DEPARTMENT OF COMPUTER ENGINEERING

A. Y. 2025-26 Semester-I

MINI PROJECT REPORT

Subject: Data Structures

Group No. : 6

Title of the Project: Parking Lot Management System

Group Members:

Sr. No.	PRN No.	Name of the Student
1.	B24CE1064	Aman Gotad
2.	B24CE1069	Shruti Deshmukh
3.	B24CE1079	Srushti Salgar
4.	B24CE1085	Swapnil Waghmare

Data Structures Used:

Arrays

- Used to represent the parking slots (slots[]) and to store the vehicle numbers.
- Each index in the array corresponds to a parking slot.
- Another array slotTimes[] stores entry times (in minutes) for each parked vehicle.
- Purpose: Enables direct access and constant-time slot management.

Queue

- When all parking slots are full, vehicles are placed in a waiting queue.
- The queue is implemented using a circular array with waitFront and waitRear pointers to track insertion and removal.
- Vehicles are dequeued automatically when a slot becomes vacant.
- Purpose: Manages overflow vehicles efficiently without losing entry order.

Class

- The entire system is implemented inside the class ParkingLot.
- It encapsulates data (slots, waiting queue, rates) and behaviors (park, unpark, search, display).
- Demonstrates encapsulation, data abstraction, and modularity.
- Purpose: Simplifies maintenance and improves readability.

Mini-Project Idea:

The **Parking Lot Allotment System** aims to automate the process of allocating parking spaces to vehicles efficiently.
This system minimizes manual effort, reduces waiting time, and ensures systematic management of parking slots and waiting queues. It also automatically calculates the parking charges based on the duration of parking and generates a detailed summary for each transaction.

Input:

1. Number of Parking Slots – Total available slots.
2. Rate per Hour – The amount charged for every hour.
3. Vehicle Number – Unique identifier for each vehicle.
   Example: MH12AB1234
4. Entry and Exit Times – Given in the format HH:MMAM or HH:MMPM.
   Example: 09:30AM, 03:45PM

Output:

1. Parking Confirmation – Displays slot number and entry time.
2. Waiting Queue Message – Shows queue position if parking is full.
3. Billing Information – Displays entry time, exit time, total duration, and payment due.
4. Search Results – Shows slot number and entry time if vehicle is found.
5. Current Status Display – Lists all occupied and empty slots with waiting queue details.

Techniques Used:

1. Time Conversion Logic:
   - Functions timeToMinutes() and minutesToTime() convert between standard time format and total minutes since midnight.
   - This allows easy arithmetic for calculating parking durations.
2. Dynamic Memory Allocation:
   - Arrays are created dynamically using new based on user input.
   - Prevents wastage of memory and ensures scalability.
3. Conditional Logic:
   - Handles special cases such as when the lot or queue is full, or when duration is less than one hour.
4. Encapsulation and Abstraction:
   - Hides complex logic (time calculation, queue management) inside class methods.
   - Provides a clean and simple interface to the user.
5. Formatted Display:
   - Uses iomanip manipulators for structured and professional console output.

Program:

```
#include <iostream>
#include <string>
using namespace std;

// Convert time string to minutes
int timeToMinutes(string t) {
   int h = (t[0] - '0') * 10 + (t[1] - '0');
   int m = (t[3] - '0') * 10 + (t[4] - '0');
   char ampm = t[6];
   if (ampm == 'P' || ampm == 'p') {
      if (h != 12) {
         h = h + 12;
      }
   } else {
      if (h == 12) {
         h = 0;
      }
   }
   return h * 60 + m;
}

// Convert minutes TO time string
string minutesToTime(int mins) {
   int h = (mins / 60) % 24;
   int m = mins % 60;
   string ampm = "AM";
   if (h >= 12) {
      ampm = "PM";
   }
   if (h == 0) {
      h = 12;
   } else if (h > 12) {
      h = h - 12;
   }
   string time = "";
   if (h < 10) {
      time = time + "0";
   }
   time = time + to_string(h) + ":";
```

```
 if (m < 10) {
      time = time + "0";
   }
   time = time + to_string(m) + ampm;
   return time;
}

class ParkingLot {
   string* slots;
   int* entryTimes;
   string* waiting;
   int capacity;
   int waitCapacity;
   int front;
   int rear;
   float rate;

public:
   ParkingLot(int n, float r) {
      capacity = n;
      rate = r;
      slots = new string[capacity];
      entryTimes = new int[capacity];
      for (int i = 0; i < capacity; i++) {
         slots[i] = "EMPTY";
         entryTimes[i] = 0;
      }
      waitCapacity = capacity * 2;
      waiting = new string[waitCapacity];
      front = 0;
      rear = -1;
   }

   void park(string vehicle, int time) {
      for (int i = 0; i < capacity; i++) {
         if (slots[i] == "EMPTY") {
            slots[i] = vehicle;
            entryTimes[i] = time;
            cout << "\nVehicle " << vehicle << " parked in slot " << i + 1 << endl;
            return;
         }
      }
```

```
if (rear - front + 1 < waitCapacity) {
        rear++;
        waiting[rear % waitCapacity] = vehicle;
        cout << "\nParking full. Vehicle " << vehicle << " added to waiting queue." << endl;
    } else {
        cout << "\nParking and waiting full!" << endl;
    }
}

void unpark(string vehicle, int exitTime) {
    for (int i = 0; i < capacity; i++) {
        if (slots[i] == vehicle) {
            int duration = exitTime - entryTimes[i];
            if (duration <= 0) {
                duration = 60;
            }
            float hours = (float)duration / 60;
            if (hours < 1) {
                hours = 1;
            }
            float amount = hours * rate;
            cout << "\nVehicle " << vehicle << " unparked from slot " << i + 1 << endl;
            cout << "Entry time: " << minutesToTime(entryTimes[i]) << " | Exit time: " << minutesToTime(exitTime) << endl;
            cout << "Parking hours: " << hours << endl;
            cout << "Amount to pay: Rs " << amount << endl;

            slots[i] = "EMPTY";
            entryTimes[i] = 0;

            if (front <= rear) {
                string next = waiting[front % waitCapacity];
                front++;
                slots[i] = next;
                entryTimes[i] = exitTime;
                cout << "Next vehicle " << next << " moved from waiting to slot " << i + 1 << endl;
            }
            return;
        }
    }
    cout << "\nVehicle not found!" << endl;
}
```

```
 void search(string vehicle) {
      for (int i = 0; i < capacity; i++) {
         if (slots[i] == vehicle) {
            cout << "\nVehicle " << vehicle << " found in slot " << i + 1;
            cout << " (Entry time: " << minutesToTime(entryTimes[i]) << ")" << endl;
            return;
         }
      }
      cout << "\nVehicle not found!" << endl;
   }

   void show() {
      cout << "\n--- Parking Status ---" << endl;
      for (int i = 0; i < capacity; i++) {
         cout << "Slot " << i + 1 << ": " << slots[i];
         if (slots[i] != "EMPTY") {
            cout << " (" << minutesToTime(entryTimes[i]) << ")";
         }
         cout << endl;
      }
      cout << "\nWaiting : ";
      if (front > rear) {
         cout << "Empty" << endl;
      } else {
         for (int i = front; i <= rear; i++) {
            cout << waiting[i % waitCapacity] << " ";
         }
         cout << endl;
      }
   }

   ~ParkingLot() {
      delete[] slots;
      delete[] entryTimes;
      delete[] waiting;
   }
};

int main() {
   int n, choice;
   float rate;
   string vehicleNum, timeStr;
```

```
cout << "Enter number of parking slots: ";
  cin >> n;
  cout << "Enter rate per hour: ";
  cin >> rate;

  ParkingLot lot(n, rate);

  do {
    cout << "\n1. Park Vehicle\n2. Unpark Vehicle\n3. Search Vehicle\n4. Show Parking Status\n5. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    if (choice == 1) {
      cout << "Enter vehicle number: ";
      cin >> vehicleNum;
      cout << "Enter entry time (HH:MM AM/PM): ";
      cin.ignore();
      getline(cin, timeStr);
      lot.park(vehicleNum, timeToMinutes(timeStr));
    } else if (choice == 2) {
      cout << "Enter vehicle number: ";
      cin >> vehicleNum;
      cout << "Enter exit time (HH:MM AM/PM): ";
      cin.ignore();
      getline(cin, timeStr);
      lot.unpark(vehicleNum, timeToMinutes(timeStr));
    } else if (choice == 3) {
      cout << "Enter vehicle number: ";
      cin >> vehicleNum;
      lot.search(vehicleNum);
    } else if (choice == 4) {
      lot.show();
    } else if (choice == 5) {
      cout << "Thank you for using the parking system.\n";
    } else {
      cout << "Invalid input, please try again." << endl;
    }
  } while (choice != 5);

  return 0;
}
```

Output:

Option 1 – Park

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   dsa
PS C:\Users\Gotad\OneDrive\Desktop\DSA MINI> g++ dsa.cpp -o dsa.exe
PS C:\Users\Gotad\OneDrive\Desktop\DSA MINI> ./dsa.exe
Enter number of parking slots: 3
Enter rate per hour: 10

1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 1
Enter vehicle number: MH12XA4566
Enter entry time (HH:MM AM/PM): 01:00 PM

Vehicle MH12XA4566 parked in slot 1

1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 1
Enter vehicle number: MH12SA4510
Enter entry time (HH:MM AM/PM): 02:00 PM

Vehicle MH12SA4510 parked in slot 2

1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 1
Enter vehicle number: MH12XA4567
Enter entry time (HH:MM AM/PM): 05:00 PM

Vehicle MH12XA4567 parked in slot 3
```

Parking full so new vehicle waiting list

```
1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 1
Enter vehicle number: MH41SA2156
Enter entry time (HH:MM AM/PM): 06:00 PM

Parking full. Vehicle MH41SA2156 added to waiting list.
```

Option 2 -Unpark vehicle

```
1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 2
Enter vehicle number: MH12XA4567
Enter exit time (HH:MM AM/PM): 06:00 PM

Vehicle MH12XA4567 unparked from slot 3
Entry time: 05:00PM | Exit time: 06:00PM
Parking hours: 1
Amount to pay: Rs 10
Next vehicle MH82XA4002 moved from waiting to slot 3
```

Option 3 – Search vehicle

```
1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 3
Enter vehicle number: MH12XA4566

Vehicle MH12XA4566 found in slot 1 (Entry time: 01:00PM)
```

Option 4 – Parking status display

```
1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 4

--- Parking Status ---
Slot 1: MH12XA4566 (01:00PM)
Slot 2: MH12SA4510 (02:00PM)
Slot 3: MH82XA4002 (06:00PM)

Waiting : Empty
```

Option 5 – Exit

```
1. Park Vehicle
2. Unpark Vehicle
3. Search Vehicle
4. Show Parking Status
5. Exit
Enter your choice: 5
Thank you for using the parking system.
```

Github Link : https://github.com/AmansWork-24-28/Data-Structures-/commit/22c7619d14757e4251c9d291cc45dbe986dced9c

Analysis:

Time Complexity:

Operation	Data Structure	Best Case	Average Case	Worst Case
Park Vehicle	Array + Queue	O(1)	O(n)	O(n)
Unpark Vehicle	Array + Queue	O(1)	O(n)	O(n)
Search Vehicle	Array	O(1)	O(n)	O(n)
Show Status	Array + Queue	O(n)	O(n)	O(n)

Space Complexity:

Component	Space	Description
Parking Slots Array	O(n)	Stores vehicle numbers
Entry Time Array	O(n)	Stores entry time of vehicles
Waiting Queue	O(2n)	Stores waiting vehicles
Auxiliary Variables	O(1)	Temporary storage and counters

Total Space Complexity: O(n)