

Worksheet – 3

Name : Crishtina K.C.

Student ID : 23085130

Cyber Security And Digital Foresics  
  
Github link: <https://github.com/crishtina01/cpp_Worksheet>

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 InvalidTimeError

{

public:

string message;

InvalidTimeError(string msg) : message(msg) {}

};

class Time

{

private:

int hours;

int minutes;

public:

Time()

{

cout << "Enter time in hours (0-24): ";

cin >> hours;

cout << "Enter time in minutes (0-60): ";

cin >> minutes;

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

{

throw InvalidTimeError("Invalid time! Hours must be between 0 and 23, and minutes must be between 0 and 59.");

}

}

Time add(const Time& other) const

{

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

int newHours = (totalMinutes / 60) % 24;

int newMinutes = totalMinutes % 60;

return Time(newHours, newMinutes);

}

Time(int h, int m) : hours(h), minutes(m) {}

bool isGreaterThan(const Time& other) const

{

int thisTotalMinutes = hours \* 60 + minutes;

int otherTotalMinutes = other.hours \* 60 + other.minutes;

return thisTotalMinutes > otherTotalMinutes;

}

void display() const

{

cout << (hours < 10 ? "0" : "") << hours << ":"

<< (minutes < 10 ? "0" : "") << minutes << endl;

}

};

int main()

{

try

{

cout << "Enter Time 1:" << endl;

Time time1;

cout << "Enter Time 2:" << endl;

Time time2;

cout << "Time 1: ";

time1.display();

cout << "Time 2: ";

time2.display();

Time sumTime = time1.add(time2);

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

sumTime.display();

if (time1.isGreaterThan(time2))

{

cout << "Time 1 is greater than Time 2." << endl;

}

else

{

cout << "Time 2 is greater than Time 1." << endl;

}

}

catch (InvalidTimeError& e)

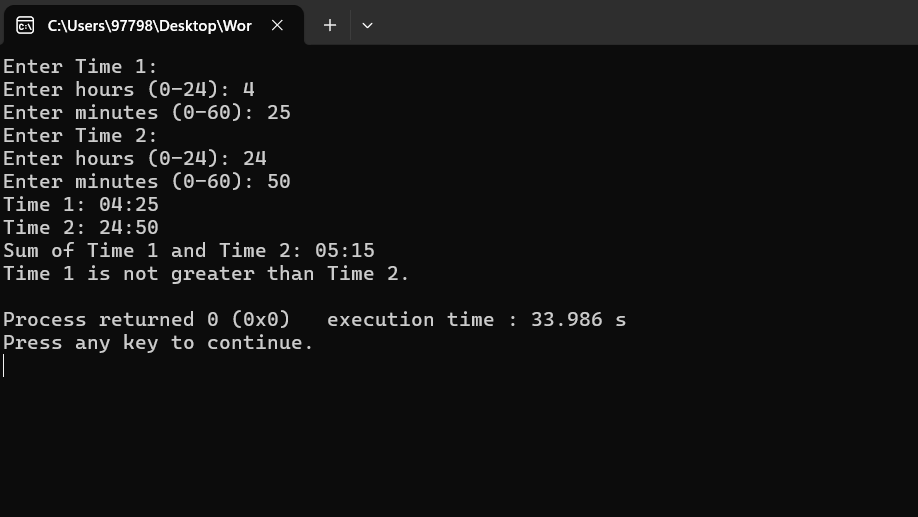
{

cout << "Error: " << e.message << endl;

}

return 0;

}



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>

using namespace std;

class Vehicle

{

protected:

string registrationNumber;

string color;

public:

Vehicle(string regNo, string col)

{

registrationNumber = regNo;

color = col;

}

virtual void display()

{

cout << "Registration Number: " << registrationNumber << endl;

cout << "Color: " << color << endl;

}

virtual void writeToFile(ofstream& file)

{

file << "Registration Number: " << registrationNumber << endl;

file << "Color: " << color << endl;

}

virtual ~Vehicle() {}

};

class Car : public Vehicle

{

private:

int numberOfSeats;

public:

Car(string regNo, string col, int seats) : Vehicle(regNo, col)

{

numberOfSeats = seats;

}

void display() override

{

cout << "\n--- Car Details ---" << endl;

cout << "Vehicle Type: Car" << endl;

Vehicle::display();

cout << "Number of Seats: " << numberOfSeats << endl;

}

void writeToFile(ofstream& file) override

{

file << "\n--- Car Details ---" << endl;

file << "Vehicle Type: Car" << endl;

Vehicle::writeToFile(file);

file << "Number of Seats: " << numberOfSeats << endl;

}

};

class Bike : public Vehicle

{

private:

double engineCapacity;

public:

Bike(string regNo, string col, double capacity) : Vehicle(regNo, col)

{

engineCapacity = capacity;

}

void display() override

{

cout << "\n--- Bike Details ---" << endl;

cout << "Vehicle Type: Bike" << endl;

Vehicle::display();

cout << "Capacity of engine: " << engineCapacity << " cc" << endl;

}

void writeToFile(ofstream& file) override

{

file << "\n--- Bike Details ---" << endl;

file << "Vehicle Type: Bike" << endl;

Vehicle::writeToFile(file);

file << "Engine Capacity: " << engineCapacity << " cc" << endl;

}

};

int main()

{

int choice;

ofstream file("vehicle\_details.txt", ios::app);

if (!file)

{

cout << "Error: Unable to open your file for writing." << endl;

return 1;

}

while (true)

{

cout << "\nVehicle Menu:\n";

cout << "1. Add Car Details\n";

cout << "2. Add Bike Details\n";

cout << "3. Display All Vehicle Details\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1)

{

string regNo, color;

int seats;

cout << "Enter Car details:\n";

cout << "Registration Number: ";

cin >> regNo;

cout << "Color: ";

cin >> color;

cout << "Number of Seats: ";

cin >> seats;

Car car(regNo, color, seats);

car.writeToFile(file);

car.display();

}

else if (choice == 2)

{

string regNo, color;

double capacity;

cout << "Enter Bike details:\n";

cout << "Registration Number: ";

cin >> regNo;

cout << "Color: ";

cin >> color;

cout << "Capacity of engine(in cc): ";

cin >> capacity;

Bike bike(regNo, color, capacity);

bike.writeToFile(file);

bike.display();

}

else if (choice == 3)

{

cout << "\n--- All Vehicle Details ---\n";

ifstream inputFile("vehicle\_details.txt");

if (!inputFile)

{

cout << "Error: Unable to open file for reading." << endl;

return 1;

}

string line;

while (getline(inputFile, line))

{

cout << line << endl;

}

inputFile.close();

}

else if (choice == 4)

{

cout << "Exiting the program.\n";

break;

}

else

{

cout << "Please try again,Invalid choice.\n";

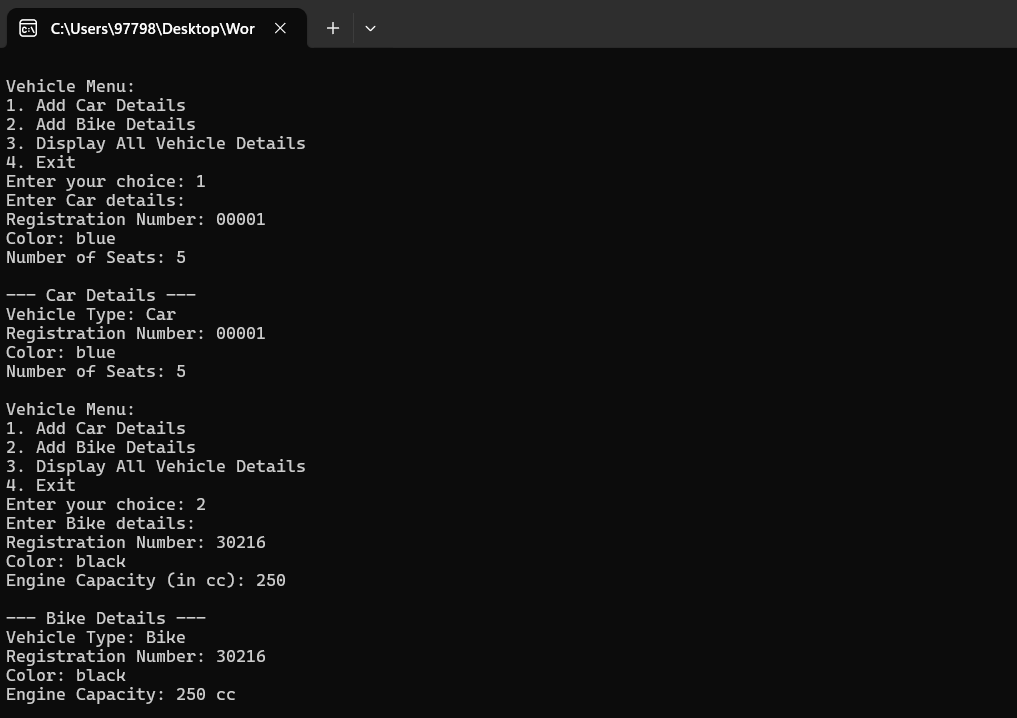
}

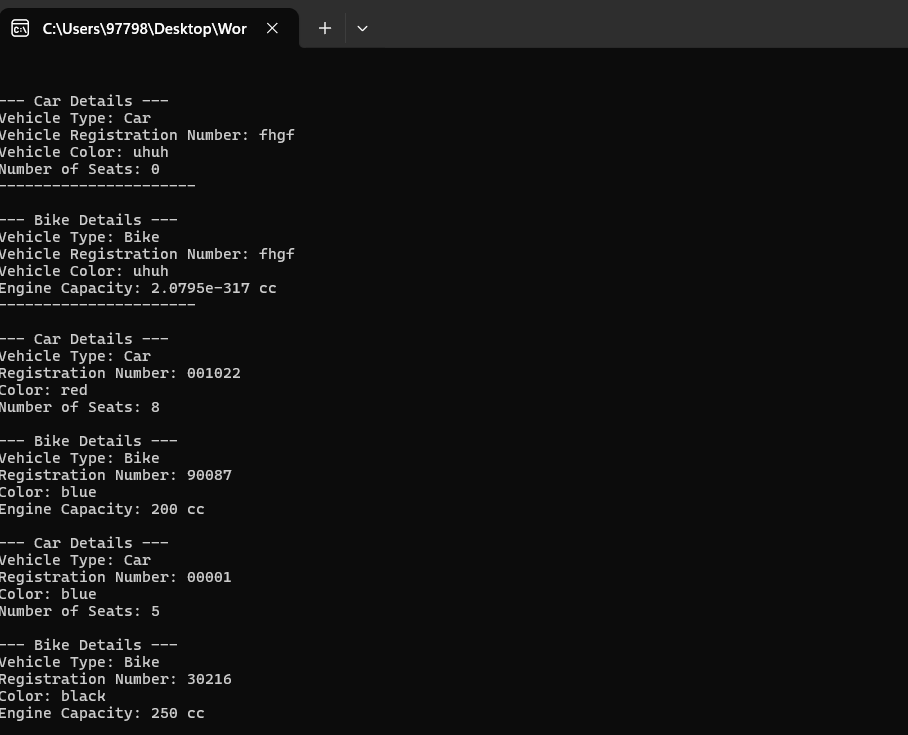
}

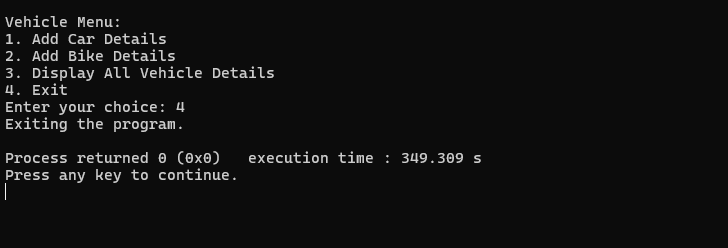
file.close();

return 0;

}







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

#include <iostream>

#include <fstream>

#include <string>

#include <stdexcept>

#include <vector>

#include <sstream>

using namespace std;

class Student

{

private:

int roll;

string name;

int marks;

public:

class InvalidMarksException : public exception

{

public:

const char\* what() const noexcept override

{

return "Invalid marks! Marks must be between 0 and 100.";

}

};

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

{

if (m < 0 || m > 100)

{

throw InvalidMarksException();

}

}

int getRoll() const

{

return roll;

}

string getName() const

{

return name;

}

int getMarks() const

{

return marks;

}

void display() const

{

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

}

void saveToFile(ofstream& file) const

{

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

}

static Student readFromFile(const string& line)

{

int r;

string n;

int m;

stringstream ss(line);

ss >> r >> n >> m;

return Student(r, n, m);

}

};

void readStudentRecords(const string& filename, vector<Student>& students)

{

ifstream file(filename);

if (!file.is\_open())

{

cout << "Error: Could not open your file for reading!" << endl;

return;

}

string line;

while (getline(file, line))

{

if (!line.empty())

{

try

{

Student s = Student::readFromFile(line);

students.push\_back(s);

}

catch (const Student::InvalidMarksException& e)

{

cout << "Error in record: " << line << " - " << e.what() << endl;

}

}

}

file.close();

}

void addStudentRecord(vector<Student>& students)

{

int roll;

string name;

int marks;

cout << "Enter Roll Number: ";

cin >> roll;

cout << "Enter Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Marks: ";

cin >> marks;

try

{

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

}

catch (const Student::InvalidMarksException& e)

{

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

}

}

void saveStudentRecords(const string& filename, const vector<Student>& students)

{

ofstream file(filename, ios::trunc);

if (!file.is\_open())

{

cout << "Error: Could not open your file for writing!" << endl;

return;

}

for (const auto& student : students)

{

student.saveToFile(file);

}

file.close();

}

void displayAllRecords(const vector<Student>& students)

{

cout << "\nStudent Records:\n";

for (const auto& student : students)

{

student.display();

}

}

int main()

{

string filename = "students.txt";

vector<Student> students;

try

{

readStudentRecords(filename, students);

int choice;

while (true)

{

cout << "\nMenu:\n";

cout << "1. Display All Student Records\n";

cout << "2. Add New Student Record\n";

cout << "3. Save and Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1)

{

displayAllRecords(students);

}

else if (choice == 2)

{

addStudentRecord(students);

}

else if (choice == 3)

{

saveStudentRecords(filename, students);

cout << "Records have been saved successfully!\n";

break;

}

else

{

cout << "!!!invalid !!! Enter correct choice.\n";

}

}

}

catch (const Student::InvalidMarksException& e)

{

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

}

return 0;

}

