

**UNIVERSITY INSTITUTE OF**

**COMPUTING**

***PROJECT***

***Program Name:***

***BCA(Data Science)***

***Subject Name/Code:***

***Data Structure Lab***

***24***

***CAH***

***-152***

Submitted By

Submitted To

Name:

**Vinayak Khanna**

Name:

**Mrs. Mehak Bhatia**

UID:

**24**

**BCD**

**10040**

Designation:

**Assistant Professor**

Section:

**24**

**BCD1-B**

***Smart Car Parking System***

**Introduction:-**

The **Smart Car Parking System** is a C programming project designed to simulate the functioning of a real-world car parking lot using the **Stack data structure**. This system operates on the **Last-In, First-Out (LIFO)** principle, where the most recently parked car is the first one to leave—just like stacking or unstacking physical objects.

In this system, cars are represented as elements pushed onto a stack when they enter the parking lot. When a car needs to be removed, especially one that is not at the top, cars above it are temporarily moved to another stack, simulating real-life car movement in narrow parking lanes. Once the target car is removed, the temporarily moved cars are restored in their original order.

The program supports a variety of operations:

* **Parking a car**
* **Removing a car**
* **Displaying parked cars**
* **Handling full and empty parking conditions**

This project not only enhances logical thinking and C programming skills but also gives a strong practical understanding of how **data structures like stacks** are used in real-world scenarios. It is ideal for **students and beginners** who want to explore the practical applications of data structures and build problem-solving skills through hands-on coding.

***Techniques Used :***

 **Stack Data Structure (LIFO):**  
Implements the parking system using a stack, where the last car to enter is the first to leave, simulating real-life narrow-lane parking behavior.

 **Array-Based Stack Implementation:**  
Uses a fixed-size array to efficiently manage car entries and exits with push/pop operations, ensuring controlled memory usage.

 **Menu-Driven Program Structure:**  
Allows user interaction through a simple interface with options to park, remove, display, or exit, using loops and switch-case for control flow.

***System Configuration :***

*** Operating System: Windows 10 / 11 or any Linux Distribution***

*** Processor: Intel Core i3 (minimum); Core i5 or higher recommended***

*** RAM: 4 GB (minimum); 8 GB or more for smoother performance***

*** Development Environment:***

* ***Any C IDE (e.g., Code::Blocks, Dev C)***
* ***Or Visual Studio Code with GCC (MinGW for Windows) or Clang for Linux***

*** Compiler: GNU GCC Compiler (recommended).***

***Code :***

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int cars[MAX];

int top;

} Stack;

void initialize(Stack \*s) {

s->top = -1;

}

int isFull(Stack \*s) {

return s->top == MAX - 1;

}

int isEmpty(Stack \*s) {

return s->top == -1;

}

void parkCar(Stack \*s, int carNumber) {

if (isFull(s)) {

printf("Parking is Full! No space available.\n");

return;

}

s->cars[++s->top] = carNumber;

printf("Car %d parked successfully.\n", carNumber);

}

void removeCar(Stack \*s, int carNumber) {

if (isEmpty(s)) {

printf("Parking is empty! No cars to remove.\n");

return;

}

Stack temp;

initialize(&temp);

int found = 0;

while (!isEmpty(s)) {

int currentCar = s->cars[s->top--];

if (currentCar == carNumber) {

printf("Car %d has left the parking.\n", carNumber);

found = 1;

break;

} else {

temp.cars[++temp.top] = currentCar; // Directly move to temp

}

}

while (!isEmpty(&temp)) {

s->cars[++s->top] = temp.cars[temp.top--]; // Restore without parking checks

}

if (!found) {

printf("Car %d not found in the parking.\n", carNumber);

}

}

void display(Stack \*s) {

if (isEmpty(s)) {

printf("Parking is empty!\n");

return;

}

printf("Current Parking Status:\n");

for (int i = s->top; i >= 0; i--) {

printf("Car %d\n", s->cars[i]);

}

}

int main() {

Stack parking;

initialize(&parking);

int choice, carNumber;

do {

printf("\n🚦Smart Car Parking System\n");

printf("1. Park a Car\n");

printf("2. Remove a Car\n");

printf("3. Display Parking\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (!isFull(&parking)) {

printf("Enter Car Number: ");

scanf("%d", &carNumber);

parkCar(&parking, carNumber);

} else {

printf("Parking is Full!\n");

}

break;

case 2:

if (!isEmpty(&parking)) {

printf("Enter Car Number to Remove: ");

scanf("%d", &carNumber);

removeCar(&parking, carNumber);

} else {

printf("Parking is already empty!\n");

}

break;

case 3:

display(&parking);

break;

case 4:

printf("🚦 Exiting Smart Parking System. Have a great day!\n");

break;

default:

printf("❌ Invalid choice! Please enter a valid option.\n");

}

} while (choice != 4);

return 0;

}

***Input :***

🚦Smart Car Parking System

1. Park a Car

2. Remove a Car

3. Display Parking

4. Exit

Enter your choice: 1

Enter Car Number: 0001

Car 1 parked successfully.

Enter your choice: 1

Enter Car Number: 0002

Car 2 parked successfully.

Enter your choice: 3

Current Parking Status:

Car 2

Car 1

Enter your choice: 2

Enter Car Number to Remove: 0002

Car 2 has left the parking.

Enter your choice: 3

Current Parking Status:

Car 1

Enter your choice: 4

🚦 Exiting Smart Parking System. Have a great day!s.

***Process :***

A diagram of a car

AI-generated content may be incorrect.

***Outputs :***

