PROJECT REPORT

ON

PARKING MANAGEMENT SYSTEM

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

SUBMITTED BY :-

NAME	:	Aishwarya Verma(22051051)
		Kinshuk (22051081)
		Rishav Kashyap (23057032)
DEPARTMENT	:	Computer Science & Engineering
SESSION	:	2022 -2026
COLLEGE	:	KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOY, BHUBANESWAR ,ODISHA.

PARKING MANAGEMENT SYSTEM

Abstract

The Parking Management System is a C application to automate vehicle parking operation in small- to medium-sized facilities. The application provides an efficient allocation of parking slots, calculates fees for parking based on time duration, and stores information constantly while providing features for searching for vehicles with registration numbers. The system brings more accuracy and reduces errors with manual efforts due to the simplification of managing parking. It can very well be used in malls and office buildings as it is stable, user-friendly, and expandable features like automated billing and GUI.

Introduction

Parking management in an urban facility is something that has been required nowadays where vehicle inflow as well as outflow needed to be systematized. The current project describes a Parking Management System in which the information of all the vehicles could be entertained, and their parking space can be allotted, which can also compute the charges based on the parking hours. Thus, this could help the system reduce various kinds of errors caused by people.

Objectives:

Major objectives of this project are to :-

Automate Slot Allocation : It automatically assigns a parking slot with the data.

Data Storage : Save and retrieve data from a vehicle from a file.

Charges to be calculated on the number of hours of parking.

Search Car : Allow users to search for parked vehicles by their registration number.

Requirement Analysis :

Functional Requirements

1) **Park Car :** Enter the details of a vehicle and provide a slot number along with the time for parking.

2) **Show Cars :** Print a list of all the currently parked vehicles along with the parking charge.

3) **Search Car :** Find a vehicle using its registration number along with information

4) **Exit :** The program will be permitted to safely exit and save data

Non-Functional Requirements

1) **Usability :** User interface is very basic and console-based.

2) **Reliability :** Data should be stored and retrieved in error-free ways for session continuation.

3) **Performance :** The system could support up to 100 vehicles with efficient processing.

4) **Security :** The access is restricted to authorized personnel.

System Requirements

1) Hardware Requirements

   i. **Processor :** At least 1 GHz or faster, 32-bit or 64-bit processor.

   ii. **Memory (RAM) :** At least 512 MB; 1 GB or more recommended.

   iii. **Storage :** 50 MB of available space for program files and data storage.

   iv. **Display :** Standard display with a minimum resolution of 800x600.

   v. **Input Device :** Keyboard and optional mouse for CLI interaction.

2) **Software Requirements**

   i. **Operating System :** Compatible with Windows, macOS, or Linux.

   ii. **Compiler :** To use a C compiler, for example GCC, to compile and run the code.

   iii. **Text Editor :** Any text editor (for instance, Visual Studio Code, Notepad++, Vim) where you will be editing your code files

   iv. **File System :** Read and write access to files used for persistent data storage (for example, parking.txt: keeping data about the vehicles)

3) Add Extra Libraries

Time Library used as needed in calculating the parking duration with the help of time.h; generally most environments of the C environment have it.

Module Description

PMS is designed as a modular one, therefore the functions of each of these functions in the program only perform its specific job. The modular approach clarifies the concept, maintenance easy, and room for any upgrades.

- Load Data Module

Function: LoadData(struct Vehicle vehicles[])

It loads data for the vehicle from a file called parking.txt so that there may not be inconsistency in sessions between programs.

Process : Reads vehicle information such as vehicle number, slot number, and parked time from the file and stores in an array.

Output : Initializes the array with data that has been saved in advance or starts with a null list if no data is found.

Code :-

```
int LoadData( struct Vehicle vehicles[])
{
  FILE *file = fopen("parking.txt", "r");
  if (file == NULL)
  {
    printf("Error opening file. Starting with an empty parking lot.\n");
    return 0;
  }

  int count = 0;
  while (fscanf(file, "%s %d %ld", vehicles[count].vehicleNo, &vehicles[count].slotNo, &vehicles[count].parkedTime) != EOF)
  {
    count++;
  }
  fclose(file);
  return count;
}
```

- Save Data Module

Function : SaveData(struct Vehicle vehicles[], int count)

Saves the list of parked vehicles currently stored in memory to a file for permanent storage.

Process : Writes vehicle details-vehicle id, slot number and elapsed time to parking.txt it overwrites the already written data.

Output : Always returns successful status which proves that the input data is stored to the appropriate place

Code :-

```
int SaveData(struct Vehicle vehicles[], int count)

{

  FILE *file = fopen("parking.txt", "w");

  if (file == NULL)

  {

    printf("Error opening file for saving.\n");

    return 0;

  }

  for (int i = 0; i < count; i++)

  {

    fprintf(file, "Vehicle No. :%s , Slot No. :%d , Fee :%ld\n", vehicles[i].vehicleNo,
vehicles[i].slotNo, vehicles[i].parkedTime);

  }

  fclose(file);

  return 1;

}
```

- Slot Allotment Module

Function : AllocateSlot(struct Vehicle cars[], int size)

Return the first free slot for new

Process : Iterates in the loop from index to MAX_VEHICLES of the given array until slot is free

Output : Returns slot number. Else returned -1

Code :-

```
int AllocateSlot(struct Vehicle vehicles[], int count)
{
  for (int i = 0; i < MAX_VEHICLES; i++)
  {
    int slotAvailable = 1;
    for (int j = 0; j < count; j++)
    {
      if (vehicles[j].slotNo == i)
      {
        slotAvailable = 0;
        break;
      }
    }
    if (slotAvailable)
    {
      return i;
    }
  }
  return -1;
}
```

- Park Module

Function: ParkVehicle(struct Vehicle vehicles[], int *count)

It parks a new vehicle, gives a slot, and notes the current time.

Process : It takes the user input for the number of the vehicle to be parked, gives a slot, records the time, saves the data, and increments the count of vehicles.

Output : It prints the slot number assigned or an error message if the parking is full.

Code :-

```
void ParkVehicle(struct Vehicle vehicles[], int *count)
{
    if (*count >= MAX_VEHICLES)
    {
        printf("**** PARKING IS FULL ****\n");
        return;
    }
    struct Vehicle newVehicle;
    printf("Enter vehicle number: ");
    scanf("%s", newVehicle.vehicleNo);

    newVehicle.slotNo = AllocateSlot(vehicles, *count);
    if (newVehicle.slotNo == -1)
    {
        printf("No available slots!\n");
        return;
    }

    newVehicle.parkedTime = time(NULL);
    vehicles[*count] = newVehicle;
    (*count)++;
    if (SaveData(vehicles, *count))
    {
        printf("Vehicle parked at slot number: %d\n", newVehicle.slotNo);
    }
}
```

- <u>Calculate Fee Module</u>

Function: CalculateFee(struct Vehicle vehicle)

It calculates the fee for parking based on how long the vehicle has been in the parking lot.

Procedure : Calculate hours by taking subtracting the parked time with the current time, multiplying to FEE_PER_HOUR

Output : It will print fee and hours parked.

Code :-

```
void CalculateFee(struct Vehicle vehicle)
{
   time_t currentTime = time(NULL);
   double hoursParked = (double)(currentTime - vehicle.parkedTime) / 3600;
   double fee = hoursParked * FEE_PER_HOUR;
   printf("Vehicle No: %s, Parked for: %.2f hours, Fee: Rupees %.2f\n", vehicle.vehicleNo, hoursParked, fee);
}
```

- Show Parked Vehicles Module

Function : DisplayVehicles (struct Vehicle vehicles [],int count)

To show the parked details of the vehicles including Slot number along with their Fee

Procedure : Pass array of the parked vehicle list. In each of it, the procedure is performing a cycle over each entry and print fee details on screen

Output : This will be a list in paper where it contains vehicle number including fees and hours.

Code :-

```
void DisplayVehicles(struct Vehicle vehicles[], int count)
{
   if (count == 0)
   {
      printf("\nNo vehicles are parked currently.\n");
      return;
   }

   printf("\nParked Vehicles:\n");
   for (int i = 0; i < count; i++)
   {
      printf("Slot No: %d\n", vehicles[i].slotNo);
```

```
        CalculateFee(vehicles[i]);
    }
}
```

- Search Vehicle Module

Function: SearchVehicle(struct Vehicle vehicles[], int count)

Searches for and prints details of a vehicle given its registration number.

Process : Search through the vehicle array to match with the input vehicle number.

Output : Prints slot number, duration of parking and fee if vehicle found otherwise prints "Vehicle not found".

Code :-

```
void SearchVehicle(struct Vehicle vehicles[], int count)
{
    char searchNo[20];
    printf("Enter vehicle number to search: ");
    scanf("%s", searchNo);

    int found = 0;
    for (int i = 0; i < count; i++)
    {
        if (strcmp(vehicles[i].vehicleNo, searchNo) == 0) {
            printf("Vehicle found!\n");
            printf("Vehicle No: %s, Slot No: %d\n", vehicles[i].vehicleNo, vehicles[i].slotNo);
            CalculateFee(vehicles[i]);
            found = 1;
            break;
        }
    }
    if (!found)
    {
        printf("Vehicle not found!\n");
    }
}
```

- Exit Module

Function: Controlled by the main loop.

The program will now safely end and save whatever data is necessary.

Process : It will end the loop in the main function

Output : It shows a message of exit and also safely ends the program.

Code :-

```
int main()
{
    struct Vehicle vehicles[MAX_VEHICLES];
    int count = LoadData(vehicles); // Load initial data from file

    int choice = 0; // Initialize choice to avoid garbage value
    while (choice != 4)
    {
        printf("\n***** PARKING MANAGEMENT SYSTEM *****\n");
        printf("1. Park Vehicle\n");
        printf("2. Display Parked Vehicles\n");
        printf("3. Search Vehicle\n");
        printf("4. Exit\n");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                ParkVehicle(vehicles, &count);
                break;
        case 2:
          DisplayVehicles(vehicles, count);
          break;
        case 3:
          SearchVehicle(vehicles, count);
          break;
        case 4:
          printf("Exiting...\n");
          break;
        default:
          printf("Invalid choice! Please try again.\n");
          break;
      }
    }

    return 0;
}
```

Function Point Analysis

Function Point Analysis (FPA) is applied for measuring the complexity and the size of the PMS as follows:

External Inputs (EI)

- Park Vehicle: 1 FP
- Search Vehicle: 1 FP

External Outputs (EO)

- Display Parked Vehicles: 1 FP
- Calculate Fee: 1 FP

External Inquiries (EQ)

- Search Vehicle: 1 FP

Internal Logical Files (ILF)

- Vehicle Data File (parking.txt): 1 FP

External Interface Files (EIF)

- None: 0 FP

Total Function Points (FP) = EI + EO + EQ + ILF + EIF = 6 FP

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX_VEHICLES 100
#define FEE_PER_HOUR 10 // Define a fee per hour

struct Vehicle
{
  char vehicleNo[20];
  int slotNo;
  time_t parkedTime;
};

int LoadData(struct Vehicle vehicles[])
{
  FILE *file = fopen("parking.txt", "r");
  if (file == NULL)
  {
    printf("Error opening file. Starting with an empty parking lot.\n");
    return 0;
  }

  int count = 0;
```

```
    while (fscanf(file, "%s %d %ld", vehicles[count].vehicleNo, &vehicles[count].slotNo, &vehicles[count].parkedTime) != EOF)
    {
        count++;
    }
    fclose(file);
    return count;
}

int SaveData(struct Vehicle vehicles[], int count)
{
    FILE *file = fopen("parking.txt", "w");
    if (file == NULL)
    {
        printf("Error opening file for saving.\n");
        return 0;
    }
    for (int i = 0; i < count; i++)
    {
        fprintf(file, "Vehicle No. :%s , Slot No. :%d , Fee :%ld\n", vehicles[i].vehicleNo, vehicles[i].slotNo, vehicles[i].parkedTime);
    }
    fclose(file);
    return 1;
}

int AllocateSlot(struct Vehicle vehicles[], int count)
{
    for (int i = 0; i < MAX_VEHICLES; i++)
    {
        int slotAvailable = 1;
        for (int j = 0; j < count; j++)
        {
            if (vehicles[j].slotNo == i)
            {
                slotAvailable = 0;
                break;
            }
        }
        if (slotAvailable)
        {
            return i;
        }
    }
    return -1;
    }

void ParkVehicle(struct Vehicle vehicles[], int *count)
{
    if (*count >= MAX_VEHICLES)
    {
        printf("**** PARKING IS FULL ****\n");
        return;
    }
    struct Vehicle newVehicle;
    printf("Enter vehicle number: ");
    scanf("%s", newVehicle.vehicleNo);

    newVehicle.slotNo = AllocateSlot(vehicles, *count);
    if (newVehicle.slotNo == -1)
    {
        printf("No available slots!\n");
        return;
    }

    newVehicle.parkedTime = time(NULL);
    vehicles[*count] = newVehicle;
    (*count)++;
    if (SaveData(vehicles, *count))
    {
        printf("Vehicle parked at slot number: %d\n", newVehicle.slotNo);
    }
}

void CalculateFee(struct Vehicle vehicle)
{
    time_t currentTime = time(NULL);
    double hoursParked = (double)(currentTime - vehicle.parkedTime) / 3600;
    double fee = hoursParked * FEE_PER_HOUR;
    printf("Vehicle No: %s, Parked for: %.2f hours, Fee: Rupees %.2f\n", vehicle.vehicleNo,
hoursParked, fee);
}

void DisplayVehicles(struct Vehicle vehicles[], int count)
{
    if (count == 0)
    {
        printf("\nNo vehicles are parked currently.\n");
        return;
    }
    printf("\nParked Vehicles:\n");
    for (int i = 0; i < count; i++)
    {
        printf("Slot No: %d\n", vehicles[i].slotNo);
        CalculateFee(vehicles[i]);
    }
}

void SearchVehicle(struct Vehicle vehicles[], int count)
{
    char searchNo[20];
    printf("Enter vehicle number to search: ");
    scanf("%s", searchNo);

    int found = 0;
    for (int i = 0; i < count; i++)
    {
        if (strcmp(vehicles[i].vehicleNo, searchNo) == 0) {
            printf("Vehicle found!\n");
            printf("Vehicle No: %s, Slot No: %d\n", vehicles[i].vehicleNo, vehicles[i].slotNo);
            CalculateFee(vehicles[i]);
            found = 1;
            break;
        }
    }
    if (!found)
    {
        printf("Vehicle not found!\n");
    }
}
```

```
int main()
{
    struct Vehicle vehicles[MAX_VEHICLES];
    int count = LoadData(vehicles);
     int choice = 0;
    while (choice != 4)
    {
        printf("\n***** PARKING MANAGEMENT SYSTEM *****\n");
        printf("1. Park Vehicle\n");
        printf("2. Display Parked Vehicles\n");
        printf("3. Search Vehicle\n");
        printf("4. Exit\n");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                ParkVehicle(vehicles, &count);
                break;
            case 2:
                DisplayVehicles(vehicles, count);
                break;
            case 3:
                SearchVehicle(vehicles, count);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
                break;
        }
    }

    return 0;
}
```

Implementation and Testing

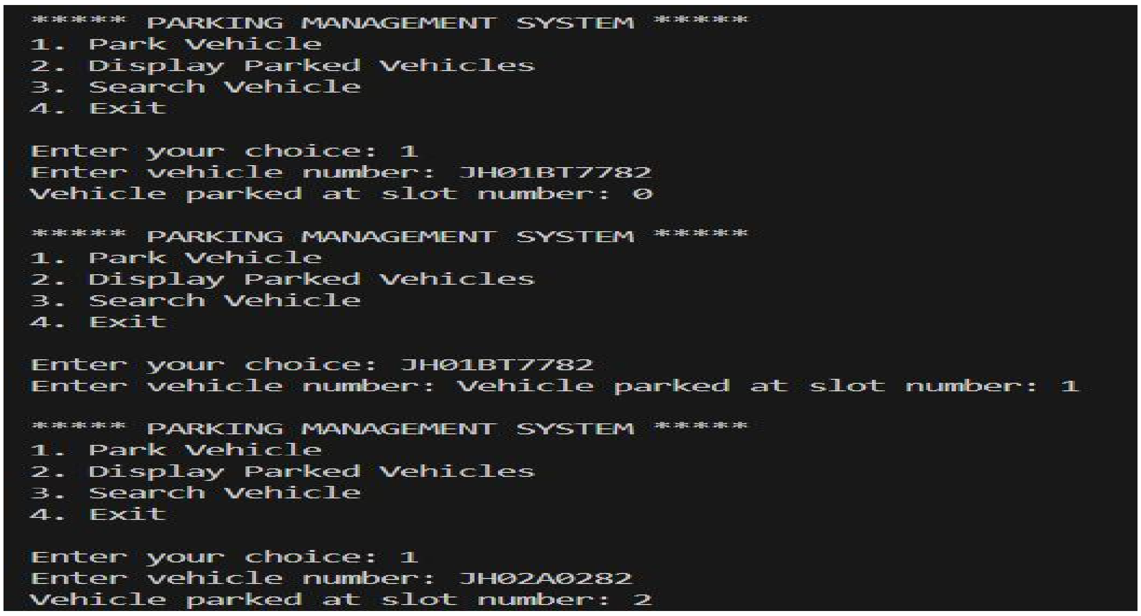
Implementation Details

The system is coded in C and made to use standard file handling for data persistence. Functions are designed to handle various tasks such as data loading, slot allocation, and fee calculation efficiently.

Testing Scenarios

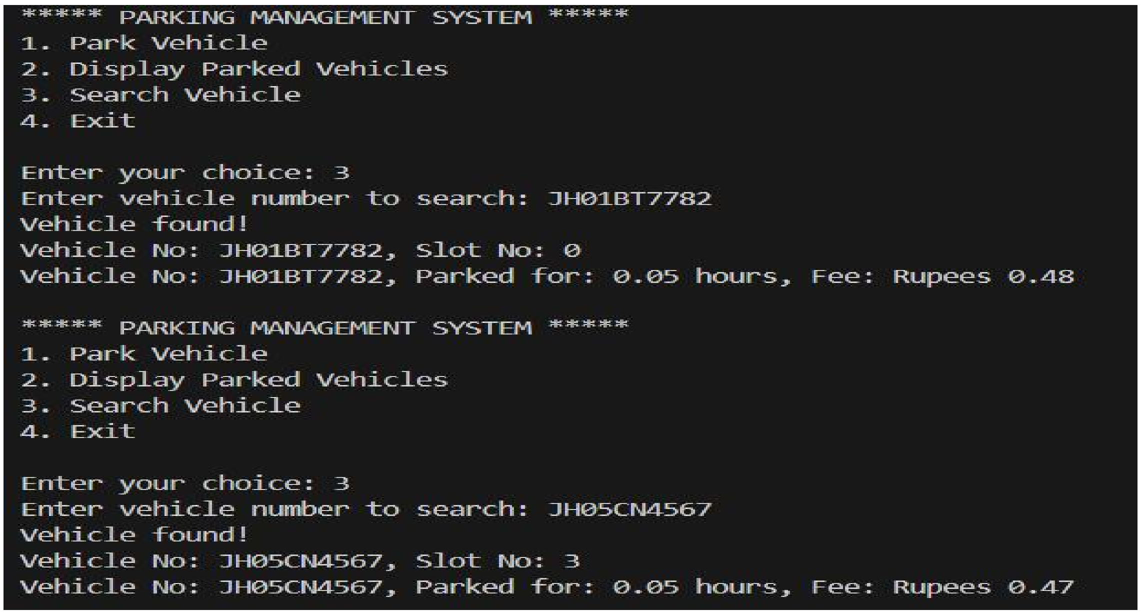
The system was tested for the following scenarios:

1. Normal operations, like parking and searching vehicles.



```
***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: 1
Enter vehicle number: JH01BT7782
Vehicle parked at slot number: 0

***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: JH01BT7782
Enter vehicle number: Vehicle parked at slot number: 1

***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: 1
Enter vehicle number: JH02A0282
Vehicle parked at slot number: 2
```

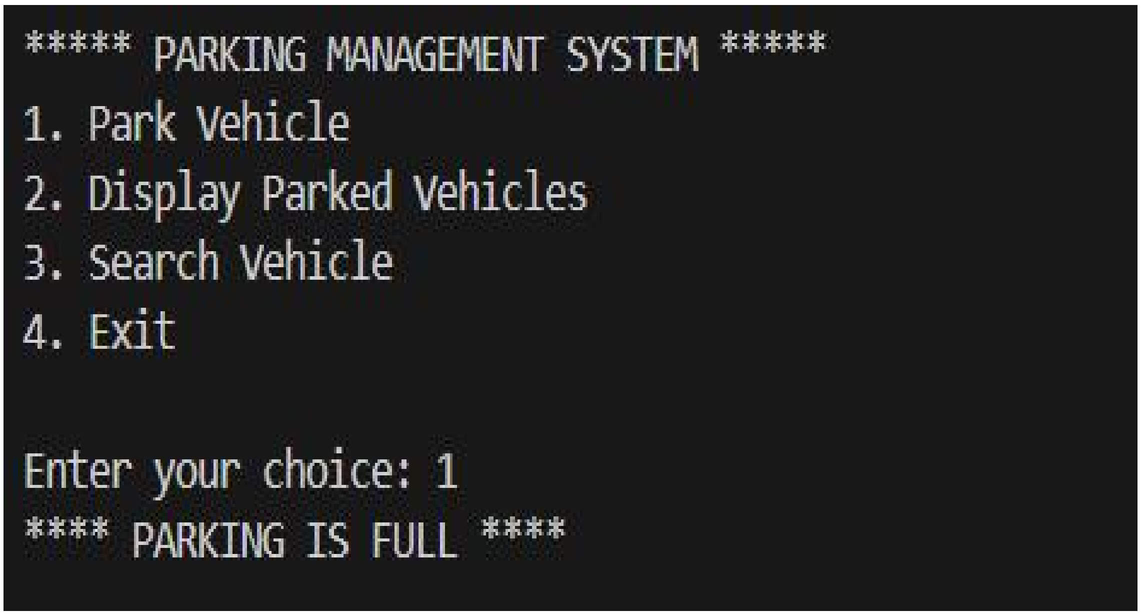
Fig 1.1 : Shows Park operation done in PMS



```
***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: 3
Enter vehicle number to search: JH01BT7782
Vehicle found!
Vehicle No: JH01BT7782, Slot No: 0
Vehicle No: JH01BT7782, Parked for: 0.05 hours, Fee: Rupees 0.48

***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: 3
Enter vehicle number to search: JH05CN4567
Vehicle found!
Vehicle No: JH05CN4567, Slot No: 3
Vehicle No: JH05CN4567, Parked for: 0.05 hours, Fee: Rupees 0.47
```

Fig 1.2 : Shows Search operation done in PMS

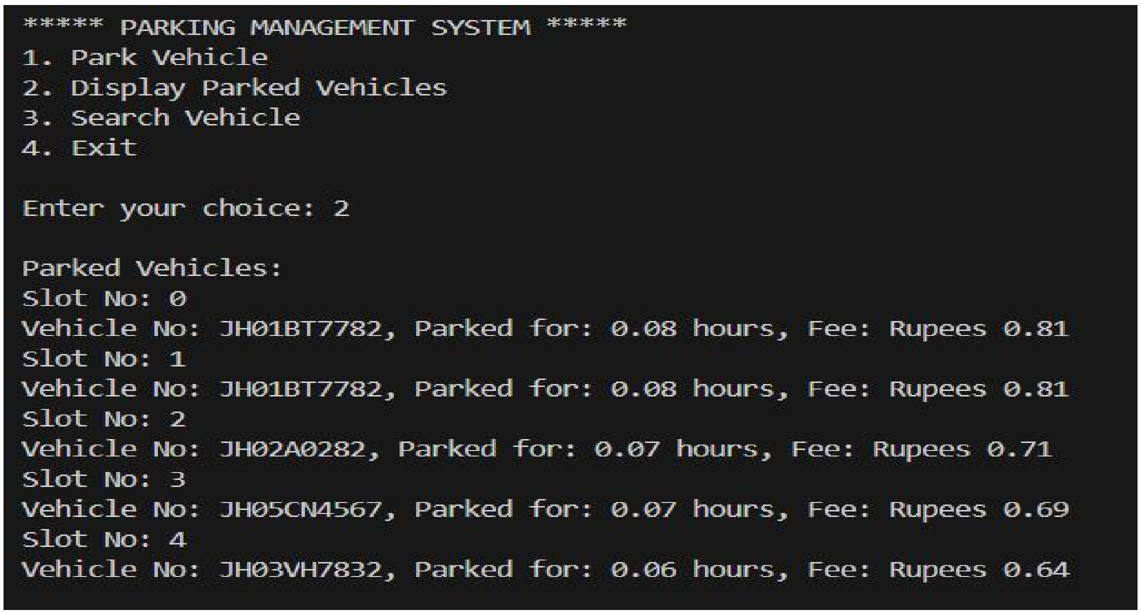
2. Edge cases, such as when a vehicle is to be parked while the lot is full.



```
***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: 1
**** PARKING IS FULL ****
```

Fig 2 : Shows Park operation done in PMS when parking slots are full

3. Accuracy of fee calculation for different time duration.



```
***** PARKING MANAGEMENT SYSTEM *****
1. Park Vehicle
2. Display Parked Vehicles
3. Search Vehicle
4. Exit

Enter your choice: 2

Parked Vehicles:
Slot No: 0
Vehicle No: JH01BT7782, Parked for: 0.08 hours, Fee: Rupees 0.81
Slot No: 1
Vehicle No: JH01BT7782, Parked for: 0.08 hours, Fee: Rupees 0.81
Slot No: 2
Vehicle No: JH02A0282, Parked for: 0.07 hours, Fee: Rupees 0.71
Slot No: 3
Vehicle No: JH05CN4567, Parked for: 0.07 hours, Fee: Rupees 0.69
Slot No: 4
Vehicle No: JH03VH7832, Parked for: 0.06 hours, Fee: Rupees 0.64
```

Fig 2 : Shows Display operation done in PMS which shows the Vehicle no. , time duration , and parking fees

Conclusion

This efficient, reliable car parking control can be carried out easily even in smaller and mid-level parking systems. Modular product design that allows ease to maintain with standard C-sided features simplifies any facility's future expansion; prospects of upgrades will thus have a Graphical User Interface as well as linking this system up with automated account billing system.