Car Rental System

Introduction

The Car Rental System project is designed to manage a fleet of rental cars. The system provides functionalities to add, delete, rent, return, and search for cars. It also includes a customer management module to handle customer details and memberships. This documentation provides an overview of the system, detailing its interface, execution, input/output formats, structure, testing, and areas for improvement.

Program Interface

Activation

To run the program, compile the main.cpp file along with cars.h and source.cpp:

```
g++ -o car_rental main.cpp source.cpp
```

```
./car_rental
```

Termination

The program can be terminated by selecting the exit option (7) from the main menu.

Program Execution

Main Menu

Upon execution, the user is presented with the main menu offering several options:

1. **Add a Car**: Add a new car to the system.
2. **Delete a Car**: Remove a car from the system.
3. **View All Cars**: Display details of all cars in the system.
4. **Search for a Car**: Search for cars by brand or manufacture year.
5. **Rent a Car**: Rent a car from the system.
6. **Return a Car**: Return a rented car to the system.
7. **Exit**: Exit the program.

Adding a Car

The user is prompted to enter the car's brand, model, year, registration number, and mileage. The system then adds the car to the fleet.

Deleting a Car

The user is prompted to enter the car model. If the car exists, it is removed from the system.

Viewing All Cars

The system displays a list of all cars, showing details like brand, model, year, registration number, mileage, and availability.

Searching for a Car

The user can search by brand or manufacture year. The system displays matching cars with their details.

Renting a Car

The user is prompted to enter the car model. If available, the car is marked as rented and the rental start time is recorded.

Returning a Car

The user is prompted to enter the car model. If the car is currently rented, it is marked as returned, and the rental fee is calculated based on the duration.

Input and Output

Input

Inputs are primarily taken through console prompts for car details and menu options. Car details include brand, model, year, registration number, and mileage.

Output

Outputs are displayed on the console, including lists of cars, search results, and rental/return confirmations.

File Input/Output

The system reads from and writes to `cars_data.txt` to persist car information. The file format is: brand,model,regNo,year,mileage,availability

Example: BMW,I3,1234,2020,15000,1

Toyota,Corolla,5678,2018,30000,0

Audi,A4,6070,2018,29000,1

Nissan,Altima,1220,2015,50000,0

Program Structure

Overall Structure

The program consists of three main classes:

- **Car**: Represents a car with attributes like brand, model, year, registration number, mileage, and availability.
- **Customer**: Represents a customer with attributes like name, ID, license number, address, email, and membership status.
- **System**: Manages the collection of cars and provides functionalities to add, delete, rent, return, and search for cars.

Car Class

Attributes:

○ brand: The brand of the car (string). ○ model: The model of the car (string). ○ year: The manufacture year of the car (int). ○ registration number: The registration number of the car (string). ○ mileage: The mileage of the car (int). ○ availability: Availability status of the car (bool)

Methods:

○ Car(string brand, string model, int year, string regNo, int mileage, bool availability): Constructor to initialize a car object. ○ displayCarDetails() const: Displays details of the car.

Customer Class

Attributes:

○ name: Name of the customer (string). ○ ID: Unique identifier for the customer (string). ○ license number: Driver's license number (string).
○ address: Address of the customer (string). ○ email: Email of the customer (string). ○ membership status: Membership status of the customer (bool).

Methods:

- Customer(string name, string ID, string license, string address, string email, bool membership): Constructor to initialize a customer object. ○ displayCustomerDetails() const: Displays details of the customer.

System Class

Attributes:

- vector<Car*> cars: Stores the list of cars.
- **map<string, time_point<high_resolution_clock>> rental_start_times**: Tracks rental start times for fee calculation.

Methods:

- addCar(Car* car): Adds a car to the system. ○ deleteCar(const string& model): Deletes a car from the system. ○ displayCars() const: Displays all cars. ○ searchByBrand(const string& brand) const: Searches cars by brand. ○ searchByYear(int year) const: Searches cars by year. ○ rentCar(const string& model): Rents a car. ○ returnCar(const string& model): Returns a car. ○ saveCars(const string& filename) const: Saves car data to a file.
- getFileCars(const string& filename): Loads car data from a file.

Key Data Structures

- vector<Car*> cars: Stores the list of cars.
- **map<string, time_point<high_resolution_clock>> rental_start_times**: Tracks rental start times for fee calculation.

Major Functions

- addCar*(*Car* car): Adds a car to the system.
- **deleteCar(const string& model)**: Deletes a car from the system.
- **displaycars() const**: Displays all cars.
- **searchBybrand(string& brand) const**: Searches cars by brand.
- **searchByYear(int& year) const**: Searches cars by year.
- **rentCar(const string& model)**: Rents a car.
- **returnCar(const string& model)**: Returns a car.
- **savecars(const string& filename) const**: Saves car data to a file.
- **getFileCars(const string& filename)**: Loads car data from a file.

Flowcharts and Control Flow Graphs

To illustrate the control flow and implementation logic of important functions and processes, the system utilizes both flowcharts and control flow graphs. These visual tools are essential for understanding the intricate workings of the car rental system, including key operations such as car searching, renting, and returning.

Flowcharts

- Start

- Open File "cars_data.txt"

- Load Cars from File into System

- Display Menu

- User Selects Option

+---+

| Option 1: Add a Car |

| - Call create_car() |

| - Add car to System |

| - Save cars to file |

+---+

| Option 2: Delete a Car |

| - Prompt for car model |

| - Delete car from System |

| - Save cars to file |

+---+

| Option 3: View All Cars |

| - Display all cars in System |

+---+

| Option 4: Search for a Car |

| - Prompt for search criteria (brand/year) |

| - Search and display results |

+--+

| Option 5: Rent a Car |

| - Prompt for car model |

| - Rent car if available |

| - Save cars to file |

+--+

| Option 6: Return a Car |

| - Prompt for car model |

| - Return car and calculate rental fees |

| - Save cars to file |

+--+

| Option 7: Exit |

| - Exit the program |

+--+

-Loop until Option 7 is selected

- End

Testing and Verification

Testing Process

The program was tested through manual inputs, ensuring each functionality worked as expected. Various edge cases were tested, such as invalid inputs, renting already rented cars, and returning cars that were not rented.

Verification

Verification involved checking the correct updates of car availability, accurate rental fee calculations, and proper file handling for data persistence.

Improvements and Extensions

Possible Improvements

- Implementing more robust error handling.

- Enhancing the customer management module with more features.
- Adding categories for cars with different rental rates.

Future Extensions

- Introducing a graphical user interface (GUI).
- Adding online booking and payment integration.
- Implementing a more detailed rental fee structure based on car categories and rental duration.

Difficulties Encountered

Challenges

- Managing dynamic memory allocation and ensuring no memory leaks.
- Implementing accurate time-based rental fee calculation.
- Handling user input validation robustly.

Solutions

- Careful use of pointers and thorough testing to avoid memory leaks.
- Using the chrono library for precise time tracking.
- Implementing try-catch blocks and input validation loops for robust user input handling.

Conclusion

The Car Rental System effectively manages car rentals, providing functionalities for adding, deleting, renting, returning, and searching cars. Despite challenges, the project was successfully implemented with potential areas identified for future enhancements. This documentation serves as a comprehensive guide for understanding and using the system.

References

Object-Oriented Programming in C++ (4th Edition) by Robert Lafore.

Savitch, W. (2017). Absolute C++.

Eckel, B. (2014). Thinking in C++ (Volume 1 & Volume 2).