Final Project

DOCUMENTATION

Course: Basics of Programming II

Instructors: Mr. Dunaev Dmitriy & Mr. Al-Magsoosi Husam

Program: Car Rental System

Done By: Maged Daoud

Neptun Code: I9XF28

Contents

Introduction:

This Car Rental Management System is a small software console-based application developed in C++ language using Object Oriented Programming (OOP) to manage the rental of cars.

This system allows users to perform various operations, such as adding new cars, renting cars, view rental cars history, and deleting a car.

The system is implemented using C++ programming language and provides a console based with command line interface for user doing operation.

Problem Statement:

The car rental industry is growing rapidly, and there is a need for efficient management systems to streamline the rental process. Traditional manual methods for managing car rentals can be time-consuming and error-prone. Therefore, the aim of this project is to develop a Car Rental Management System that automates the process and provides a reliable and user-friendly solution.

The system consists of several key functionalities, including the ability to add new cars to the rental inventory, rent cars to customers, track car availability, and delete a car. The system stores car data, such as car ID, year, model, make, daily price, and availability, in a text file. It allows users to load car data from the file and update the file with any changes made to the car inventory.

To ensure data integrity, the system includes validation checks for user inputs. For example, when adding new cars or renting cars, the system verifies that the input values are of the correct data type and use exception handling (try catch while reading or load data from file).

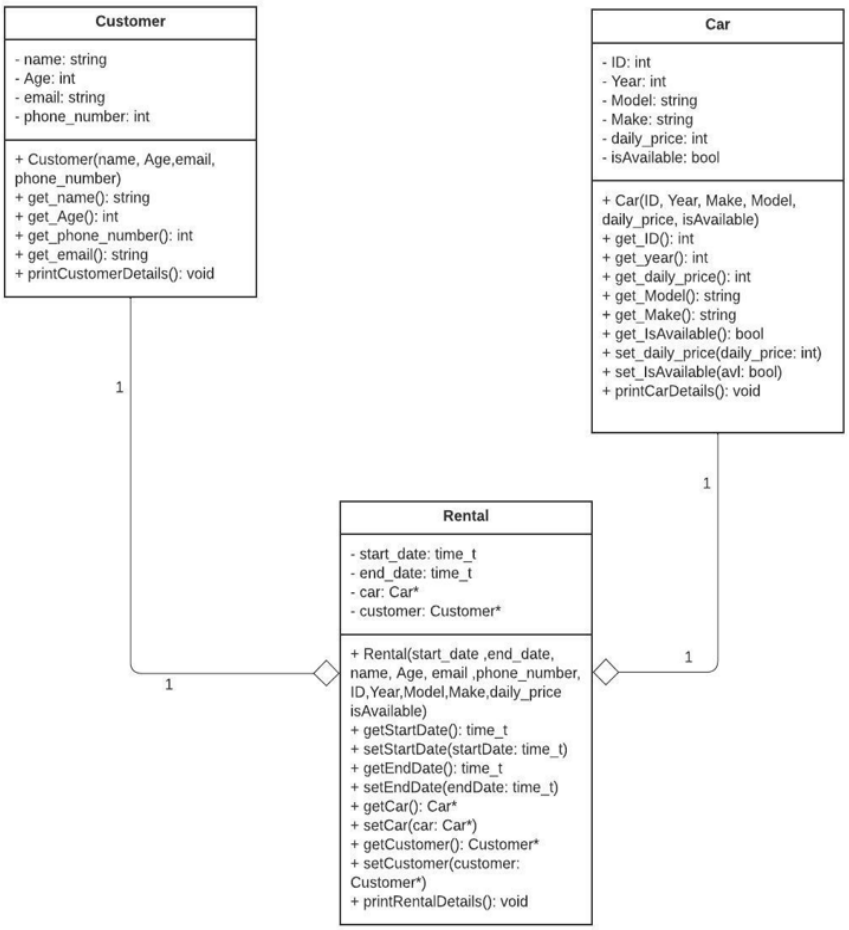
In case of invalid input, the system prompts the user to enter valid data.

The Car Rental Management System aims to simplify the rental process for both the rental company and the customers. It provides an efficient and organized way to manage car inventory, track rentals, and generate reports for analysis and decision-making. By automating manual tasks and reducing the chances of errors, the system improves overall productivity and customer satisfaction.

Class Diagram:

I implemented the solution using object-oriented programming (OOP) that uses the concepts of classes and objects. I have created 3 classes **Car**, **Rental** and **Customer**. each class has private attributes that represent the features of each class and public methods that perform the operations.

You can see from the class diagram that **Rental** class has aggregation relationship with **Car** and **Customer** classes that means This means that an instance of the **Rental** class contains a pointer to an instance of the **Car** class.



Code Explanation:

Class Car:

```cpp
#ifndef CARS_H
#define CARS_H

#include <string>
using namespace std;

class Car {

    int ID;
    int Year;
    string Model;
    string Make;
    int daily_price;
    bool isAvailable;

public:
    Car(int ID, int Year, string Make, string Model, int daily_price, bool isAvailable) {
        this->ID = ID;
        this->Year = Year;
        this->Model = Model;
        this->Make = Make;
        this->daily_price = daily_price;
        this->isAvailable = isAvailable;
    }
    // Getters
    int get_ID() {
        return this->ID;
    }
    int get_year() {
        return this->Year;
```

Explanation:
The class **"Car"** represents a car object. It has Private member variables such **as ID, Year, Model, Make, daily_price, and isAvailable**.

The class has a constructor to initialize these variables, as well as getter and setter methods for accessing and modifying the data members.

The constructor takes in parameters value of all data member and assign values to the member variables. The getter and setter method in class also added to access or modify data members.

The method in class **printCarDetails()** that prints out the details of a car object, including the ID, Year, Model, Make, daily price, and availability status.

Class Rental:

main.cpp | Customer.h | Cars.h | Rental.h

```cpp
#ifndef RENTAL_H
#define RENTAL_H

#include <string>
#include <iostream>
#include <time.h>
#include "Cars.h"
#include "Customer.h"
using namespace std;

class Rental {
    time_t start_date;
    time_t end_date;
    Car* car;
    Customer* customer;

public:
    Rental(time_t start_date, time_t end_date, string name, int Age,
        string email, int phone_number, int ID, int Year,
        string Model, string Make, int daily_price,bool isAvailable) {

        this->start_date = start_date;
        this->end_date = end_date;
        car = new Car(ID, Year, Model, Make, daily_price, isAvailable);
        customer = new Customer(name, Age, email, phone_number);
    }

    // Getter and Setter for start_date
    time_t getStartDate() const {
        return start_date;
    }
```

Explanation:

The class has private member variables: **start_date and end_date of type time_t**, and pointers to objects of the "**Car**" and "**Customer**" classes.

The class has a constructor that takes parameters to initialize the **start_date, end_date, car**, and customer objects. The constructor **dynamically allocates memory** for the car and customer objects using the provided parameters.

The class also includes getter and setter methods for **start_date, end_date, car, and customer,** allowing access to and modification of these member variables.

The class has a **printRentalDetails()** method, which outputs the rental details, including the start and end dates, customer details, and car details.

Class Customer:

main.cpp Customer.h Cars.h Rental.h

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

class Customer  {

    string name;
    int Age;
    string email;
    int phone_number;


public:
    Customer (string name, int Age, string email, int phone_number) {
        this->name = name;
        this->Age = Age;
        this->email = email;
        this->phone_number = phone_number;
    }
    // Friend Functions
    // Getters

    string get_name() {
        return this->name;
    }

    int get_Age() {
        return this->Age;
    }
```

Explanation:

The class has private member variables: name, Age, email, and phone_number, all of which are of type string or int.

The class has a constructor that takes parameters to initialize the member variables **name, Age, email, and phone_number**. The constructor assigns the parameter values to the corresponding member variables using this pointer.

The class also includes getter methods for accessing the private member variables: **get_name(), get_Age(), get_phone_number(), and get_email().** These methods return the values of the respective member variables.

The class has a **printCustomerDetails()** method, which outputs the customer's details, including the **name, age, email, and phone number**.

Class Car Rental (Main Class):

main.cpp Customer.h Cars.h Rental.h

```
int main() {
    vector<Car*> cars;
    vector<Rental*> rentals;

    // load data from files
    // try catch for exception handling
    try {
        cars = loadCarData("car_data.txt");
        rentals = loadRentalData("rental_data.txt");
    } catch (const exception& e) {
        cerr << "Error: " << e.what() << endl;
        return 1;
    }
    int choice;
    bool exit = false;
    bool found = false;
    Car* selectedCar = nullptr;

    while(!exit){
        // display menu to user
        cout << "\n******************* Car Rental Management System *******************\n";
        cout << "1. Add a new car" << endl;
        cout << "2. Rent a car (also show all cars to select car for rent)" << endl;
        cout << "3. Show all cars" << endl;
        cout << "4. Display rental details" << endl;
        cout << "5. Remove a Car" << endl;
        cout << "6. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
```

Explanation:

This is our main class that provide the user menu to perform operation on **Car Rental Management System**, such as adding a new car, renting a car, displaying car details, displaying rental details, removing a car, exit.

The program will not end until user want to exit the system.

Function in Main Class:

1. The **loadCarData** function loads car data from a text file. It reads the data line by line and creates Car objects using the read data. The cars are then stored in a vector.

2. The **getValidIntegerInput** function prompts the user with a message and accepts an integer input. It validates the input and only allows valid integer values. If the input is not a valid integer, it displays an error message and asks for input again.

3. The **convertStringToTimestamp** function takes a string date in the format "DD:MM:YYYY" and converts it to a timestamp in second.

4. The **loadRentalData** function loads rental data from a file. Similar to **loadCarData,** it reads the data line by line and creates Rental objects using the read data. The rentals are then stored in a vector.

5. The **saveCarData** function saves the car data to a file. It opens the file and writes the car details in the specified format.

6. The **saveRentalData** function saves the rental data to a file. It opens the file and writes the rental details in the specified format.

The main function is the entry point of the program. It initializes vectors for **cars** and **rentals**, and then it loads the existing data from text files using the **loadCarData** and **loadRentalData** functions.

File Management:

Cars Data Loading:

```
// function to load car data from text file
// using dynamic store
vector<Car*> loadCarData(const string& filename) {
    vector<Car*> cars;
    ifstream file(filename);
    if (file.is_open()) {
        int ID, Year, daily_price;
        string Model, Make;
        bool avl;
        while (file >> ID >> Year >> Model >> Make >> daily_price >> avl) {
            Car* car = new Car(ID, Year, Model, Make, daily_price, avl);
            cars.push_back(car);
        }
        cout << "Number of cars added: " << cars.size() << endl;
        file.close();
    } else {
        throw runtime_error("Failed to open text file: " + filename);
    }
    return cars;
}
```

The function **loadCarData**, is responsible for loading car data from a text file. It takes a single parameter, filename, which is the name of the file to load the data from.

Inside the function, it creates an input file stream (ifstream) and opens the specified file. If the file is successfully opened, it enters a loop to read the data line by line. For each line, it extracts the ID, year, model, make, daily price, and availability values using the input operator (>>), and creates a Car object with these values. The newly created Car object is then added to the cars vector which

store the pointers to these objects. After reading all the car data, the file is closed. If the file fails to open, it throws a runtime_error with an appropriate error message.

At end of adding all cars it shows the message of how many cars object is added from file. (Actually, array size).

Cars Data Saving:

```
// function to save car data to file
void saveCarData(const string& filename,vector<Car*>& cars) {
    ofstream file(filename);
    if (file.is_open()) {
        for (Car* car : cars)  {
            file << car->get_ID() << " " << car->get_year() << " " << car->get_Model() << " " << car->get_Make()
            << " " << car->get_daily_price()<< " " << 1 << endl;
        }
        file.close();
        cout<<"\nCars database text file updated success"<<endl;
    } else {
        throw runtime_error("Failed to open car data file for writing");
    }
}
```

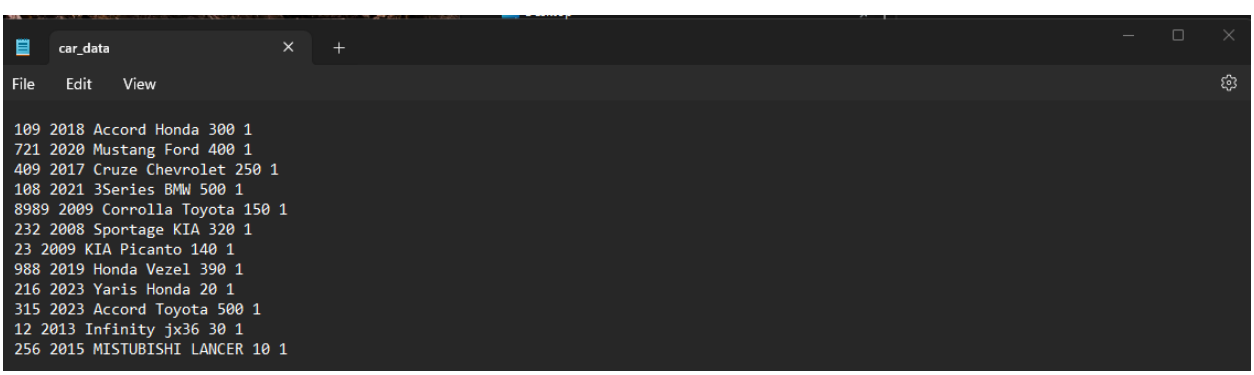
The function **saveCarData**, is responsible for saving car data to a text file. It takes two parameters: filename, which is the name of the file to save the data to, and cars, which is a vector containing the car objects to be saved.

It creates an output file stream (ofstream) and opens the specified file. If the file is successfully opened, it iterates over each car object in the cars vector. For each car, it retrieves its ID, year, model, make, daily price, and availability using getter methods (car.get_ID(), car.get_year(), etc.), and writes these values to the file separated by spaces. Each car's data is written on a new line. After writing all the car data, the file is closed. If the file fails to open, it throws a runtime_error with an appropriate error message.

At end of adding all rental data it shows the message of how many rental objects is added from file. (Actually, array size).

Cars Data Text File:

The text file from where the cars data is read.



Rental Data Text File:

The text file from where the cars data is read.

```
rental_data
File  Edit  View

1631836800 1632355200 cust1 29 c1.@yahoo.com 1111 1 2018 Accord Honda 300 0
1631836800 1632355200 cust2 20 c2.@gmail.com 2222 2 2019 Camry Toyota 350 0
1631836800 1632355200 cust3 36 c3.@yahoo.com 3333 1 2020 Mustang Ford 400 0
1631836800 1632355200 cust4 37 c4.@email.com 4444 1 2017 Cruze Chevrolet 250 0
1631836800 1632355200 cust5 43 c5.@gmail.com 5555 1 2021 4Series BMW 500 0
1662922800 1660158000 c7 39 aa@gmail.com 878787 8989 2009 Corrolla Toyota 150 0
1694458800 1691780400 CustomerX 34 123@email.com 23232 211 2019 Toyota Camry 350 0
1694458800 1697050800 HHHH 23 a@gmail.com 656767 211 2019 Toyota Camry 350 0
1694804400 1697050800 dddd 34 a@yahho.com 8786 988 2019 Honda Vezel 390 0
1697756400 1697842800 Maged 19 m@gmail.com 365183 211 2019 Toyota Camry 350 0
1710889200 1710975600 Maged 19 maged/2gmail.com 609080 12 2013 jx36 Infinity 30 0
1674169200 1708470000 Maged 19 magedhosni@gmail.com 6709073 315 2023 Toyota Accord 500 0
1320966000 1355266800 MAGED 20 Maged@gmail.com 5203604 256 2015 MISTUBISHI LANCER 10 0
```

Exception Handling and Data Validation:

The function **getValidIntegerInput** is used to obtain a valid integer input from the user. It takes a string parameter msg which is a message or prompt to be displayed to the user before input.

The user enters a value of it's a valid integer the loop end and it returns the number and if not than it shows message that value is not valid and prompt user to enter again until get the valid number.

```
//take valid integer put from user
//the loop will not ternimate until user enter valid input
int getValidIntegerInput(const string& msg) {
    int number;
    while (true) {
        cout << msg;
        if (cin >> number) {
            // Input is a valid integer
            break;
        } else {
            // Input is not a valid integer
            cout << "Invalid input. Please enter a valid integer" << endl;
            cin.clear();  // clear error flags
            // discarding the input buffer
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
    }
    return number;
}
```

We use **try catch** to catch exception every where we are loading or saving data into the files.

```
// load data from files
// try catch for exception handling
try {
    cars = loadCarData("car_data.txt");
    rentals = loadRentalData("rental_data.txt");
} catch (const exception& e) {
    cerr << "Error: " << e.what() << endl;
    return 1;
}
```

Future Enhancements:

1. Adding GUI (Graphical User Interface) to the Application:
2. Adding an actual database like (SQL) to manage and store data instead of text files.
3. Adding more features like Customer Verification or Car inspection.

Conclusion:

The Car Rental Management System developed in this project provides an effective solution for managing car rentals. By automating key processes such as adding cars, renting cars, and generating reports, the system streamlines the rental process and improves efficiency. The system incorporates validation checks to ensure data integrity and provides a user-friendly interface for easy interaction.

In conclusion, the Car Rental Management System presented in this project offers a demo solution for managing car rentals efficiently. It simplifies the rental process, improves data accuracy, and enhances overall productivity. With further development and enhancements, the system can be customized and expanded to meet the specific needs of car rental businesses

References:

https://www.geeksforgeeks.org/file-handling-c-classes/

https://www.w3schools.com/cpp/cpp_exceptions.asp

https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm

https://www.softwaretestinghelp.com/date-and-time-in-cpp/

https://www.tutorialspoint.com/cplusplus/cpp_date_time.htm

https://www.javatpoint.com/university-management-system-in-cpp