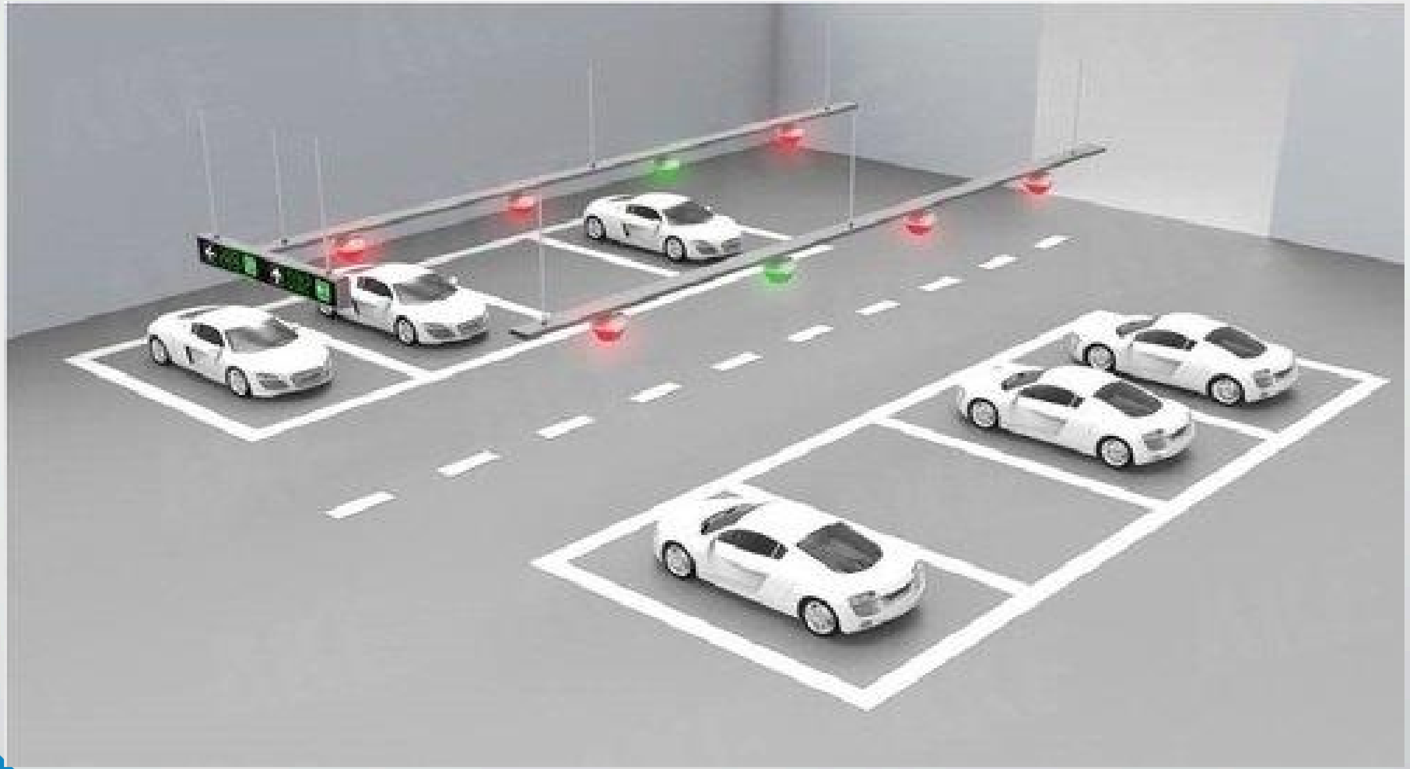
Parking management system



NAME : M . Jayanth Swaroop
ROLL NO : 25A31A0142

Component Requirement

Processor (CPU): Dual-Core

Processor RAM : 512 GB

MORE HDD : 20GB

Software Requirements:

Operating System Windows : 7 / 8 / 10 / 11. (OR) Linux (OR) macOS

Programming Tools :C Compiler Turbo Ccode

Editors / IDE:

Use any one : Code Blocks

VS Code DevC++

Turbo C++

• INTRODUTION:

1. Helps managevehicleentry and exit in a systematic way.

2. Stores the details of each parked vehicle safely in the system.

3. Provides quick information about available and occupied parkingslots.

4.Reduces manualwork andhumanerrorsduring parking operations.

5. Improves time management by speeding up the parking process.

6. Offers features like parking charges and waiting list for better control.

Algorithm:

Parking Management System

Step 1: Start

Step 2: Initialize total parking slots (N) and set parked vehicle count to 0.

Step 3: Display the menu options:

1. ParkVehicle
2. Remove Vehicle
3. View Parking Status
4. Count Available Slots
5. Parking Charges (if required)
6. Exit

```
Step 4: Read user's choice
 Step 5: If choice is 1: (Park Vehicle)
 a. If parking is not full:
 Enter vehicle number
Add vehicle details to parking list
Increase vehicle count
Display “Vehicle Parked Successfully”
b. Else:
Display “Parking Full, Vehicle Added to Waiting List”
Step 6: If choice is 2: (Remove Vehicle)
a. Enter vehicle number
b. Search vehicle in parking list
c. If found:
Remove vehicle
Decrease vehicle count
Calculate parking charges
If waiting list has vehicles, move first waiting vehicle into slot
```

d. Else:

- Display “Vehicle Not Found”

Step 7: If choice is 3: (View Parking Status)

Display all parked vehicles and slot numbers

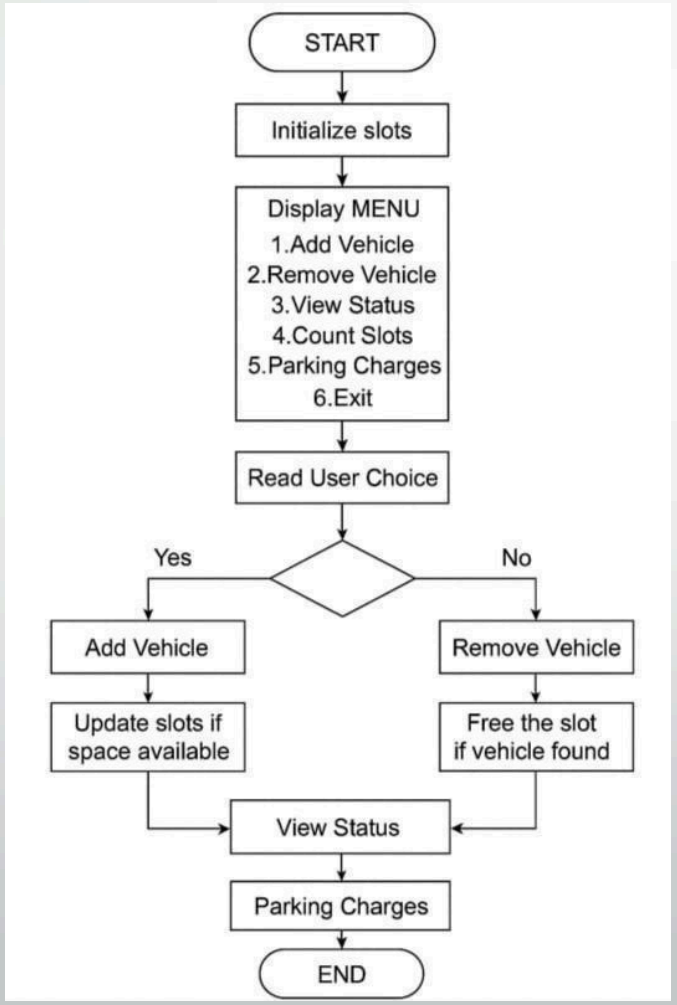
- Step 8: If choice is 4: (Count Available Slots)

Display Available Slots = N -current vehicle count

- Step 9: If choice is 6:

- Exit the program
- Step 10: Repeat steps 3–10 until exit selected
- Step 11: Stop

FLOW CHART:


START
Initialize slots
Display MENU
1.Add Vehicle
2.Remove Vehicle
3.View Status
4.Count Slots
5.Parking Charges
6.Exit
Read User Choice
Yes
No
Add Vehicle
Remove Vehicle
Update slots if space available
Free the slot if vehicle found
View Status
Parking Charges
END

- MODULES:

1. Park Vehicle

Takes input: Vehicle Number and Owner Name

Stores details in the parking array

Increases the count of parked vehicles

Displays success message

- 2. Remove Vehicle

Takes input: Vehicle Number to be removed

Searches the list using string comparison

Removes the vehicle by shifting remaining records

Decreases vehicle count

Shows status message whether removed/not found

- 3. View Parked Vehicles

Displays all currently parked vehicles

Shows Serial No., Vehicle Number, and Owner Name

If no vehicles parked → displays appropriate message

4. Count Total Vehicles

Shows current number of vehicles parked in the lot

5. Exit

Terminates the program safely with a message

- LOGIC(Module wise logic):

Module 1: parkVehicle()

Logic:

1. Check if parking is full (count ≥ MAX)
2. If full → display message “Parking is Full” and return
3. Else → ask user to enter vehicle number
4. Store entered vehicle number in the parked[count].vehicleNo
5. Ask user to enter owner name

6. Store entered name in parked count. owner Name

7. Increase count by 1

8. Display “Vehicle Parked Successfully”

Module 2: remove Vehicle()

- Logic:

1. Check if no vehicles are parked (count == 0)
2. If yes → display “No Vehicles to Remove” and exit
3. Ask user to enter vehicle number to remove
4. Search for the vehicle in the array using a loop
5. If match found:

Remove the vehicle by shifting all next elements left

Decrease count by 1

Display “Vehicle Removed Successfully”

6. If not found → display “Vehicle Not Found”

- ## Module 3: view Vehicles()

Logic:

1. Check if count == 0

2. If yes → display “No Vehicles Parked” and exit

3. Else display heading: “Parked Vehicles”

4. Loop through all parked vehicles from index 0 to count-1

5. Print vehicle number and owner name for each

- ## Module 4: count Vehicles()

Logic:

1. Simply display the value of count

(Total number of currently parked vehicles)

Module 5: main() – Menu / Control Module

Logic:

- 1. Use infinite loop to repeatedly show menu
- 2. Display options:

1 Park Vehicle

2 Remove Vehicle

3 View Vehicles

4 Count Vehicles

5 Exit

- 3. Read user choice
- 4. Using switch() call specific module:

1 → park Vehicle()

2 → remove Vehicle()

3 → view Vehicles()

4 → count Vehicles()

5 → Exit program

5. If wrong input→ display “Invalid Choice

MODULE INTEGRATION:

```
#include <stdio.h>
#include <string.h>
#define MAX 100
struct Parking {
 char vehicleNo[20];
 char ownerName[30];
};
struct Parking parked[MAX];
int count = 0;
void parkVehicle() {
 if (count >= MAX) {
 printf("\nParking is Full!\n");
     return;
   }
 printf("Enter Vehicle Number: ");
        scanf("%s",        parked[count].vehicleNo);
printf("Enter   Owner   Name:   ");     scanf("%s",
parked[count].ownerName);
 count++;
 printf("Vehicle Parked Successfully!\n");
}
void removeVehicle() {
 if (count == 0)
 {
       printf("\nNo Vehicles to Remove!\n");
       return;
  }
 char vehicle[20];
 printf("Enter Vehicle Number to Remove: ");
 scanf("%s", vehicle);
```

```
int found = 0;
for (int i = 0; i < count; i++) {
    if (strcmp(parked[i].vehicleNo, vehicle) == 0) {
        for (int j = i; j < count - 1; j++) {
            parked[j] = parked[j + 1];
        }
        count--;
        found = 1;
        printf("Vehicle Removed Successfully!\n");
        break;
    }
  }
  if (!found) {
    printf("Vehicle Not Found!\n");
  }
}
```

```
void viewVehicles() {
if (count == 0) {
      printf("\nNo Vehicles Parked!\n");
      return;
   }
 printf("\nParked Vehicles:\n");
 for (int i = 0; i < count; i++)
{
      printf("%d. Vehicle No: %s | Owner: %s\n", i + 1, parked[i].vehicleNo, parked[i].ownerName);
   }
}
 void countVehicles()
 {
printf("\nTotal Vehicles Parked: %d\n", count);
}
```

```
int main()
{
int choice;
while (1)
{
printf("\n===== Parking Management System =====\n");
printf("1. Park Vehicle\n");
printf("2. Remove Vehicle\n");
printf("3. View Parked Vehicles\n");
printf("4. Count Total Vehicles\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
```

```
switch (choice) {
        case1:
           parkVehicle();
           break;
        case 2:
           removeVehicle();
           break;
        case 3:
           viewVehicles();
           break;
        case 4:
           countVehicles();
           break;
        case 5:
        printf("Thank you! Exiting...\n") return 0;
return 0;
        default:
           printf("Invalid Choice! Try Again.\n");
}
   }

 return 0;
}
```

OUTPUT:

```
===== Parking Management System =====
1. Park Vehicle
2. Remove Vehicle
3. View Parked Vehicles
4. Count Total Vehicles
5. Exit
Enter your choice: 1
Enter Vehicle Number: Ap36cd6666
Enter Owner Name: srinu
Vehicle Parked Successfully!
```

```
===== Parking Management System =====
1. Park Vehicle
2. Remove Vehicle
3. View Parked Vehicles
4. Count Total Vehicles
5. Exit
Enter your choice: 1
Enter Vehicle Number: Ap34ct1111
Enter Owner Name: siddu
Vehicle Parked Successfully!

===== Parking Management System =====
1. Park Vehicle
2. Remove Vehicle
3. View Parked Vehicles
4. Count Total Vehicles
5. Exit
Enter your choice: 2
Enter Vehicle Number to Remove: Ap36cd6666
Vehicle Removed Successfully!
```

```
===== Parking Management System =====
1. Park Vehicle
2. Remove Vehicle
3. View Parked Vehicles
4. Count Total Vehicles
5. Exit
Enter your choice: 3

Parked Vehicles:
1. Vehicle No: Ap34ct1111 | Owner: siddu

===== Parking Management System =====
1.Park Vehicle
2. Remove Vehicle
3. View Parked Vehicles
4. Count Total Vehicles
5. Exit
Enter your choice: 4
Total Vehicles Parked: 1
```

CONCLUSION :

The Parking Management System simplifies the process of managing parked vehicles by storing and organizing vehicle details efficiently. This program reduces manual work, avoids errors, and shows how C programming can be used to solve real-life problems through proper logic and data handling.