr->saveToFile(record);
            cout << "Vehicle registration successfully recorded:\n";
            ptr->print();
        }
    } while (option != 3);

    record.close();
    return 0;
}
```

Output:



```
1. Add Car
2. Add Bike
3. Save & exit
Select Option: 1
Enter Car Registration Number: Ba 2 Pa 3091
Enter Car Color: Graphite'
Enter Seat Count: 4
Vehicle registration successfully recorded:
Car Reg No: Ba 2 Pa 3091,
Color: Graphite',
Seats: 4

******* Vehicle Entry Menu *******
1. Add Car
2. Add Bike
3. Save & exit
Select Option: 2
Enter Bike Registration Number: Ba 3 Pa 5001
Enter Bike Color: Black
Enter Engine Capacity (cc): 125
Vehicle registration successfully recorded:
Bike Reg No: Ba 3 Pa 5001,
Color: Black,
Engine: 125cc

******* Vehicle Entry Menu *******
1. Add Car
2. Add Bike
3. Save & exit
Select Option: 3
Saving and exiting program...

Process returned 0 (0x0)   execution time : 70.875 s
Press any key to continue.
```

Question_2.2

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

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

struct Student
{
   int roll;
   string name;
   int marks;
};

void validateMarks(int marks)
{
   if (marks < 0 || marks > 100)
   {
      throw out_of_range("Marks should be between 0 and 100.");
   }
}

vector<Student> readRecords(string fileName)
{
   vector<Student> records;
   ifstream File(fileName);

   if (!File)
   {
      cout << "The file does not exist.\n";
      return records;
   }

   Student student;
   while (File >> student.roll >> student.name >> student.marks)
```

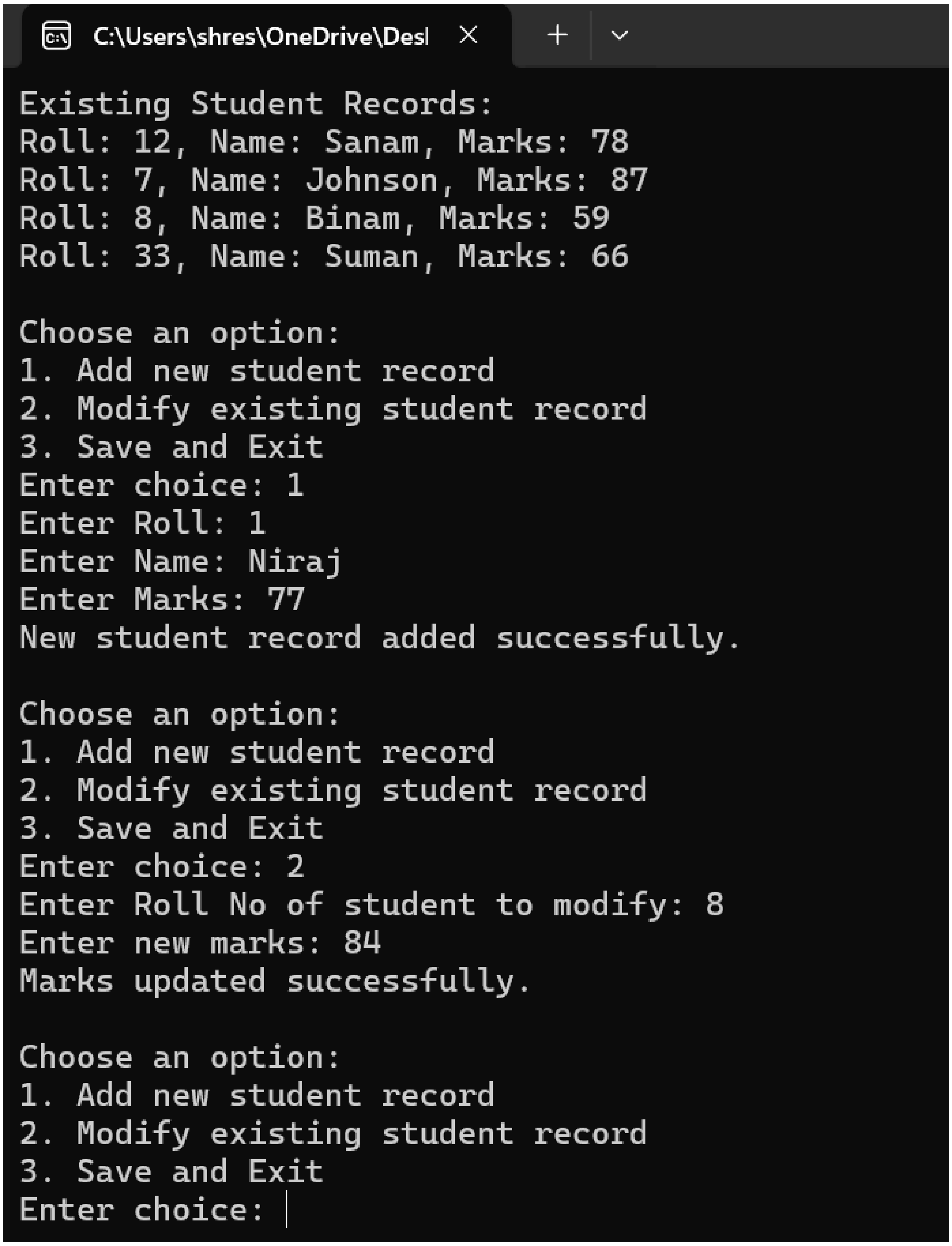
```
        {
           records.push_back(student);
        }

    File.close();
    return records;
}

void saveStdRecords(string fileName, vector<Student> records)
{
    ofstream File(fileName);

    if (!File)
    {
        cout << "Error opening file for writing!\n";
        return;
    }

    for (const auto& records : records)
    {
        File << records.roll << " " << records.name << " " << records.marks <<
endl;
    }
    File.close();
}

int main()
{
    string fileName = "Student's Record.txt";
    vector<Student> studentlist = readRecords(fileName);

    if (!studentlist.empty())
    {
        cout << "Existing Student Records:\n";
        for (const auto& student : studentlist)
        {
                cout << "Roll: " << student.roll << ", Name: " << student.name << ",
Marks: " << student.marks << endl;
        }
    }
    else
    {
        cout << "No records found.\n";
```

```
}

bool running = true;
while (running)
  {
    int userchoice;
    cout << "\nChoose an option:\n";
    cout << "1. Add new student record\n";
    cout << "2. Modify existing student record\n";
    cout << "3. Save and Exit\n";
    cout << "Enter choice: ";
    cin >> userchoice;

  if (userchoice == 1)
    {
      Student newStudent; // Option to add a new student
      cout << "Enter Roll: ";
      cin >> newStudent.roll;
      cin.ignore(); // To clear the buffer after taking integer input
      cout << "Enter Name: ";
      getline(cin, newStudent.name);
      cout << "Enter Marks: ";
      cin >> newStudent.marks;

      try
      {
        validateMarks(newStudent.marks);
        studentlist.push_back(newStudent);
        cout << "New student record added successfully.\n";
      }
  catch (const out_of_range& e)
  {
    cout << "Error: " << e.what() << endl;
  }
    }

  else if (userchoice == 2)
    {
      int rollNoToModify;
      cout << "Enter Roll No of student to modify: ";
      cin >> rollNoToModify;

      bool recfound = false;
```

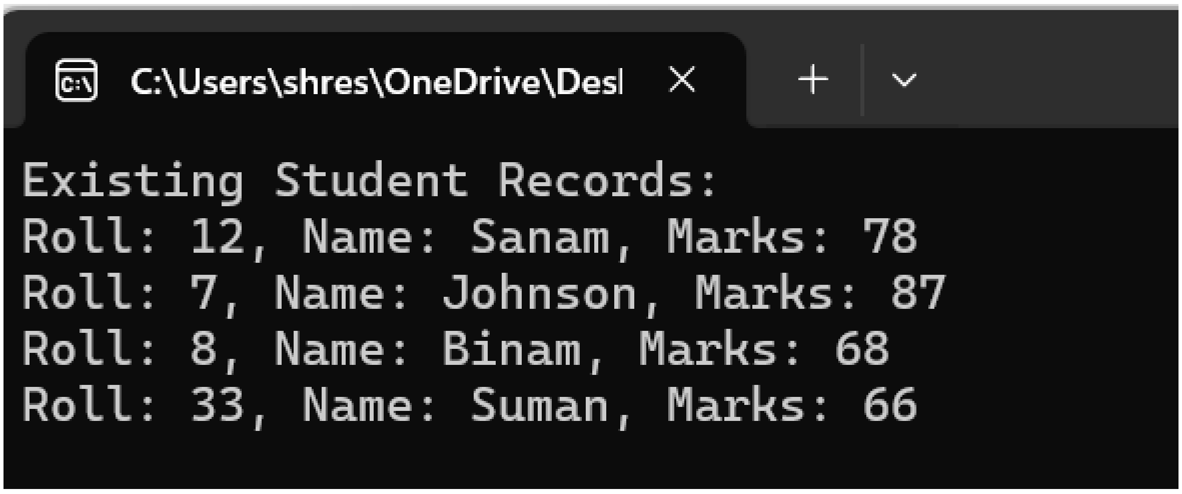
```
            for (auto& student : studentlist)
              {
                if (student.roll == rollNoToModify)
                  {
                    recfound = true;
                    cout << "Enter new marks: ";
                    int newMarks;
                    cin >> newMarks;

                try
                  {
                  validateMarks(newMarks);
                  student.marks = newMarks;
                  cout << "Marks updated successfully.\n";
                  }
                catch (const out_of_range& e)
                  {
                    cout << "Error: " << e.what() << endl;
                  }
                break;
                  }
              }
                if (!recfound)
                  {
                    cout << "Student with Roll No " << rollNoToModify  << "
not found.\n";
                  }
          }

        else if (userchoice == 3)
          {
            saveStdRecords(fileName, studentlist);
            cout << "Records saved successfully! Exiting program.\n";
            running = false;
          }

        else
          {
            cout << "Invalid choice. Please try again.\n";
          }
        }

    return 0;
```

}

Output:



```
Existing Student Records:
Roll: 12, Name: Sanam, Marks: 78
Roll: 7, Name: Johnson, Marks: 87
Roll: 8, Name: Binam, Marks: 59
Roll: 33, Name: Suman, Marks: 66

Choose an option:
1. Add new student record
2. Modify existing student record
3. Save and Exit
Enter choice: 1
Enter Roll: 1
Enter Name: Niraj
Enter Marks: 77
New student record added successfully.

Choose an option:
1. Add new student record
2. Modify existing student record
3. Save and Exit
Enter choice: 2
Enter Roll No of student to modify: 8
Enter new marks: 84
Marks updated successfully.

Choose an option:
1. Add new student record
2. Modify existing student record
3. Save and Exit
Enter choice:
```



```
Existing Student Records:
Roll: 12, Name: Sanam, Marks: 78
Roll: 7, Name: Johnson, Marks: 87
Roll: 8, Name: Binam, Marks: 68
Roll: 33, Name: Suman, Marks: 66
```