MULTIMEDIA UNIVERSITY®

FACULTY COMPUTING AND INFORMATIC

Inquire, Inspire and Innovate

CPP 4213

DATA STRUCTURES AND ALGORYTHMS

Project Report
The Paddock Club

October/November 2025 (Term 2530)

BY:

NO.	STUDENT NAME	STUDENT ID
1.	Adam bin Anwar	243DC245L4

TO

LECTURER NAME	Ruzanna Abdullah

MULTIMEDIA UNIVERSITY

Table Of Contents

1.0 Introduction

1.1 Project Overview

"The Paddock Club" is a specialized console based management system developed for a high-end car rental business. Unlike standard rental systems, this project caters to a niche market of exotic and performance vehicles, requiring specific logic for high-horsepower restrictions, maintenance cycles, and dynamic inventory management.

The system is built using C++ and demonstrates the practical application of advanced data structures including Linked Lists, Circular Queues, Stacks, and Hash Tables to solve real world business problems efficiently. It features a persistent database system (CSV) to ensure data integrity across multiple sessions.

1.2 Objective

The primary objective of the project are:

- To implement core Data Structures (Linked List, Stacks, Queues and Hash Tables) in practical business scenario.
- To implement efficient Algorithms (Merge Sort, Binary Search) to manage data retrieval and organization.
- To provide a seamless CRUD (Create, Read, Update, Delete) experience for administrators managing the fleet.

1.3 Scope and Target User

The system is designed for rental administrators. Its scope includes:

- **Fleet Management:** Dynamic addition and removal of vehicles.
- **Rental Processing:** Customer booking with age verification and stock tracking.
- **Maintenance:** A "Wash Bay" system that queues returned cars before they can be rented again.
- **Analytics:** Tracking revenue and generating audit logs for all user actions.

2.0 Data Structures

The system architecture relies on four distinct data structures, each chosen to optimize a specific aspect of the application.

2.1 Linked Lists (Inventory & Customers)

Theory: A Linked List is a linear data structure where elements are not stored in contiguous memory locations. Instead, each element (node) points to the next.

Implementation A: Fleet Inventory, we selected a Singly Linked List for the main showroom inventory because the fleet size is dynamic. In an real world scenario, cars are frequently bought (added) and sold (deleted). A linked list allows for *O(1)* insertion and deletion at the head, preventing the need to resize arrays constantly.

Figure 2.1.1: Node Structure Implementation First, we defined the Node structure, which acts as the building block. Each node holds a pointer to a Car object and a pointer to the next node.

```
//Node for Linked List of Cars
struct Node { Car* data; Node* next; Node(Car* c) : data(c), next(NULL) {} };
```

Implementation B: Customer Records, a second linked list tracks active rentals. As customers rent cars, new nodes are linked to the head of the list. This provides a dynamic record of all active bookings without pre-allocating memory for a maximum number of customers.

Figure 2.1.2: Add Function Implementation The add function handles the logic of inserting these nodes. We traverse to the end of the list to ensure new cars are added to the bottom of the showroom table, maintaining the order of entry.

```
// Adds a new car to the fleet
void add(Car* c) {
    Node* n = new Node(c);
    if (!head) head = n; else { Node* t = head; while (t->next) t = t->next; t->next = n; }
    ht->insert(c);
}
```

2.2 Circular Queue (Wash Bay)

Theory: A Queue follows the First-In, First-Out (FIFO) principle. We implemented this as a Circular Array to simulate the "Wash Bay." When a car is returned, it must be washed before it is available again.

Implementation: The ServiceQueue class uses a fixed array of size 10. Variables front and rear track the queue's state. Modulo arithmetic (%10) is used to wrap the indices around the array. This ensures that we can reuse array positions continuously without shifting elements, providing *O(1)* complexity for enqueue and dequeue operations.

Figure 2.2.1: Queue Class Properties We defined a fixed integer array and tracking variables for the front and rear of the queue.

```cpp
// QUEUE DATA STRUCTURE: Circular Array implementation for the Wash Bay
// Represents a "First-In, First-Out" (FIFO) system for servicing returned cars.
class ServiceQueue {
    Car* queue[10]; int front, rear, size;
public:
    ServiceQueue() : front(0), rear(-1), size(0) {}
    bool isFull() { return size == 10; }
    bool isEmpty() { return size == 0; }

```

Figure 2.2.2: Enqueue Logic The enqueue function adds a car to the wash bay. It uses the modulo operator to "wrap around" to index 0 if the array reaches index 9.

```cpp
    // Adds a car to the service queue
    void enqueue(Car* c) {
        if (isFull()) { cout << RED << "Bay Full!" << RST << endl; return; }
        rear = (rear + 1) % 10; queue[rear] = c; size++; c->status = "In-Service";
    }
```

2.3 Linked Stack (Activity Logs)

Theory: A Stack follows the Last-In, First-Out (LIFO) principle. This is ideal for activity logs (Audit Trails) because administrators typically want to see the most recent actions first. We used a Linked List implementation for the stack to avoid the fixed-size limitations of array-based stacks.

Figure 2.3.1: Stack Node Structure Similar to the main inventory, the logs require their own node structure to hold the string data.

```
// STACK DATA STRUCTURE: Linked List implementation for History/Logs
// Represents a "Last-In, First-Out" (LIFO) system for tracking user actions.
struct LogNode { string log; LogNode* next; LogNode(string s) : log(s), next(NULL) {} };
```

Figure 2.3.2: Push Operation The push operation creates a new node and points it to the current top.

```
// STACK DATA STRUCTURE: Linked List implementation for History/Logs
// Represents a "Last-In, First-Out" (LIFO) system for tracking user actions.
struct LogNode { string log; LogNode* next; LogNode(string s) : log(s), next(NULL) {} };
class HistoryStack {
    LogNode* top;
public:
    HistoryStack() : top(NULL) {}

    // Pushes a new action to the stack and logs it to a text file
    void push(string s) {
        LogNode* n = new LogNode(s); n->next = top; top = n;
        ofstream f(s: getDataPath(fileName: "audit_log.txt").c_str(), mode: ios::app);
        if (f.is_open()) {
            time_t now = time(0); string ts = ctime(&now);
            if (!ts.empty()) ts.erase(pos: ts.length()-1);
            f << "[" << ts << "] " << s << endl;
        }
    }
```

2.4 Hash Table (Fast Lookup)

Theory: Searching through a Linked List is *O(n)*, which can be slow for large inventories. A Hash Table reduces average search time to *O(1)* by mapping keys (Car IDs) to specific indices in an array.

Implementation: We used a custom hash function that sums the ASCII values of the Car ID characters and applies modulo 50. Collision resolution is handled via Chaining (each bucket contains a linked list of nodes).

Figure 2.4.1: Hash Function This function converts the string ID (e.g., "GTR35") into an integer index for the array.

```
// Hash Function: Converts a string ID into an array index
int hashFn(string id) {
    int s = 0; for (int i = 0; i < (int)id.length(); i++) s += id[i];
    return s % 50;
}
```

Figure 2.4.2: Search Logic The search function jumps directly to the calculated index, bypassing the need to scan the entire list.

```
// Find a car by its unique ID
Car* search(string id) {
    for (Node* t = table[hashFn(id)]; t; t = t->next) if (t->data->id == id) return t->data;
    return NULL;
}
```

3.0 Algorithms

3.1 Merge Sort (Complex Sorting)

Theory: Merge Sort is a divide-and-conquer algorithm with a time complexity of *O(n log n)*. It is preferred over Bubble Sort for large datasets because of its superior worst-case performance.

Application: We implemented Merge Sort to allow users to sort the car fleet by **Price** (Low-High), **Horsepower** (High-Low), or **Year**.

Figure 3.1.1: Merge Logic This function takes two sorted sub-lists (a and b) and merges them into a single sorted list based on the chosen mode.

```
Node* sortedMerge(Node* a, Node* b, int mode) {
    if (!a) return b; if (!b) return a;
    Node* res = NULL;
    bool cond = false;
    if (mode == 1) cond = (a->data->rate >= b->data->rate); // Price DESC
    else if (mode == 2) cond = (a->data->hp >= b->data->hp); // HP DESC
    else if (mode == 4) cond = (a->data->year <= b->data->year); // Year ASC
    else cond = (a->data->makeModel <= b->data->makeModel); // Name ASC

    if (cond) { res = a; res->next = sortedMerge(a: a->next, b, mode); }
    else { res = b; res->next = sortedMerge(a, b: b->next, mode); }
    return res;
}
```

3.2 Bubble Sort (Simple Sorting)

Theory: Bubble Sort is a simple comparison-based algorithm with *O(n^2)* complexity.

Application: We utilized Bubble Sort specifically for the sortByBrand() function. This runs automatically when the database is loaded to ensure the fleet is alphabetized by default.

Figure 3.2.1: Bubble Sort Implementation The nested loops compare adjacent nodes and swap their data if they are out of order.

```
// Bubble Sort Algorithm: Sorts by brand name alphabetically
void sortByBrand() {
    for (Node* i = head; i; i = i->next)
        for (Node* j = i->next; j; j = j->next)
            if (i->data->makeModel > j->data->makeModel) { Car* temp = i->data; i->data = j->data; j->d
}
```

3.3 Binary Search (Search Logic)

Theory: Binary Search is an efficient search algorithm that finds an element in a sorted list by repeatedly dividing the search interval in half. It has a time complexity of *O(log n)*.

Application: Binary Search is used when searching for cars by **Year**. Since Binary Search requires a sorted array, our implementation first performs a Merge Sort on the list by year, then converts the list to a temporary array.

Figure 3.3.1: Binary Search Implementation

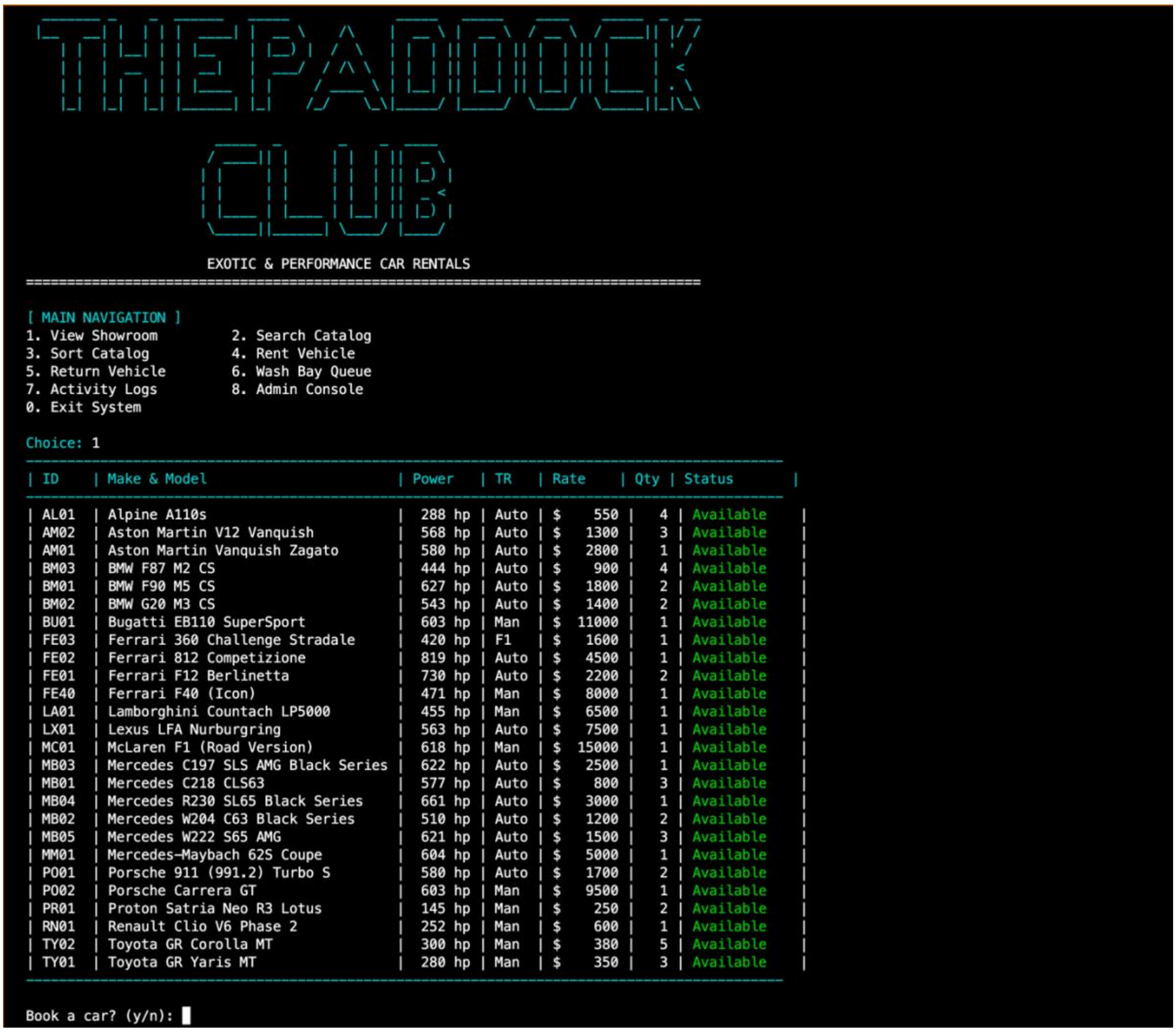
```
// Binary Search Implementation: Search by Year
// Note: Requires the list to be sorted by year first.
void searchByYear(int targetYear) {
    // Sort by year first for binary search to work using Merge Sort (O(n log n))
    performMergeSort(mode: 4);

    // Copy nodes to array for O(1) access
    int count = 0; for (Node* t = head; t; t = t->next) count++;        if (count == 0) return;
Car** arr = new Car*[count];
Node* curr = head; for (int i = 0; i < count; i++) { arr[i] = curr->data; curr = curr->next; }

int l = 0, r = count - 1;
bool found = false;
displayHeader();
while (l <= r) {
    int m = l + (r - l) / 2;
    if (arr[m]->year == targetYear) {
        // Found one, but there might be others with the same year
        // Scan left and right to find all matches
        int left = m; while (left >= 0 && arr[left]->year == targetYear) left--;
        int right = m; while (right < count && arr[right]->year == targetYear) right++;
        for (int i = left + 1; i < right; i++) arr[i]->displayRow();
        found = true; break;
    }
    if (arr[m]->year < targetYear) l = m + 1;
    else r = m - 1;
}
```

4.0 System Features and Output

4.1 Fleet Management (Read/Display)

The main dashboard displays the fleet in a formatted table. We used ANSI escape codes (\033[32m for Green, etc.) to visually distinguish available cars from those that are out of stock.

Figure 4.1.1: Main System Dashboard displaying the full vehicle inventory with ANSI color-coded status indicators.


THE PADDOCK CLUB

EXOTIC & PERFORMANCE CAR RENTALS

[MAIN NAVIGATION]
1. View Showroom 2. Search Catalog
3. Sort Catalog 4. Rent Vehicle
5. Return Vehicle 6. Wash Bay Queue
7. Activity Logs 8. Admin Console
0. Exit System

Choice: 1

ID	Make & Model	Power	TR	Rate	Qty	Status
AL01	Alpine A110s	288 hp	Auto	$ 550	4	Available
AM02	Aston Martin V12 Vanquish	568 hp	Auto	$ 1300	3	Available
AM01	Aston Martin Vanquish Zagato	580 hp	Auto	$ 2800	1	Available
BM03	BMW F87 M2 CS	444 hp	Auto	$ 900	4	Available
BM01	BMW F90 M5 CS	627 hp	Auto	$ 1800	2	Available
BM02	BMW G20 M3 CS	543 hp	Auto	$ 1400	2	Available
BU01	Bugatti EB110 SuperSport	603 hp	Man	$ 11000	1	Available
FE03	Ferrari 360 Challenge Stradale	420 hp	F1	$ 1600	1	Available
FE02	Ferrari 812 Competizione	819 hp	Auto	$ 4500	1	Available
FE01	Ferrari F12 Berlinetta	730 hp	Auto	$ 2200	2	Available
FE40	Ferrari F40 (Icon)	471 hp	Man	$ 8000	1	Available
LA01	Lamborghini Countach LP5000	455 hp	Man	$ 6500	1	Available
LX01	Lexus LFA Nurburgring	563 hp	Auto	$ 7500	1	Available
MC01	McLaren F1 (Road Version)	618 hp	Man	$ 15000	1	Available
MB03	Mercedes C197 SLS AMG Black Series	622 hp	Auto	$ 2500	1	Available
MB01	Mercedes C218 CLS63	577 hp	Auto	$ 800	3	Available
MB04	Mercedes R230 SL65 Black Series	661 hp	Auto	$ 3000	1	Available
MB02	Mercedes W204 C63 Black Series	510 hp	Auto	$ 1200	2	Available
MB05	Mercedes W222 S65 AMG	621 hp	Auto	$ 1500	3	Available
MM01	Mercedes-Maybach 62S Coupe	604 hp	Auto	$ 5000	1	Available
PO01	Porsche 911 (991.2) Turbo S	580 hp	Auto	$ 1700	2	Available
PO02	Porsche Carrera GT	603 hp	Man	$ 9500	1	Available
PR01	Proton Satria Neo R3 Lotus	145 hp	Man	$ 250	2	Available
RN01	Renault Clio V6 Phase 2	252 hp	Man	$ 600	1	Available
TY02	Toyota GR Corolla MT	300 hp	Man	$ 380	5	Available
TY01	Toyota GR Yaris MT	280 hp	Man	$ 350	3	Available

Book a car? (y/n):

4.2 Rental Process (Update)

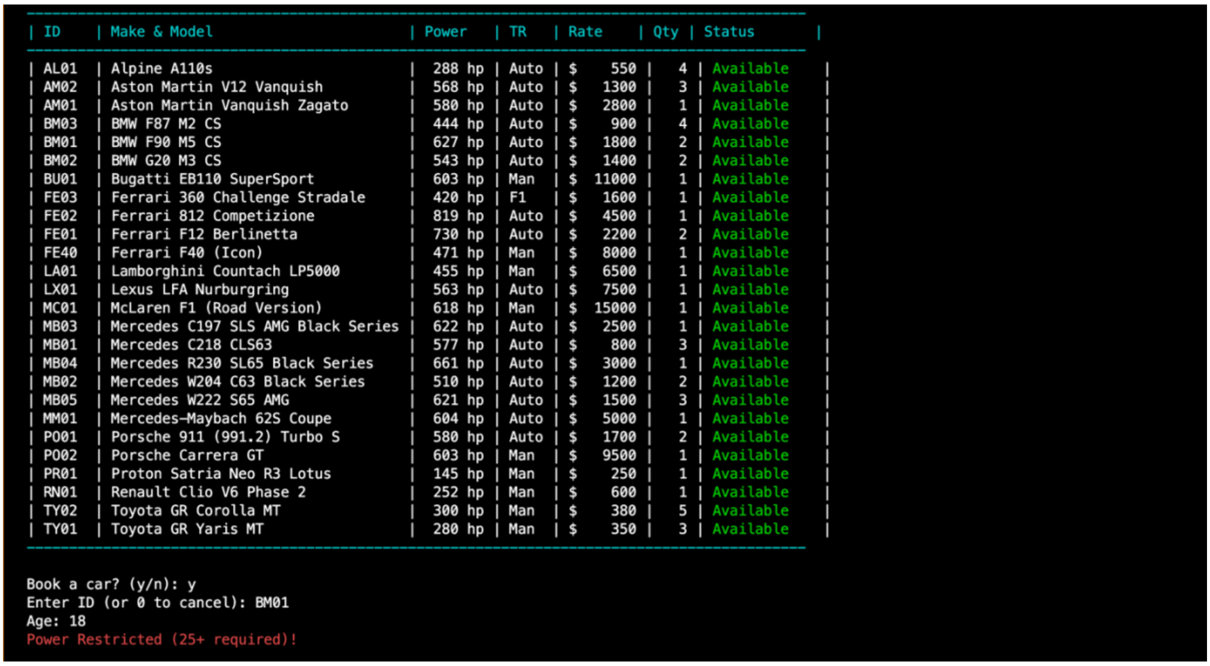
When a user rents a car:

1. The system checks availability via the Linked List.
2. It enforces an age restriction (Driver must be 25+ for >500 HP cars).
3. Stock is decremented, and the status is updated to "Rented".
4. Revenue is calculated and saved to Revenue_Report.txt.

Figure 4.2.1: Successful rental transaction output showing booking confirmation and total cost calculation.

```
-----------------------------------------------------------------------------------------------
| ID    | Make & Model                         | Power    | TR     | Rate      | Qty | Status     |
-----------------------------------------------------------------------------------------------
| AL01  | Alpine A110s                         |  288 hp | Auto | $    550 |   4 | Available  |
| AM02  | Aston Martin V12 Vanquish            |  568 hp | Auto | $   1300 |   3 | Available  |
| AM01  | Aston Martin Vanquish Zagato         |  580 hp | Auto | $   2800 |   1 | Available  |
| BM03  | BMW F87 M2 CS                        |  444 hp | Auto | $    900 |   4 | Available  |
| BM01  | BMW F90 M5 CS                        |  627 hp | Auto | $   1800 |   2 | Available  |
| BM02  | BMW G20 M3 CS                        |  543 hp | Auto | $   1400 |   2 | Available  |
| BU01  | Bugatti EB110 SuperSport             |  603 hp | Man  | $  11000 |   1 | Available  |
| FE03  | Ferrari 360 Challenge Stradale       |  420 hp | F1   | $   1600 |   1 | Available  |
| FE02  | Ferrari 812 Competizione             |  819 hp | Auto | $   4500 |   1 | Available  |
| FE01  | Ferrari F12 Berlinetta               |  730 hp | Auto | $   2200 |   2 | Available  |
| FE40  | Ferrari F40 (Icon)                   |  471 hp | Man  | $   8000 |   1 | Available  |
| LA01  | Lamborghini Countach LP5000          |  455 hp | Man  | $   6500 |   1 | Available  |
| LX01  | Lexus LFA Nurburgring                |  563 hp | Auto | $   7500 |   1 | Available  |
| MC01  | McLaren F1 (Road Version)            |  618 hp | Man  | $  15000 |   1 | Available  |
| MB03  | Mercedes C197 SLS AMG Black Series   |  622 hp | Auto | $   2500 |   1 | Available  |
| MB01  | Mercedes C218 CLS63                  |  577 hp | Auto | $    800 |   3 | Available  |
| MB04  | Mercedes R230 SL65 Black Series      |  661 hp | Auto | $   3000 |   1 | Available  |
| MB02  | Mercedes W204 C63 Black Series       |  510 hp | Auto | $   1200 |   2 | Available  |
| MB05  | Mercedes W222 S65 AMG                |  621 hp | Auto | $   1500 |   3 | Available  |
| MM01  | Mercedes-Maybach 62S Coupe           |  604 hp | Auto | $   5000 |   1 | Available  |
| PO01  | Porsche 911 (991.2) Turbo S          |  580 hp | Auto | $   1700 |   2 | Available  |
| PO02  | Porsche Carrera GT                   |  603 hp | Man  | $   9500 |   1 | Available  |
| PR01  | Proton Satria Neo R3 Lotus           |  145 hp | Man  | $    250 |   2 | Available  |
| RN01  | Renault Clio V6 Phase 2              |  252 hp | Man  | $    600 |   1 | Available  |
| TY02  | Toyota GR Corolla MT                 |  300 hp | Man  | $    380 |   5 | Available  |
| TY01  | Toyota GR Yaris MT                   |  280 hp | Man  | $    350 |   3 | Available  |
-----------------------------------------------------------------------------------------------

Book a car? (y/n): y
Enter ID (or 0 to cancel): PR01
Age: 18
Name: Adam
Phone: 0136657525
Days: 14

--- BOOKING CONFIRMED ---
Total: $3500.00
```

Figure 4.2.2: Security feature demonstrating the Age Verification logic rejecting an underage user for a high-horsepower vehicle.



ID	Make & Model	Power	TR	Rate	Qty	Status
AL01	Alpine A110s	288 hp	Auto	$ 550	4	Available
AM02	Aston Martin V12 Vanquish	568 hp	Auto	$ 1300	3	Available
AM01	Aston Martin Vanquish Zagato	580 hp	Auto	$ 2800	1	Available
BM03	BMW F87 M2 CS	444 hp	Auto	$ 900	4	Available
BM01	BMW F90 M5 CS	627 hp	Auto	$ 1800	2	Available
BM02	BMW G20 M3 CS	543 hp	Auto	$ 1400	2	Available
BU01	Bugatti EB110 SuperSport	603 hp	Man	$ 11000	1	Available
FE03	Ferrari 360 Challenge Stradale	420 hp	F1	$ 1600	1	Available
FE02	Ferrari 812 Competizione	819 hp	Auto	$ 4500	1	Available
FE01	Ferrari F12 Berlinetta	730 hp	Auto	$ 2200	2	Available
FE40	Ferrari F40 (Icon)	471 hp	Man	$ 8000	1	Available
LA01	Lamborghini Countach LP5000	455 hp	Man	$ 6500	1	Available
LX01	Lexus LFA Nurburgring	563 hp	Auto	$ 7500	1	Available
MC01	McLaren F1 (Road Version)	618 hp	Man	$ 15000	1	Available
MB03	Mercedes C197 SLS AMG Black Series	622 hp	Auto	$ 2500	1	Available
MB01	Mercedes C218 CLS63	577 hp	Auto	$ 800	3	Available
MB04	Mercedes R230 SL65 Black Series	661 hp	Auto	$ 3000	1	Available
MB02	Mercedes W204 C63 Black Series	510 hp	Auto	$ 1200	2	Available
MB05	Mercedes W222 S65 AMG	621 hp	Auto	$ 1500	3	Available
MM01	Mercedes-Maybach 62S Coupe	604 hp	Auto	$ 5000	1	Available
PO01	Porsche 911 (991.2) Turbo S	580 hp	Auto	$ 1700	2	Available
PO02	Porsche Carrera GT	603 hp	Man	$ 9500	1	Available
PR01	Proton Satria Neo R3 Lotus	145 hp	Man	$ 250	1	Available
RN01	Renault Clio V6 Phase 2	252 hp	Man	$ 600	1	Available
TY02	Toyota GR Corolla MT	300 hp	Man	$ 380	5	Available
TY01	Toyota GR Yaris MT	280 hp	Man	$ 350	3	Available

```
Book a car? (y/n): y
Enter ID (or 0 to cancel): BM01
Age: 18
Power Restricted (25+ required)!
```

4.3 Return & Maintenance (Queue Enqueue)

Returning a car does not immediately make it available. It is enqueued into the Wash Bay. This prevents dirty cars from being rented immediately.

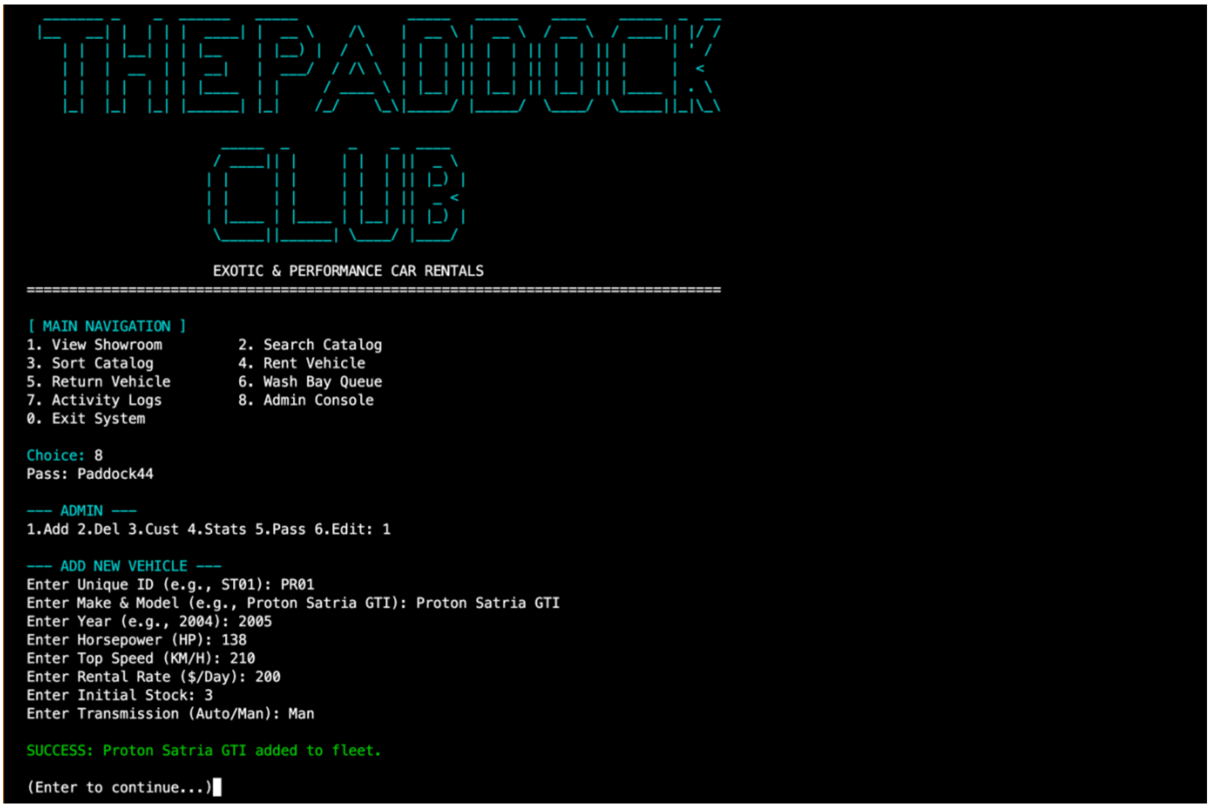
Figure 4.3.1: Vehicle Return Interface prompting the user for the Car ID to be checked back in.

Figure 4.3.2: Wash Bay Queue display demonstrating the Circular Array structure holding returned vehicles.

4.4 Admin Console (Create/Delete)

A password-protected admin panel allows for adding new cars or removing old ones. The deletion process verifies that the car is not currently rented or in the wash bay before removing it from the Linked List and Hash Table.

Figure 4.4.1: Secure Admin Console showing the vehicle addition interface and fleet modification options.



4.5 File I/O & Persistence

To ensure data is not lost when the program closes, we implemented custom file handling using C++ fstream.

- **Database:** We use db_fleet.csv and db_customers.csv to store data.
- **Parsing:** We utilized stringstream to parse comma-separated values (CSV) manually, converting strings to integers (stoi) and doubles (stod) where necessary.

Figure 4.5.1: C++ implementation of the loadFromFile function utilizing std::vector for robust CSV parsing and data persistence.

```cpp
    // Data Persistence: Loads fleet data from CSV
    void loadFromFile() {
        string path = getDataPath(fileName: "db_fleet.csv"); ifstream f(s: path.c_str());
        if (!f.is_open()) return;
        string line;
        while (getline(&is: f, &str: line)) {
            if (line.length() < 5) continue;
            vector<string> col; string part; stringstream ss(s: line);
            while (getline(&is: ss, &str: part, dlm: ',')) col.push_back(x: trim(s: part));
            if (col.size() >= 7) {
                try {
                    Car* c = new Car(i: col[0], m: col[1], y: stoi(str: col[2]), h: stoi(str: col[3]), t: stoi(str: col[4]), tr: col[5], r: stod(str: col[6]),
                                     s: col.size() > 7 ? stoi(str: col[7]) : 1, rc: col.size() > 9 ? stoi(str: col[9]) : 0);
                    c->maxStock = col.size() > 8 ? stoi(str: col[8]) : c->stock;
                    add(c);
                } catch(...) {}
            }
        }
        sortByBrand();
    }
};
```

5.0 Conclusion

The Paddock Club project successfully integrates advanced data structures to create a robust car rental management system. By implementing a Hash Table, we achieved $O(1)$ lookup times for inventory management. The use of Merge Sort ensures the system remains performant even as the fleet grows, maintaining $O(n \log n)$ sorting efficiency. Finally, the Circular Queue correctly models the real-world constraint of a wash bay service line. The project meets all functional requirements and demonstrates a deep understanding of memory management and algorithm design in C++.

6.0 Refrences

- **Multimedia University.** (2026). *Chapter 1: Introduction to Data Structure and Algorithms* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 2: Pointer and pointer variable* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 3: Stack and Queue* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 4: List and Linked List* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 5: Linked Stacks and Queues* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 6: Searching* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 7: Hashing* [Lecture slides]. Faculty of Computing and Informatics.

- **Multimedia University.** (2026). *Chapter 8: Sorting* [Lecture slides]. Faculty of Computing and Informatics.

- **DevDocs.** (2025). *std::vector - C++ Vector Library*. Retrieved from https://devdocs.io/cpp/container/vector

- **DevDocs.** (2025). *std::fstream - C++ Input/Output Library*. Retrieved from https://devdocs.io/cpp/io/basic_fstream

- **GeeksforGeeks.** (2024). *Merge Sort for Linked Lists*. Retrieved from https://www.geeksforgeeks.org/merge-sort-for-linked-list/