East West University

Department of Computer Science and Engineering

Aftabnagar, Dhaka 1212

A Report on the Mini Project on the " **VEHICLE MANAGEMENT SYSTEM**"

Course Title: Data Structures

Course Code: CSE207

Section: 7

Date of Submission: 16/01/2025

Submitted By:

Tasnia Taher Esha

Student ID: 2023-1-60-173

Abdul Wadud Priyo

Student ID: 2022-2-60-133

Ohiduzzaman Rayhan

Student ID: 2023-1-60-108

Submitted To:

Dr. Md. Atiqur Rahman

Assistant Professor

Department of Computer Science and Engineering

East West University

Table of Contents

Abstract

The **"Smart Vehicle Parking System"** project aims to develop an efficient and user-friendly platform to manage parking facilities in urban areas or institutions. This system is designed to streamline the parking process, providing users with seamless access to parking spaces and administrators with tools for effective space utilization and management. Key features of the system include real-time parking space availability updates, automated entry/exit processes, reservation options, and secure payment integration.

The project employs advanced data structures to handle and optimize the large volume of data associated with vehicles and parking slots. Using data structures such as hash maps and priority queues ensures fast data retrieval, optimal allocation of parking spaces, and real-time updates. Through this system, users can save time, avoid congestion, and access parking details effortlessly, while administrators benefit from enhanced operational efficiency and revenue tracking.

The implementation of the **Smart Vehicle Parking System** is expected to improve the parking experience by reducing wait times, optimizing space usage, and fostering better organization and communication in busy parking facilities

Dedication

“This project is dedicated to all those working toward innovative solutions for urban challenges. We extend our gratitude to our mentors, peers, and families for their unwavering support and encouragement.

Lastly, we dedicate this work to users and administrators, with the hope that it contributes to a more efficient and organized parking experience.”

Acknowledgments :

We would like to express our heartfelt gratitude to everyone who contributed to the successful completion of the **Vehicle Parking System** project.

First and foremost, we extend our sincere thanks to our course instructor, Dr. Md. Atiqur Rahman, for his expert guidance, valuable insights, and continuous encouragement throughout the development process. His constructive feedback and unwavering support played a crucial role in shaping this project

We also acknowledge the efforts of our peers who assisted in testing the system and provided valuable feedback, helping us enhance its functionality and user experience.

A special thanks to our families and friends for their constant support, patience, and encouragement, which kept us motivated during challenging times.

Finally, we express our gratitude to everyone who, directly or indirectly, contributed to the successful completion of this project. Your support and contributions have been instrumental in bringing this idea to life.

Vehicle Parking System

Introduction:

The **Vehicle Parking System** is a console-based C program designed to manage parking operations efficiently. This system allows users to add, remove, search for, and display parked vehicles. The program uses a linked list data structure to store and manage vehicle information dynamically, ensuring efficient data management and retrieval. Each vehicle record includes details such as type, model, number, and entry time, making it easy to track and manage parking operations.

The system is user-friendly, offering a menu-driven interface for seamless interaction. It provides a structured approach to handle parking management, ensuring accuracy and data integrity while simplifying operational tasks.

Specific Objects:

- **Efficient Data Management**.
- **User-Friendly Interface**.
- **Vehicle Addition**.
- **Vehicle Removal**.
- **Vehicle Search**.
- **Display Parked Vehicles**.
- **Data Integrity and Validation**.

Program Overview and Introduction:

- **Header and Library Inclusions:** The code includes necessary header files such as stdio.h, stdlib.h, string.h.
- **Data Structures:**
- **Vehicle Structure**: Stores information about each vehicle, including:
- Type: The type of vehicle (e.g., car, bike).
- Model: The model of the vehicle.
- Number: The unique vehicle number.
- EntryTime: The time of vehicle entry.
- Next: A pointer to the next vehicle in the linked list.

- **Functions**:
  - Functions are: CreateVehicle(), addVehicle(), displayVehicles(), removeVehicle(), searchVehicle() are declared.

• Functions Definition:

- **createVehicle()**:
  Creates a new vehicle node and initializes its fields.

- **addVehicle()**:
  Adds a new vehicle to the linked list, either as the head (if the list is empty) or at the end of the list.

- **displayVehicles()**:
  Displays the list of all currently parked vehicles, including their type, model, number, and entry time.

- **removeVehicle()**:
  Removes a vehicle from the linked list based on the provided vehicle number.

- **searchVehicle()**:
  Searches for a vehicle in the linked list by its number and displays its details if found.

- **`searchVehicle()`**:
  Searches for a vehicle in the linked list by its number and displays its details if found.

.

Complexity analysis:

The overall time complexity of the **Vehicle Parking System** depends on the specific operations performed and the use of a linked list for managing vehicle data. Below is the detailed complexity analysis of the key operations:

Adding a Vehicle

- Traversing the linked list to find the last node takes **O(n)**, where **n** is the number of vehicles in the list.
- Adding the new node at the end of the list takes constant time **O(1)**.
- **Overall time complexity: O(n)**.

Displaying Parked Vehicles

- Iterating through the entire linked list to display vehicle information takes **O(n)**, where **n** is the number of vehicles in the list.
- **Overall time complexity: O(n)**.

Removing a Vehicle

- Traversing the linked list to find the vehicle with the specified number takes **O(n)**, where **n** is the number of vehicles in the list.
- Removing the node (if found) takes constant time **O(1)**.
- **Overall time complexity: O(n)**.

Searching for a Vehicle

- Traversing the linked list to find the vehicle with the specified number takes **O(n)**, where **n** is the number of vehicles in the list.
- **Overall time complexity: O(n)**.

Driver Function (`main`)

- The driver function uses a menu-driven approach and calls one of the above functions based on user input.
- The time complexity of the **`main`** function depends on the selected operation but is dominated by the function with the highest time complexity, which is **O(n)**.

Summary of Complexities for Key Operations

Operation	Time Complexity
Adding a Vehicle	O(n)
Displaying Vehicles	O(n)
Removing a Vehicle	O(n)
Searching for a Vehicle	O(n)

The overall complexity of the program is determined by the specific operations invoked during its execution. Since all key operations involve traversing the linked list, the program's complexity is **O(n)** in most cases.

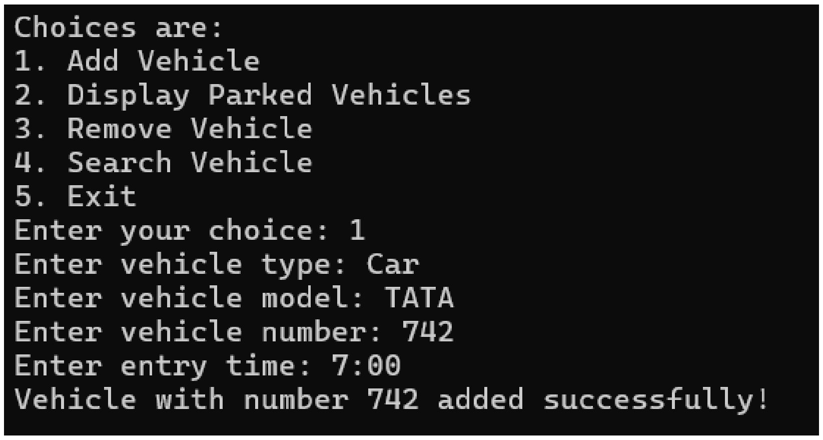
Code Snapshots:

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


struct Vehicle {
    char type[20];
    char model[20];
    char number[20];
    char entryTime[20];
    struct Vehicle* next;
};


struct Vehicle* createVehicle(char* type, char* model, char* number, char* entryTime) {
    struct Vehicle* newVehicle = (struct Vehicle*)malloc(sizeof(struct Vehicle));
    strcpy(newVehicle->type, type);
    strcpy(newVehicle->model, model);
    strcpy(newVehicle->number, number);
    strcpy(newVehicle->entryTime, entryTime);
    newVehicle->next = NULL;
    return newVehicle;
}


void addVehicle(struct Vehicle** head, char* type, char* model, char* number, char* entryTime) {
    struct Vehicle* newVehicle = createVehicle(type, model, number, entryTime);
    if (*head == NULL) {
        *head = newVehicle;
    } else {
        struct Vehicle* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newVehicle;
    }
    printf("Vehicle with number %s added successfully!\n", number);
}


void displayVehicles(struct Vehicle* head) {
    if (head == NULL) {
        printf("No vehicles parked.\n");
        return;
    }

    printf("\nParked Vehicles:\n");
    printf("Type\t\tModel\t\tNumber\t\tEntryTime\n");
    printf("-------------------------------------------------------------\n");

    struct Vehicle* temp = head;
    while (temp != NULL) {
        printf("%s\t\t%s\t\t%s\t\t%s\n", temp->type, temp->model, temp->number, temp->entryTime);
        temp = temp->next;
    }
}


void removeVehicle(struct Vehicle** head, char* number) {
    if (*head == NULL) {
        printf("No vehicles parked.\n");
        return;
    }

    struct Vehicle* temp = *head;
    struct Vehicle* prev = NULL;

    while (temp != NULL && strcmp(temp->number, number) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Vehicle with number %s not found.\n", number);
        return;
    }

    if (prev == NULL) {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }

    free(temp);
    printf("Vehicle with number %s removed successfully!\n", number);
```

```
}


void searchVehicle(struct Vehicle* head, char* number) {
    if (head == NULL) {
        printf("No vehicles parked.\n");
        return;
    }

    struct Vehicle* temp = head;
    while (temp != NULL) {
        if (strcmp(temp->number, number) == 0) {
            printf("\nVehicle Found:\n");
            printf("Type: %s\n", temp->type);
            printf("Model: %s\n", temp->model);
            printf("Number: %s\n", temp->number);
            printf("Entry Time: %s\n", temp->entryTime);
            return;
        }
        temp = temp->next;
    }

    printf("Vehicle with number %s not found.\n", number);
}

int main() {
    struct Vehicle* head = NULL;
    int choice;
    char type[20], model[20], number[20], entryTime[20];


    printf("\t\t\t\t<Welcome to Vehicle Parking System>\n");

    while (choice != 5){
        printf("\nChoices are:\n");
        printf("1. Add Vehicle\n");
        printf("2. Display Parked Vehicles\n");
        printf("3. Remove Vehicle\n");
        printf("4. Search Vehicle\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Consume newline character
        switch (choice) {
            case 1:
                printf("Enter vehicle type: ");
                scanf("%s", type);
                printf("Enter vehicle model: ");
                scanf("%s", model);
                printf("Enter vehicle number: ");
                scanf("%s", number);
                printf("Enter entry time: ");
                scanf("%s", entryTime);
                addVehicle(&head, type, model, number, entryTime);
                break;

            case 2:
                displayVehicles(head);
                break;

            case 3:
                printf("Enter vehicle number to remove: ");
                scanf("%s", number);
                removeVehicle(&head, number);
                break;

            case 4:
                printf("Enter vehicle number to search: ");
                scanf("%s", number);
                searchVehicle(head, number);
                break;

            case 5:
                printf("Exiting the system.\n");
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

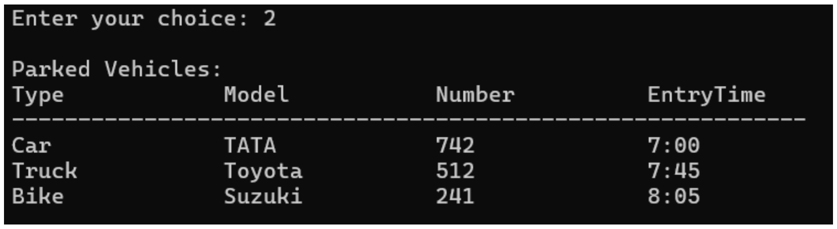
Output snapshot:

Enter choice 1 to add vehicle's information;



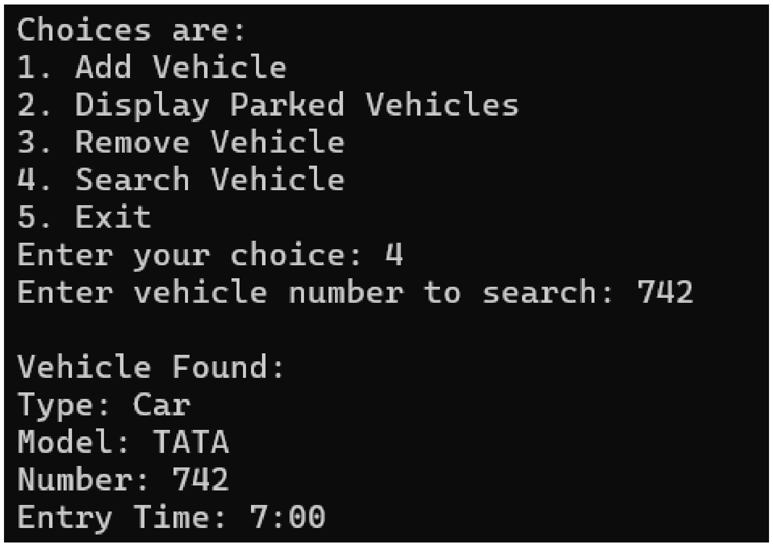
```
Choices are:
1. Add Vehicle
2. Display Parked Vehicles
3. Remove Vehicle
4. Search Vehicle
5. Exit
Enter your choice: 1
Enter vehicle type: Car
Enter vehicle model: TATA
Enter vehicle number: 742
Enter entry time: 7:00
Vehicle with number 742 added successfully!
```

Enter choice 2 to Display Parked Vehicles;



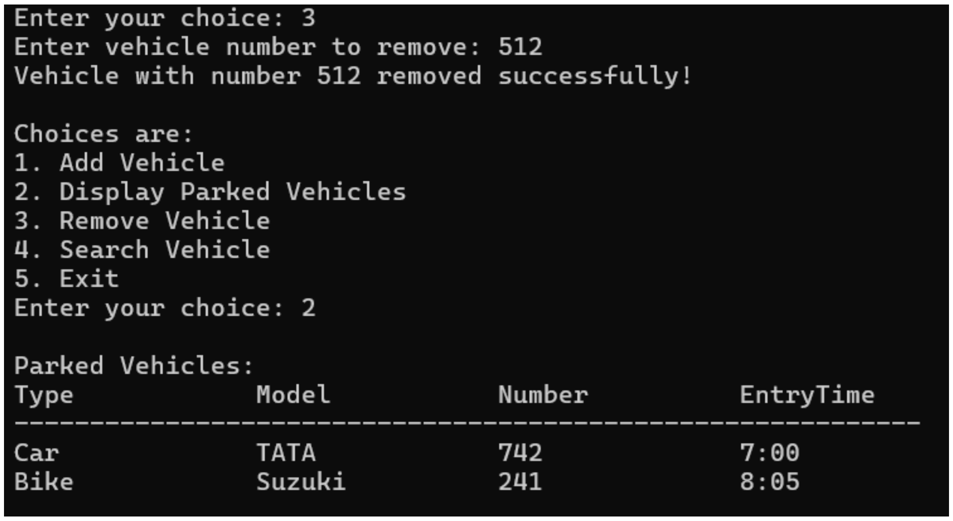
```
Enter your choice: 2

Parked Vehicles:
Type            Model           Number          EntryTime
------------------------------------------------------------
Car             TATA            742             7:00
Truck           Toyota          512             7:45
Bike            Suzuki          241             8:05
```

Enter choice 4 to search vehicle by giving vehicle number;



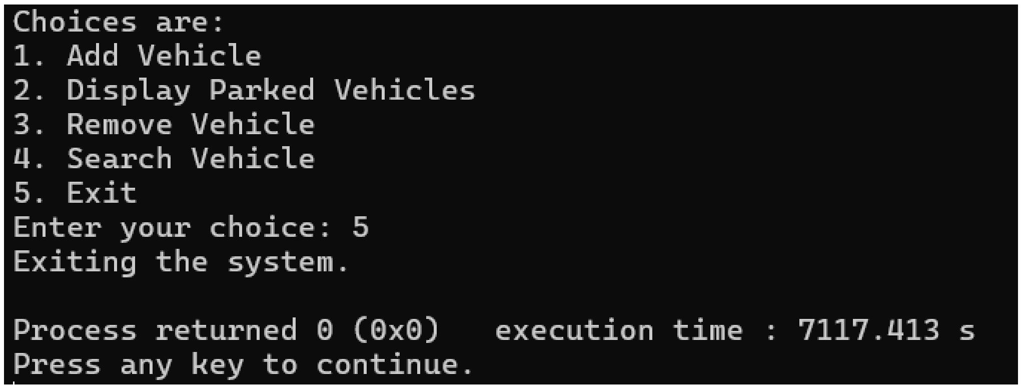
```
Choices are:
1. Add Vehicle
2. Display Parked Vehicles
3. Remove Vehicle
4. Search Vehicle
5. Exit
Enter your choice: 4
Enter vehicle number to search: 742

Vehicle Found:
Type: Car
Model: TATA
Number: 742
Entry Time: 7:00
```

Enter choice 3 remove vehicle by giving vehicle number;



```
Enter your choice: 3
Enter vehicle number to remove: 512
Vehicle with number 512 removed successfully!

Choices are:
1. Add Vehicle
2. Display Parked Vehicles
3. Remove Vehicle
4. Search Vehicle
5. Exit
Enter your choice: 2

Parked Vehicles:
Type            Model           Number          EntryTime
------------------------------------------------------------
Car             TATA            742             7:00
Bike            Suzuki          241             8:05
```

Lastly, enter choice 5 and successfully exit.



```
Choices are:
1. Add Vehicle
2. Display Parked Vehicles
3. Remove Vehicle
4. Search Vehicle
5. Exit
Enter your choice: 5
Exiting the system.

Process returned 0 (0x0)   execution time : 7117.413 s
Press any key to continue.
```

Conclusion:

The **Vehicle Parking System** efficiently manages vehicle data using a linked list, supporting operations like adding, displaying, removing, and searching vehicles. Each operation is designed with a time complexity of **O(n)** due to the traversal of the list, where **n** is the number of vehicles. The program's modular design ensures flexibility and clarity, while its menu-driven approach enhances user interaction. Overall, the system offers a simple yet effective solution for vehicle parking management.