PROJECT REPORT

Parking Lot Management

Course: Programming in C (CSEG1041)

Submitted To: Mohsin F. Dar

Submitted By: Arpita Sinha

SAP ID: 590025329

University: University of Petroleum and Energy Studies

Date: November 30, 2025

TABLE OF CONTENTS

Abstract

The "Parking Management System" is a console-based software application developed in the C programming language. Vehicle management is a very important issue in today's urban environment. Maintaining records of parking with traditional physical registers is inefficient, prone to human error, and highly time-consuming. This project digitizes the procedure for accuracy and efficiency.

The main goal of this project is to provide an efficient interface for parking lot attendants to manage the entry and exit of vehicles. The system is developed on the basics of C programming, namely Structures (struct), which are used for storing complex data about vehicles; Arrays, used to handle multiple instances of vehicles; and Modular Programming, which allows separating the logic into different files (.c and .h).

Key Features of the System:

Vehicle Entry: This form captures information on the vehicle number, owner's name, assigned slot, and entry time.

Slot Validation: The system automatically checks whether a certain parking slot is already taken or not in order to avoid double-booking.

Exit & Billing: Based on entry and exit times, calculate the parking duration and compute the parking fee automatically (Rs. 20/hour).

Search Functionality: Allows the user to search for any vehicle by number plate or name of the owner.

History Log: Records all the vehicles which have exited to provide a full audit trail.

This project describes the application of the C programming language in solving a real-life logistical problem. It will form the basis for even more complex systems that could later involve database integration or automation based on sensors.

PROBLEM DEFINITION

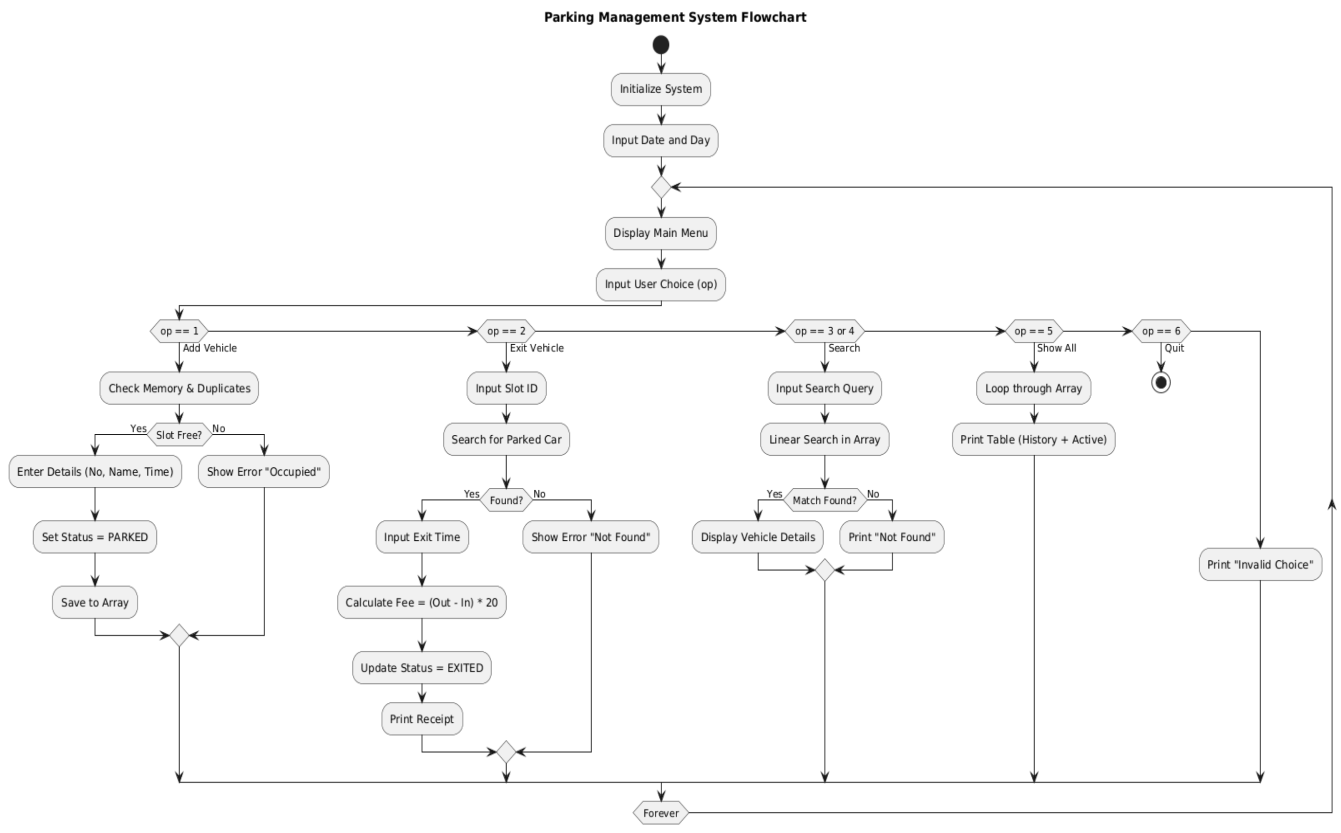
Introduction to the Problem Managing a parking lot manually using physical registers is inefficient and prone to errors. Security guards often struggle to track which slots are empty in real-time, leading to confusion and traffic inside the lot. Furthermore, calculating parking fees manually based on entry and exit times often results in mathematical errors and potential revenue loss. Finding a specific vehicle's record in a thick physical notebook is also extremely time-consuming.

Proposed Solution: **The Parking Management System automates these tasks to solve the issues mentioned above**:

- **Accurate Billing:** The system automatically calculates the duration (Exit Time - Entry Time) and the total fee, removing the possibility of human math errors.
- **Real-Time Slot Tracking:** The program checks the status of every slot before assigning it. If a slot is occupied, it prevents double-booking, ensuring efficient space management.
- **Instant Search:** The search feature allows attendants to instantly retrieve details like the owner's name and slot number just by entering the vehicle plate number.
- **Digital Organization**: By storing data in a structured digital format, the system ensures records are neat, organized, and easy to read compared to handwriting in a register.

SYSTEM DESIGN

Flowcharts



ALGORITHMS

ALGORITHM 1: MAIN SYSTEM FLOW

1. Start the program and ask for Date/Day.
2. Show Menu: Display options (Add, Exit, Search, Show, Quit).
3. Input: Ask user to choose an option number.
4. Decision:
   - If 1: Go to *Add Vehicle*.
   - If 2: Go to *Exit Vehicle*.
   - If 3/4: Go to *Search Vehicle*.
   - If 5: Go to *Show List*.
   - If 6: Stop the program.
5. Repeat: Go back to Step 2 (Loop forever until Quit).

Algorithm 2: Add Vehicle

1. Check Space: If parking is full (100 cars), stop.
2. Input: Ask for Vehicle Number.
3. Check Duplicate: If this vehicle is already inside, stop.
4. Input: Ask for Name and Slot Number.
5. Check Slot: If the slot is already taken, stop.
6. Save: Store all details and mark status as "PARKED".
7. Success: Print confirmation message.

Algorithm 3: Exit Vehicle

1. Input: Ask for the Slot Number to empty.
2. Find: Search for the car currently parked in that slot.
3. Input: Ask for Exit Time.
4. Calculate:

    - Duration = Exit Time - Entry Time.
    - (If negative, add 24 hours).
5. Bill: Fee = Duration * Rs 20.
6. Update: Change status to "EXITED".
7. Print: Display the bill receipt.

Algorithm 4: Search Vehicle

1. Input: Ask user for Vehicle Number or Owner Name.
2. Search: Look through the entire list of records.
3. Result:
    - If found: Print Slot, Time, and Status (Parked/Exited).
    - If not found: Print "Record does not exist."

Algorithm 5: Show History

1. Check: If list is empty, print "No records".
2. Display: Print a table header.
3. Loop: Go through every car recorded.
    - Show Number, Name, Slot, and Time.
    - Show if they are currently "PARKED" or "EXITED".

IMPLEMENTATION DETAILS

Modular Programming Approach: In order to have organized, readable, and debuggable code, I employed a Modular Programming approach. Instead of having all the code in one file, I broke the project down into three specific files, each with a specific responsibility:

Include File - parking.h: The include file works like a "blueprint" for the system. It defines what a struct Car is, and declares prototypes of all the functions used within this project. Now multiple .c files can use the same structure definitions.

src/parking.c: This is where the actual coding and logic of the system go. All the heavy lifting, such as finding free slots, finding time differences, and updating vehicle status, goes here.

Main File (src/main.c): Entry point of the program; it takes care of the user interface/Menu, and - depending on the choice of the user - calls the corresponding functions implemented in the file parking.c.

KEY CODE SNIPPETS

Structure Definition The struct Car is the backbone of the system and encapsulates all data that belongs to a single car into one unit

Code-

```
struct Car {
    char no[20];        // Vehicle Plate Number
    char name[30];      // Owner Name
    int slot_id;        // Parking Slot Number
    int t_in;           // Entry Time
    int t_out;          // Exit Time
    char type;          // 'P' for Parked, 'E' for Exited
};
```

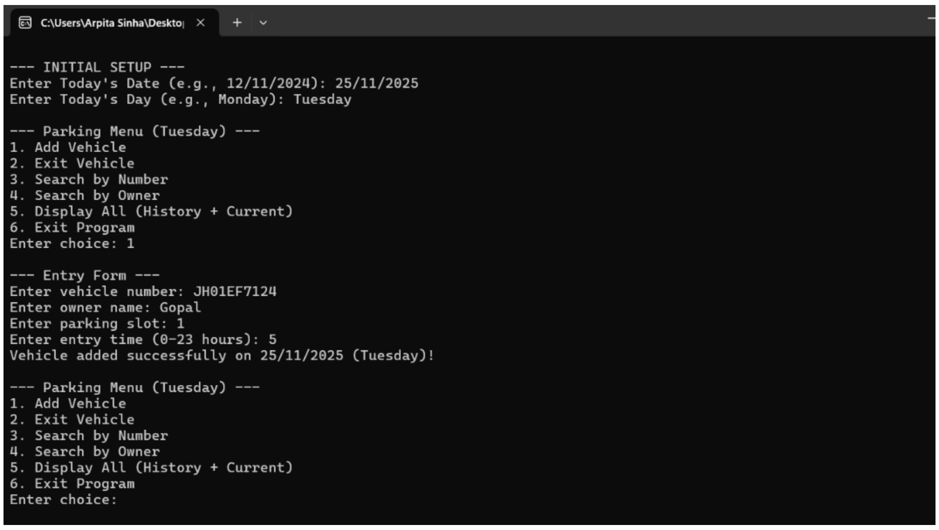
Slot Validation Logics: The most important feature is to make sure no two cars get the same slot. This function checks in the array before confirming any entry.

Code-

```
int check_slot(int s) {
    for(int i = 0; i < cnt; i++) {
        // If slot matches AND status is 'P' (Parked), it's
busy
        if(data[i].slot_id == s && data[i].type == 'P') {
            return 0; // Slot is occupied
        }
    }
    return 1; // Slot is free
}
```

TESTING & RESULTS

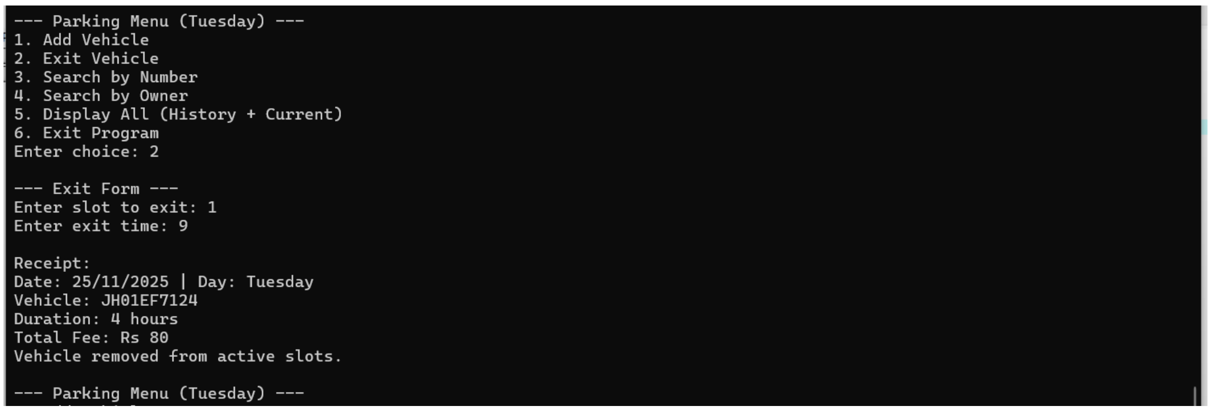
Test Case 1: Vehicle entry



```
--- INITIAL SETUP ---
Enter Today's Date (e.g., 12/11/2024): 25/11/2025
Enter Today's Day (e.g., Monday): Tuesday

--- Parking Menu (Tuesday) ---
1. Add Vehicle
2. Exit Vehicle
3. Search by Number
4. Search by Owner
5. Display All (History + Current)
6. Exit Program
Enter choice: 1

--- Entry Form ---
Enter vehicle number: JH01EF7124
Enter owner name: Gopal
Enter parking slot: 1
Enter entry time (0-23 hours): 5
Vehicle added successfully on 25/11/2025 (Tuesday)!

--- Parking Menu (Tuesday) ---
1. Add Vehicle
2. Exit Vehicle
3. Search by Number
4. Search by Owner
5. Display All (History + Current)
6. Exit Program
Enter choice:
```

Fig 1: Successful entry of a vehicle. The system accepted the Vehicle Number, Name, Slot, and Time, and confirmed the storage.

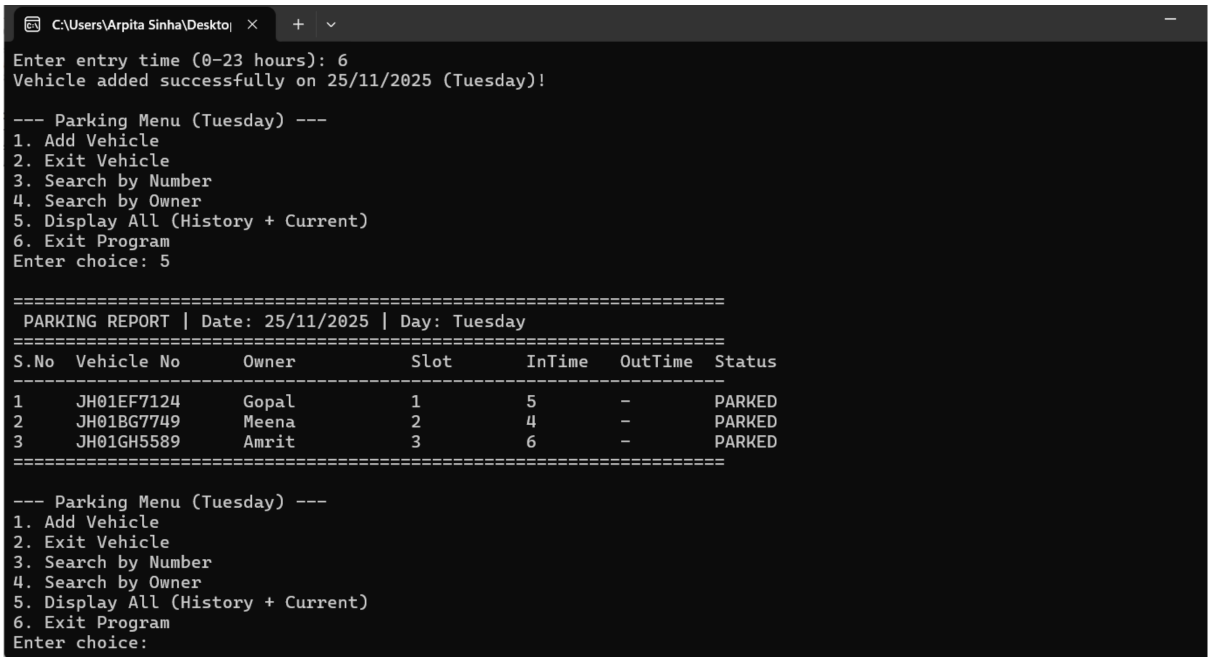
Test Case 2: Vehicle Exit & Billing



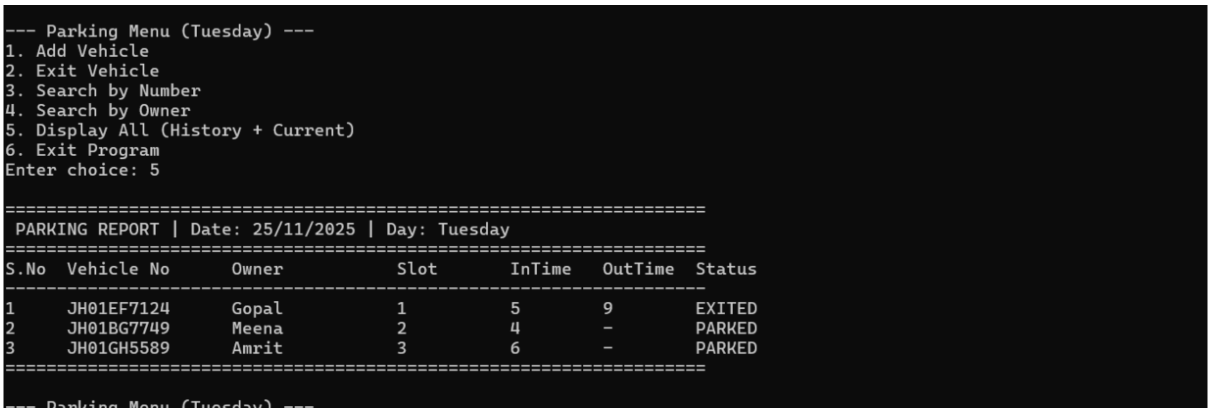
```
--- Parking Menu (Tuesday) ---
1. Add Vehicle
2. Exit Vehicle
3. Search by Number
4. Search by Owner
5. Display All (History + Current)
6. Exit Program
Enter choice: 2

--- Exit Form ---
Enter slot to exit: 1
Enter exit time: 9

Receipt:
Date: 25/11/2025 | Day: Tuesday
Vehicle: JH01EF7124
Duration: 4 hours
Total Fee: Rs 80
Vehicle removed from active slots.

--- Parking Menu (Tuesday) ---
```

Fig 2: A vehicle checking out. The user entered Slot 1, and the system retrieved the entry details and duration automatically, generating a bill of Rs. 80.

Test Case 3: **Parking History**



```
Enter entry time (0-23 hours): 6
Vehicle added successfully on 25/11/2025 (Tuesday)!

--- Parking Menu (Tuesday) ---
1. Add Vehicle
2. Exit Vehicle
3. Search by Number
4. Search by Owner
5. Display All (History + Current)
6. Exit Program
Enter choice: 5

====================================================================
 PARKING REPORT | Date: 25/11/2025 | Day: Tuesday
====================================================================
S.No  Vehicle No      Owner           Slot       InTime   OutTime  Status
--------------------------------------------------------------------
1     JH01EF7124      Gopal           1          5        -        PARKED
2     JH01BG7749      Meena           2          4        -        PARKED
3     JH01GH5589      Amrit           3          6        -        PARKED
====================================================================

--- Parking Menu (Tuesday) ---
1. Add Vehicle
2. Exit Vehicle
3. Search by Number
4. Search by Owner
5. Display All (History + Current)
6. Exit Program
Enter choice:
```



```
--- Parking Menu (Tuesday) ---
1. Add Vehicle
2. Exit Vehicle
3. Search by Number
4. Search by Owner
5. Display All (History + Current)
6. Exit Program
Enter choice: 5

====================================================================
 PARKING REPORT | Date: 25/11/2025 | Day: Tuesday
====================================================================
S.No  Vehicle No      Owner           Slot       InTime   OutTime  Status
--------------------------------------------------------------------
1     JH01EF7124      Gopal           1          5        9        EXITED
2     JH01BG7749      Meena           2          4        -        PARKED
3     JH01GH5589      Amrit           3          6        -        PARKED
====================================================================

--- Parking Menu (Tuesday) ---
```

Fig 3: The System Report showing the status of all vehicles. Note that the vehicle which checked out is marked as "EXITED", while others remain "PARKED".

CONCLUSION & FUTURE WORK

Conclusion

The "Parking Management System" successfully digitizes the manual process of vehicle tracking. In completing this project, I have achieved the following objectives:

Error Reduction: Manual calculations of parking fees were eliminated.

Efficiency: Minimized the time in finding the owners of a vehicle or an empty slot.

Organization: Designed a system to keep a structured history of all transactions.

The project runs efficiently without crashing. It handles invalid inputs, such as trying to park in a full slot, and it is a great example of how C concepts like Structures, Arrays, and Modular file handling are used.

Future Scope

Though the current system works perfectly, there is always scope for further improvement. In the future version of this application, I intend to add:

File Handling: Currently, data is stored in RAM and is lost when the program closes. I'll implement file I/O (fprintf / fscanf) to save records permanently in a text file.

Authentication: Adding a login system to ensure only authorized security personnel access the menu.

GUI stands for Graphical User Interface. Moving away from that black console screen and shifting towards a window-based application with buttons and colours.

REFERENCES

1. Textbooks:

Let Us C by Yashavant Kanetkar (for basic logic and syntax).

The C Programming Language by Kernighan & Ritchie (for standard library functions).

2. Online Resources:

GeeksforGeeks: Referenced for understanding Linear Search algorithms.

TutorialsPoint: Referenced for struct and modular programming concepts.

Stack Overflow: Used for troubleshooting compiler errors - undefined reference to `WinMain`.

3. Course Materials:

UPES C Programming Lecture Notes

APPENDIX

File 1: Header File (include/parking.h) *This file contains the structure definition and function prototypes*

```c
#ifndef PARKING_H
#define PARKING_H

// Structure to store car details
struct Car {
    char no[20];
    char name[30];
    int slot_id;
    int t_in;      // Entry time
    int t_out;     // Exit time
    char type;     // 'P' = Parked, 'E' = Exited
};

// Global variables (We use 'extern' to tell the main file they exist)
extern struct Car data[100];
extern int cnt;
extern char d_date[20], d_day[20];

// Function Prototypes (List of features)
int check_slot(int s);
void new_entry();
void checkout();
void find_car();
void show_list();

#endif
```

File 2: Logic Implementation (src/parking.c) *This file contains the core logic for adding, removing, and searching vehicles*.

```
#include <stdio.h>

#include <string.h>

#include "../include/parking.h" // Links to the header file


// Global variable definitions

struct Car data[100];

int cnt = 0;

char d_date[20], d_day[20];


// Function to check if a specific slot is empty

int check_slot(int s) {

    int i;

    for(i = 0; i < cnt; i++) {

        // If slot matches AND status is 'P' (Parked), it's busy

        if(data[i].slot_id == s && data[i].type == 'P') {

            return 0;

        }

    }

    return 1; // Slot is free

}


void new_entry() {

    if(cnt >= 100) {

        printf("Memory Full.\n");

        return;

    }


    struct Car temp;
```

```c
    int i;

    printf("\n--- Entry Form ---\n");
    printf("Enter vehicle number: ");
    scanf("%s", temp.no);

    // Check if this car is already inside (Duplicate check)
    for(i = 0; i < cnt; i++) {
        if(strcmp(data[i].no, temp.no) == 0 && data[i].type == 'P') {
            printf("Error: This vehicle is already inside the parking
lot!\n");
            return;
        }
    }

    printf("Enter owner name: ");
    scanf(" %[^\n]s", temp.name); // Reads name with spaces

    printf("Enter parking slot: ");
    scanf("%d", &temp.slot_id);

    // Validate if slot is free
    if(check_slot(temp.slot_id) == 0) {
        printf("Slot %d is currently occupied!\n", temp.slot_id);
        return;
    }

    printf("Enter entry time (0-23 hours): ");
    scanf("%d", &temp.t_in);
    temp.type = 'P'; // Set status to Parked
    temp.t_out = 0;

    data[cnt] = temp; // Save to array
    cnt++;

    printf("Vehicle added successfully on %s (%s)!\n", d_date, d_day);
}

void checkout() {
    int s, t, i, found = -1;

    printf("\n--- Exit Form ---\n");
    printf("Enter slot to exit: ");
    scanf("%d", &s);

    // Find the car in this slot that is currently parked
    for(i = 0; i < cnt; i++) {
        if(data[i].slot_id == s && data[i].type == 'P') {
            found = i;
            break;
        }
    }

    if(found == -1) {
        printf("No active vehicle found in slot %d!\n", s);
        return;
    }

    printf("Enter exit time: ");
    scanf("%d", &t);

    // Calculate duration
    int dur = t - data[found].t_in;
    if(dur <= 0) dur = dur + 24; // Fix for overnight parking

    // Calculate fee (Min 20 Rs)
    int amt = dur * 20;
    if(amt < 20) amt = 20;

    // Update status to 'E' (Exited) instead of deleting
    data[found].type = 'E';
    data[found].t_out = t;

    printf("\nReceipt:\n");
    printf("Date: %s | Day: %s\n", d_date, d_day);
    printf("Vehicle: %s\n", data[found].no);
    printf("Duration: %d hours\n", dur);
    printf("Total Fee: Rs %d\n", amt);
    printf("Vehicle removed from active slots.\n");
}

// Combined search function for Number and Owner
void find_car() {
    int choice, i, flag = 0;
    char query[30];

    printf("\nSearch by: 1.Number 2.Owner\nOption: ");
    scanf("%d", &choice);
    printf("Enter Value: ");
    scanf(" %[^\n]s", query);

    for(i = 0; i < cnt; i++) {
        int match = 0;
        // Check based on user choice
        if(choice == 1 && strcmp(data[i].no, query) == 0) match = 1;
        if(choice == 2 && strcmp(data[i].name, query) == 0) match = 1;

        if(match) {
            char *statStr = (data[i].type == 'P') ? "PARKED" : "EXITED";
            printf("\nFound (%s):\nNumber: %s | Slot: %d | In: %d",
                   statStr, data[i].no, data[i].slot_id, data[i].t_in);

             if(data[i].type == 'E') printf(" | Out: %d\n", data[i].t_out);
             else printf("\n");

            flag = 1;
        }
    }
    if(flag == 0) printf("Not found in records!\n");
}

void show_list() {
    int i;
    if(cnt == 0) {
        printf("No records found!\n");
        return;
    }
```

```
printf("\n==================================================================
=\n");
    printf(" PARKING REPORT | Date: %s | Day: %s\n", d_date, d_day);

printf("===================================================================\
n");
    printf("%-5s %-15s %-15s %-10s %-8s %-8s %-10s\n",
           "S.No", "Vehicle No", "Owner", "Slot", "InTime", "OutTime",
"Status");
    printf("-----------------------------------------------------------------
---\n");



    for(i = 0; i < cnt; i++) {
        printf("%-5d %-15s %-15s %-10d %-8d ",
               i+1, data[i].no, data[i].name, data[i].slot_id, data[i].t_in);

        // Display Exit time only if car has left
        if(data[i].type == 'E') {
            printf("%-8d %-10s\n", data[i].t_out, "EXITED");
        } else {
            printf("%-8s %-10s\n", "-", "PARKED");
        }
    }

printf("===================================================================\
n");
}
```

File 3: Main Menu (src/main.c) *This file contains the user interface and menu loop.*

```
#include <stdio.h>
```

```
#include "../include/parking.h" // Links to the header file

int main() {
    int op;

    // --- STEP 1: Ask for Date and Day initially ---
    printf("\n--- INITIAL SETUP ---\n");
    printf("Enter Today's Date (e.g., 12/11/2024): ");
    scanf("%s", d_date);
    printf("Enter Today's Day (e.g., Monday): ");
    scanf("%s", d_day);

    while(1) {
        // Menu exactly matches the image
        printf("\n--- Parking Menu (%s) ---\n", d_day);
        printf("1. Add Vehicle\n");
        printf("2. Exit Vehicle\n");
        printf("3. Search by Number\n");
        printf("4. Search by Owner\n");
        printf("5. Display All (History + Current)\n");
        printf("6. Exit Program\n");
        printf("Enter choice: ");
        scanf("%d", &op);

        if(op == 1) new_entry();
        else if(op == 2) checkout();
        else if(op == 3) find_car();
        else if(op == 4) find_car();
        else if(op == 5) show_list();
        else if(op == 6) return 0;
        else printf("Invalid option!\n");
    }
}
```