**PROJECT REPORT**

**CL1002 Programming Fundamentals**

**Muneeb Uddin 25k-0765**



FAST NUCES, Karachi

Department of Computer Science

## Abstract

The car rental management system is a console-based program developed in C language to store, process and display records of cars and customers which are renting cars. It allows the user to enter Brand, Model, and rent per day of car and the customer details. The system computes the rent for customers who book cars for finite number of days and displays the invoice in a formatted manner. This project applies fundamental programming concepts such as decision structures (if … else, switch ... case...), loops (Counter and semantic loops), arrays to store records while the program is running, use of functions to promote modular programming and files to store the records after the program is terminated. It demonstrated how simple data handling can replace manual calculations and reduce errors in managing the rental records.

## 1. Introduction

Managing rental records manually is time-consuming and error prone. Companies often need a simple digital solution to calculate results quickly and accurately without duplication of data. This project provides a basic computerized system where users can enter details for multiple cars and customers and instantly generate connections between them by customers renting cars. It also validates the data entered so there’s no chance of duplicate rent of same car. It strengthens understanding of variables, conditionals, loops, file handling, arrays and modular programming techniques.

## 2. Objectives

* Add new cars into the system
* Display all cars
* Register customers
* Rent a car to a customer
* Return a car and calculate total rent
* Search for a car or customer using their ID
* Keep data stored in text files
* Show revenue collected from rentals

## 3. System Design

### System Overview

Flow of the program:

* Start
* Load data from files
* Show menu
* Perform selected operation
* Update data
* Save and exit

### Algorithm

1. Start the program
2. Load car, customer, and rental data from text files
3. Show the main menu
4. Based on user choice, perform the required task
5. Validate input (car availability, customer ID, etc.)
6. Update the data in memory
7. Write updated records back to files
8. Keep repeating the menu until the user selects Exit

### Input & Output

1. Input:

* Car details (ID, Brand, Company)
* Customer information (Customer ID, Name, Address, Phone Number)
* Number of days for rental

1. Output:

* Car list
* Customer List
* Rent Calculations
* Revenue report

## 4. Implementation

* Tools Used:
* Language Used: C
* Compiler Used: GCC
* IDE Used: VS-Code and Dev C++

### Key Features

* Adding and displaying cars
* Customer Registration
* Unique customer ID system
* Multiple simultaneous rentals per customer
* Renting and Returning Cars
* Preventing double renting of Same Car
* Bill Calculation
* Revenue Tracking
* Today's actual earnings (from returned cars)
* Expected earnings (from active rentals)
* Saving all Data to files
* Sorted car list by ID before saving
* Modular structure using .c and .h files.
* Simple recursive functions for searching
* Three separate data files for organization

Project Breakdown:

The Core Programming concepts used:

1. **Structures:** Car, Customer, Rental data structures
2. **Arrays:** Static global arrays for data storage
3. **File Handling:** Save/load data persistence
4. **Recursion:** Search, sort, and calculation functions
5. **Functions:** Modular programming approach
6. **Loops:** while, do-while, for loops
7. **Conditional Statements:** if-else, switch-case
8. **Preprocessor Directives:** #define, #include

**The main menu**:  
1. Add New Car

2. Display All Cars

3. Register New Customer

4. Rent a Car

5. Return a Car

6. Search Car by ID

7. Display All Customers

8. Revenue Report

9. Exit

Enter Number:

**Code Snippet:**

1. For Searching Car for checking availibility of car in the file:
2. int searchCar(int car\_id)
3. {
4. int i = 0;
5. while(i < car\_count)
6. {
7. if(cars[i].id == car\_id)
8. return i;
9. i++;
10. }
11. return -1;
12. }
13. Finding the customer id through indirect recursion:

int findCustomer(int cid, int index)

{

    if(index >= customer\_count)

        return -1;

    if(customers[index].id == cid)

        return index;

    return findCustomer(cid, index + 1);

}

int findCustomerById(int customer\_id) {

    return findCustomer(customer\_id, 0);

}

1. The commonly used features to clear buffer and pause screen for whole program:

static inline void clearBuffer()

{

    int c;

    while ((c = getchar()) != '\n' && c != EOF);

}

static inline void pauseScreen()

{

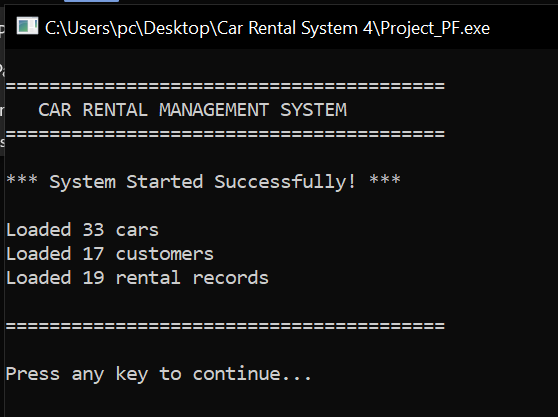
    printf("\nPress any key to continue...");

    getch();

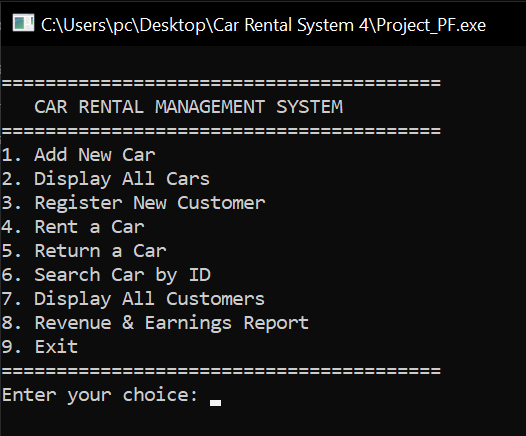
}

### Sample Output

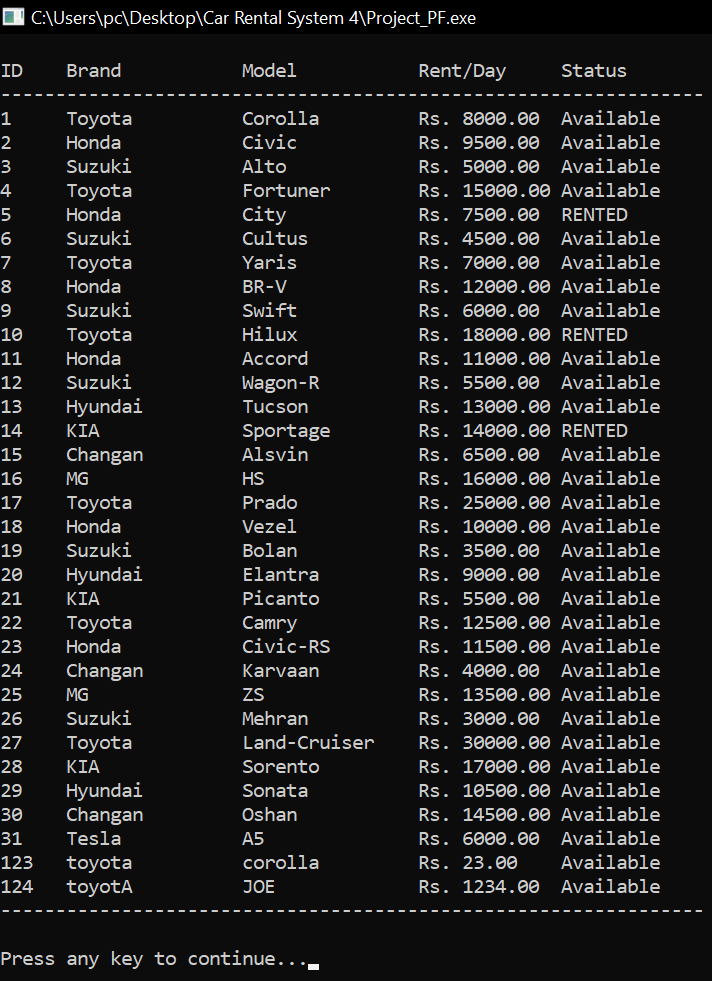
1. Program Starts



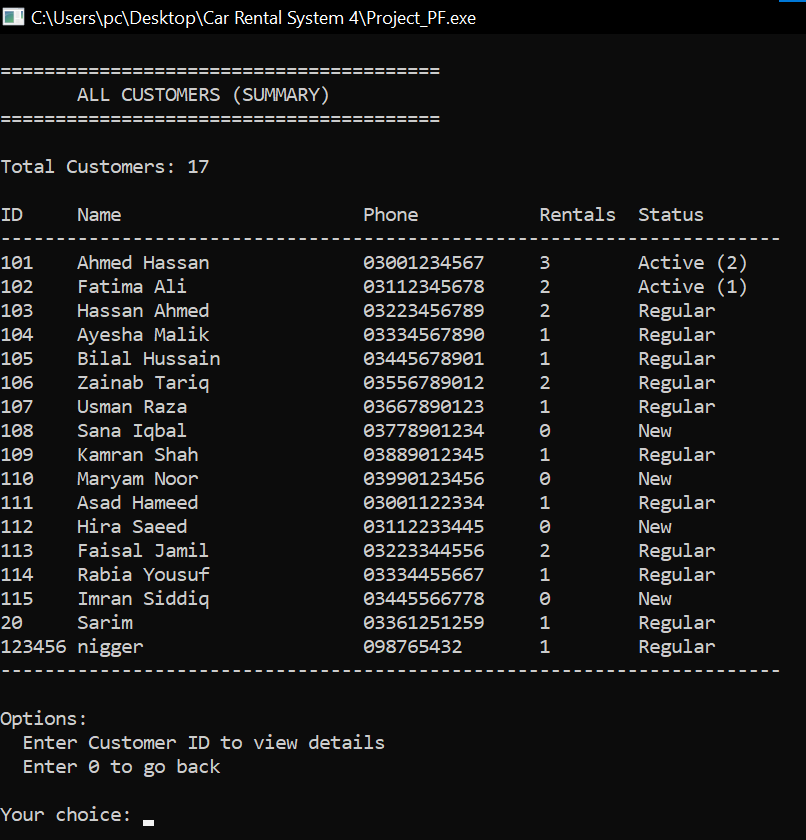
1. Main Menu



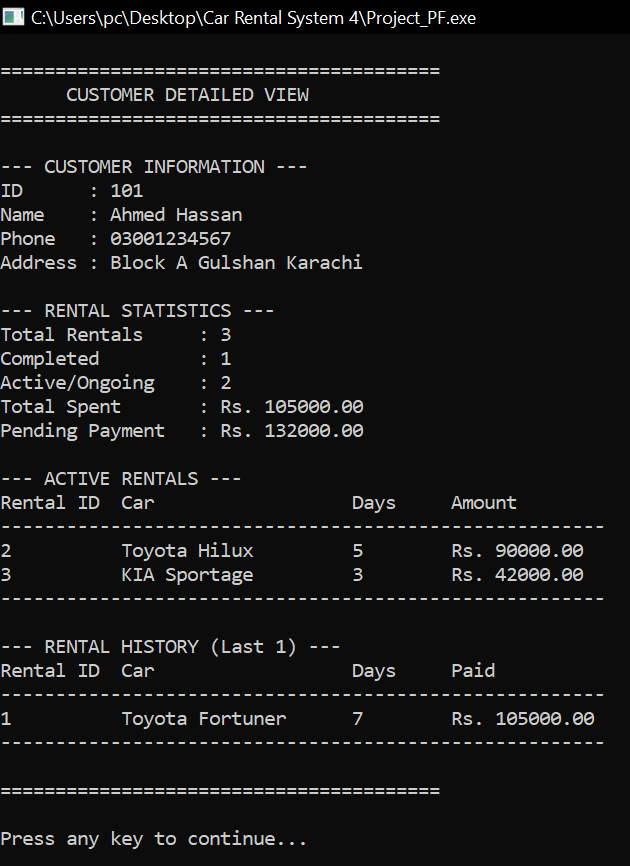
1. Case 2: Display Cars



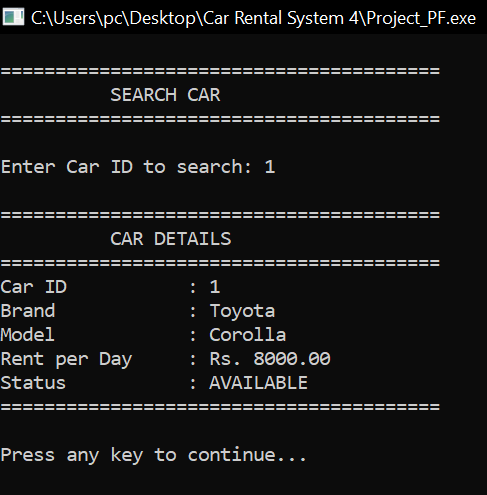
1. Case 7: Customer View



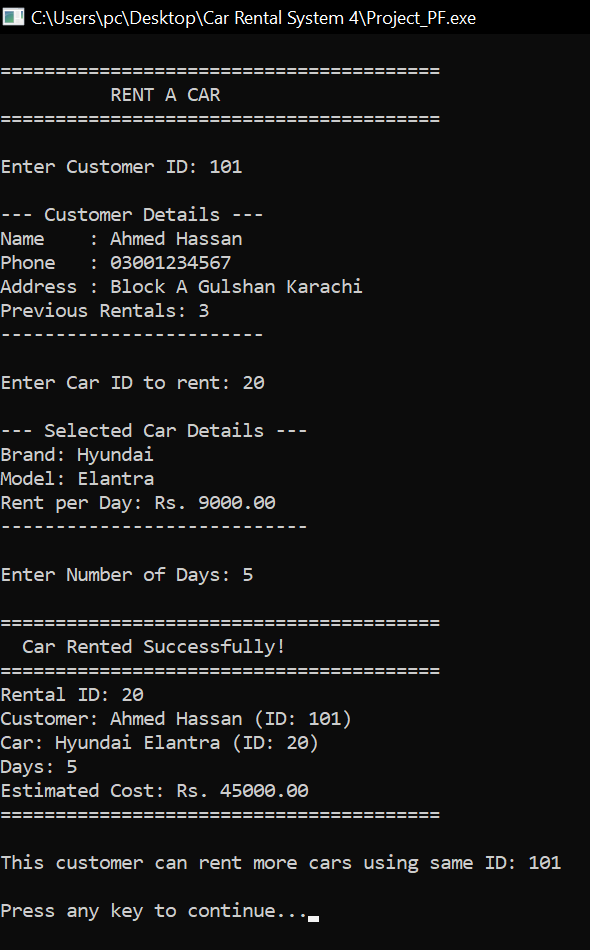
1. 101: Detailed View Customer



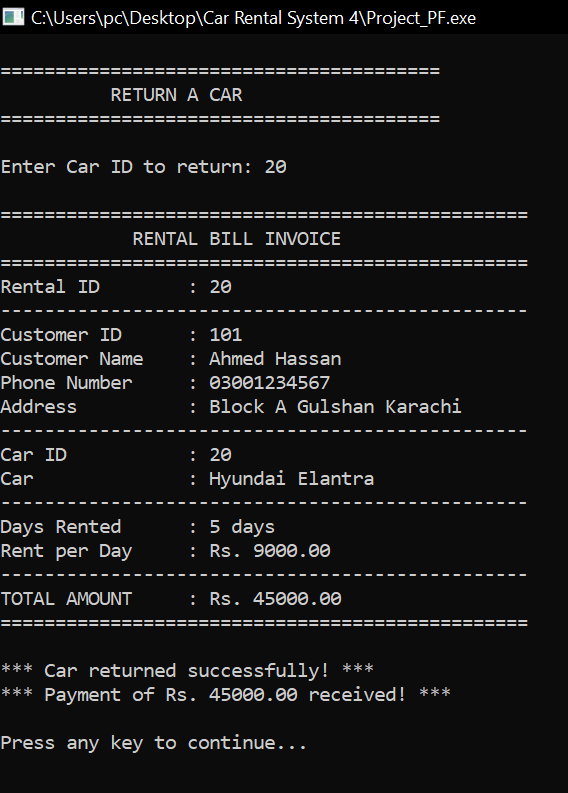
1. Search Car



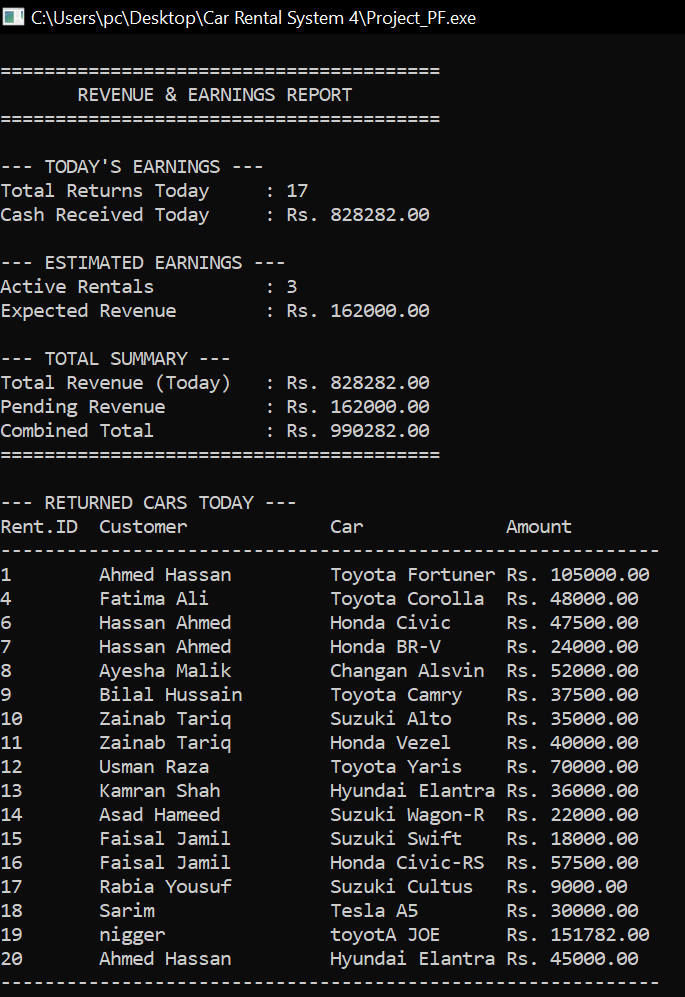
1. Rent a Car



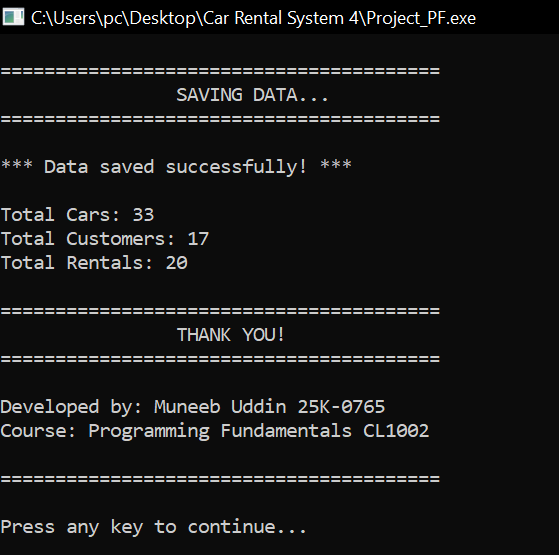
1. Return a Car



1. Revenue Report



1. Program Exit



## 5. Testing & Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **What I Did** | **What Should Happen** | **What Actually Happened** | **Result** |
| 1 | Added car with ID 5 | Car gets added | Car got added | Pass |
| 2 | Tried adding ID 5 again | Error message | Showed error | Pass |
| 3 | Rented an available car | Rental works | Rental worked | Pass |
| 4 | Tried renting same car | Error message | Showed error | Pass |
| 5 | Returned a car | Bill shows up | Bill displayed correctly | Pass |
| 6 | Registered new customer | Customer saved | Customer got saved | Pass |
| 7 | Same customer rents 3 cars | All tracked separately | All 3 tracked | Pass |
| 8 | Checked revenue report | Correct totals | Numbers were right | Pass |
| 9 | Closed and opened program | Data still there | Data loaded fine | Pass |
| 10 | Searched for fake car ID | Not found message | Showed not found | Pass |

The program performed successfully for all test cases. It handled both high and low marks efficiently, produced accurate totals and grades, and validated all inputs. Execution speed was near-instant, and the program required minimal system resources.

**Problems I Faced During Programming**

Problem 1: Input Buffer Issue

After using scanf, when I used another scanf right after, it would skip asking for input. This was super annoying.

How I fixed it: Made a function called clearBuffer() that removes leftover characters after scanf.

Problem 2: Double Renting

Initially, I could rent the same car to two people. This was a big bug.

How I fixed it: Added a check - before renting, system looks at is\_rented status. If it's 1, show error.

Problem 3: Spaces in Names

When I saved names with spaces to file, loading them back was messed up. Like "Ahmed Hassan" would only load "Ahmed".

How I fixed it: Replace spaces with underscores when saving (Ahmed\_Hassan), then replace back to spaces when loading.

Problem 4: Finding Customers

At first, tracking which customer rented which car was confusing.

How I fixed it: Made a third file called rentals.txt that connects customers to cars using their IDs.

## 6. Conclusion, Limitations & References

### Conclusion

### What I Achieved:

### The system does what it's supposed to do - manage car rentals without manual work. It's not perfect but it works reliably. I'm pretty happy with how it turned out considering this is my first real project.

### What I Learned:

### Honestly, I learned a lot:

### How to split a big problem into small pieces

### Why saving data in files is important

### How to debug when things go wrong (spent hours on this)

### Why checking user input matters

### How real programs are structured with multiple files

### The file handling part was the hardest. I had to redo it like 3 times before it worked properly.

### Limitations:

### Only handles 50 cars and 50 customers (I used fixed arrays)

### No proper interface, just text on black screen

### Doesn't track dates, so you can't tell when car was rented

### No late fees if someone returns car late

### Can't search by car brand or model, only ID

### No password, anyone can use it

### Future Enhancements

* Dynamic memory allocation (after First Semester)
* Date/time tracking
* Late fee calculation
* Customer loyalty points
* Advanced search filters (After 3rd Semester)
* Database integration (After 5th semester)

### References

* Let Us C by Yashavant P. Kanetkar
* https://www.programiz.com/c-programming
* <https://www.geeksforgeeks.org/c-programming-language/>
* <https://stackoverflow.com>
* Youtube Channel: Apna College
* Teacher’s Support.
* My Classmates such as Sarim, Zohair, Usman etc helps me a lot too.