

# Parking Lot Management System – Documentation

By: Mohammad Mahdi Jafari & Mobin Motamedi Moghadam

## 1. Project Overview

This project was developed as part of a **Data Structures and Algorithms learning assignment**.

Its main purpose is to help students understand and practice:

- **Queue operations (FIFO behavior)**
- **Stack operations using Linked Lists (LIFO behavior)**
- **Linked List node manipulation**
- **Array of complex data structures (array of stacks)**
- **Search, sort, move, and remove operations on custom data structures**
- **Real-world application of stacks and queues in simulation problems**

To achieve this, we modeled a real-life parking lot system using core data structures:

- **Entrance Queue** → Cars enter in FIFO order
- **Parking Lanes (Stacks)** → Implemented using Linked Lists
- **Parking Lot** → An array of these stack-based lanes

The system enforces realistic parking constraints:

- Cars enter in arrival order
- Cars exit only if they are **at the top** of their lane
- Cars cannot be removed from the middle of a lane
- Sorting, finding, and moving cars follow strictly defined rules

A complete **FLTK-based Graphical User Interface (GUI)** was also created to visualize each operation in real time.

## 2. Data Structures Used

### Queue (Entrance)

This structure teaches:

- FIFO behavior
- Node-based queue implementation
- Handling car arrival order

Used to manage incoming cars with operations:

- enqueue
- dequeue
- front
- isEmpty

## Stack (Parking Lane using Linked List)

Each lane is a **Stack implemented using a singly Linked List**, designed to reinforce:

- Node-based stack memory behavior
- Push/pop mechanics
- Handling LIFO operations
- Linked List traversal
- Implementing recursive Merge Sort on Linked Lists

Operations include:

- push
- pop
- peek
- isEmpty
- isFull
- findPosition
- size
- sort (recursive merge sort)

## Array of Stacks

Represents the entire parking lot:

- An array where each element is a Linked-List-based stack
- Strengthens understanding of arrays storing complex objects
- Helps learn multi-layered data structures

## 3. Core Functionalities

### Car Entry

- Park in **first available lane**
- Park in **specific lane**
- If all lanes are full → **"Parking Full"**

### Car Exit

- Car can exit **only if it is at the top** of its lane
- Otherwise → **"Cannot remove"**
- Demonstrates stack constraints and LIFO behavior

### Search Car

Returns:

- Lane (stack) number
- Position from top

This reinforces **Linked List traversal** and **search complexity**.

### **Sort a Lane**

- Uses **Merge Sort** adapted for Linked Lists
- Shows how traditional sorting algorithms must be modified for non-array structures

### **Move Car Between Lanes**

Only allowed if:

- Car is at the **top** of its current lane
- Target lane has remaining capacity

Involves:

- Searching through stacks
- Verifying stack constraints
- Moving node data safely

This project emphasizes understanding how time complexity changes depending on the data structure layout.

## **5. GUI (FLTK Visualizer)**

The GUI was created to:

- Help visualize how stacks and queues behave internally
- Show structural changes dynamically
- Provide an interactive learning experience

The GUI displays:

- Entrance queue
- All parking lanes
- Top car highlight
- Movement animations (push/pop)
- Stack sorting
- Real-time updates

Users can perform:

- Add car
- Park car
- Move car
- Exit car
- Search
- Sort
- Reset system

**This enhances understanding of how the data structures respond to each operation.**

## 6. Educational Purpose

This project serves as a **practical demonstration** of:

- How Stacks and Queues behave in real scenarios
- Linked List manipulations
- Using arrays to organize multiple dynamic structures
- Implementing sort algorithms for non-array structures
- Designing modular, object-oriented systems in C++
- Building a GUI to visualize algorithms

It reinforces the principles of:

- Memory links**
- Node-based operations**
- Algorithmic constraints**