Amusement Park Management System

Submitted By

Student Name	Student ID
Mahadi Rahman Jihad	251-15-008
Tasfia Mozumder Aupy	251-15-935
Tahmid Ibne Mofazzol	251-15-548
Labiba Tasneem	251-15-484
Tahmid Hasan Sadib	251-15-084

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE124: Data Structure lab in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

December 12,2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Md. Jakaria Zobair**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

_____

Md. Jakaria Zobair
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

<table>
<tr><td colspan="2">Mahadi Rahman Jihad
Student ID:251-15-008
Dept. of CSE, DIU</td></tr>
<tr><td>Tasfia Mozumder Aupy
Student ID:251-15-935
Dept. of CSE, DIU</td><td>Tahmid Ibne Mofazzol
Student ID:251-15-548
Dept. of CSE, DIU</td></tr>
<tr><td>Labiba Tasneem
Student ID:251-15-484
Dept. of CSE, DIU</td><td>Tahmid Hasan Sadib
Student ID:251-15-084
Dept. of CSE, DIU</td></tr>
</table>

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:

Table 1: Course Outcome Statements

CO's	Statements
CO1	Design the concept of fundamentals of data structure for solving real-life problem having complex engineering attributes.
CO2	Solve a real-life problem having application of abstract data type created within the scope of complex engineering problem solving with the apply modern tools.
CO3	Apply Data Structure to solve real world problems member in diverse team.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO3	PO1,P02,C3	KP5	EP1
CO2	PO5	P2,C3	KP6	EP2,EP6
CO3	PO9	P3,A4	N/A	N/A

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Chapter 1

Introduction

This chapter provides an overview of the amusement park management problem, the motivation behind the project, and the objectives guiding the system's design. It establishes the background context and identifies the gaps that the proposed solution aims to address.

1.1 Introduction

Amusement parks are complex systems involving continuous interaction between visitors, staff, and operational resources. Managing these workflows manually often results in inefficiencies such as long queues, inconsistent locker allocation, slow restaurant service, and unstructured parking management. These shortcomings negatively affect visitor satisfaction and overall operational performance. To address these challenges, this project presents a **data-structure-based Amusement Park Management System**, implemented using the C programming language.

The primary purpose of this system is to demonstrate how classical linear data structures such as **queues, stacks, and linked lists** can be utilized to model and automate real operational flows. Each subsystem of the amusement park—Fantasy Kingdom ride queue, Water Kingdom locker allocation, restaurant operations, and parking—has been mapped to an appropriate data structure, allowing the system to function efficiently and predictably. This transformation of theoretical concepts into practical mechanisms outlines the true engineering value of data structure knowledge.

This project serves a dual purpose. First, it provides an operational model that improves amusement park management. Second, and more importantly, it allows students to deeply understand how abstract data structures govern real-world software system behavior. The focus is not only on implementation but also on **modeling**, **design reasoning**, **algorithmic decision-making**, and **engineering justification**.

1.2 Motivation

Amusement parks are entertainment environments where visitor experience depends heavily on operational smoothness. Long wait times, unorganized locker distribution, delayed food orders, and chaotic parking systems are common issues in poorly managed parks. These problems arise due to lack of automation and structured workflow design.

From an academic perspective, students often struggle to connect data structure theory with real system behavior. This project provides a real-world scenario where queues, stacks, and linked lists are essential tools—not abstract ideas. By building a functioning system, students gain practical insight into problem decomposition, algorithm design, and data organization.

The motivation, therefore, is twofold:

1. **To improve amusement park service quality** through structured automation.
2. **To bridge the gap between theory and practice** by applying core data structure concepts in an actual engineering scenario.

1.3 Problem Statement

Amusement parks such as Fantasy Kingdom experience significant operational pressure during weekends and holidays. When these processes are handled manually, several issues arise, including:

- Inefficient ride queue management
- Locker mismanagement due to lack of a structured allocation system
- Restaurant delays caused by unordered or inconsistent request handling
- Parking disorder resulting from unplanned sequence-based assignment
- Human errors in ticket validation, offer calculation, and visitor handling

As visitor numbers increase, these challenges become more severe, leading to long waiting times and reduced service quality. Without an automated, data-structure-driven system, managing visitor flow and resource distribution becomes extremely difficult. Therefore, a structured computational solution is essential to streamline Fantasy Kingdom's operational activities and minimize inefficiencies.

1.4 Objectives

The primary objectives of this project are:

1. To design and implement a structured amusement park management system using data structures.
2. To apply **queues** for ride and restaurant order management, **stacks** for locker allocation, and **linked lists** for dynamic menu handling(for locker system,didn't applied it in my system manually).
3. To simulate real amusement park operations in a modular, scalable, and maintainable way.
4. To evaluate the system in terms of performance, predictability, and correctness.
5. To ensure alignment with the course outcomes (CO1–CO3) of CSE124.
6. To demonstrate how engineering design principles influence system behavior.

1.5 Feasibility Study

Technical Feasibility

The entire system is built in C, requiring only standard header files. Data structures used are simple, efficient, and memory-friendly. No complex external libraries or high-end systems are needed.

Operational Feasibility

The system's workflow mirrors real amusement park operations closely. Queue logic maps directly to ride and restaurant flows, stack logic to locker handling, and linked list logic to restaurant menus. This makes the system operable and realistic.

Economic Feasibility

The project incurs no cost except the use of a standard computer. The system can be deployed in actual parks with minimal hardware investment.

1.6 Gap Analysis

There is a clear gap between how amusement parks operate manually and how they could operate through automation:

Existing Issue	Gap Identified	DS-Based Solution
Long ride queues	No structured order	FIFO Queue
Locker confusion (didn't applied)	No reversible allocation	Stack
Restaurant delays	No ordered processing	Queue
Menu updates difficult	Static structure	Linked List
Parking unorganized	No sequence-based assignment	Queue

1.7 Project Outcome

After completing this project, the following outcomes were achieved:

- A complete Amusement Park Management System was developed using C and fundamental data structures.
- Queue, stack(not manually), and linked list structures were effectively applied to model real operational behaviors.
- The system successfully automated ride queues, locker allocation(working on it), restaurant order handling, and parking management.
- Modular design principles were followed, making the system easy to test, maintain, and extend.
- The implementation demonstrates the practical relevance of data structures in solving real-world problems.
- The project strengthened understanding of algorithm design, memory management, and structured programming.
- All course outcomes (CO1–CO3) related to problem analysis, DS application, and teamwork were achieved.
- The project enhanced students' ability to translate theoretical concepts into a working engineering solution.

Chapter 2

Proposed Methodology/Architecture

This chapter presents the methodology and system architecture for the Amusement Park Management System. It outlines key requirements, core module workflows, and the design principles used to structure the system. The chapter also highlights the implementation approach adopted to ensure efficient and user-oriented operation.

2.1 Requirement Analysis & Design Specification

This section outlines the essential hardware and software requirements needed to develop and operate the Amusement Park Management System, along with the overall design specifications that define its structural and functional behavior. The objective is to ensure that the system performs reliably, manages multiple operations efficiently, and provides a seamless experience to both users and system operators.

Requirement Analysis for the Project:

Hardware Requirements:

- A computer capable of running C programs efficiently
- Minimum: Dual-core processor, 4GB RAM
- Recommended: 8GB RAM and higher processing speed for smoother execution

Software Requirements:

- A C compiler (e.g., GCC, Clang)
- A code editor or IDE (e.g., Visual Studio Code, Code::Blocks)
- Standard C libraries for input/output and data structure operations
- Optional libraries for enhanced UI or formatting, if implemented

Design Specification:

Core Components:

Input Module:
Handles user interactions such as ride selection, booking operations, locker requests, restaurant order placement, and parking-related actions. Ensures proper input validation and structured menu-driven navigation.

Processing Module:
Implements the internal logic for all amusement park operations. This includes queue management

for rides, resource allocation for lockers and parking, order sequencing for restaurants, and execution of data structure–based algorithms used throughout the system.

Output Module:
Generates system responses and displays results on the console. This includes confirmation messages, queue statuses, ride availability updates, locker allocation details, and other operational outputs.

System Flow:
User inputs → Operational logic execution (queues, stacks, or other structures) → Updated system state and console output.

2.1.1 Overview

Purpose:
The Amusement Park Management System is designed to automate and streamline key operational activities within an amusement park. It ensures efficient handling of ride queues, locker allocation, restaurant order management, and parking coordination through structured data-processing techniques.

Key Features:

- Manages ride queues using data structures to maintain fairness and optimize waiting time.
- Handles locker allocation, restaurant order sequencing, and parking slot assignment in a systematic manner.

Implementation:

- Developed using the C programming language with a focus on efficient algorithm design and memory management.
- Utilizes appropriate data structures such as queues, stacks, and arrays to support different operational modules.
- Follows a modular architecture that separates input handling, processing logic, and output generation for maintainability.

Operational Value:

- Reduces manual workload and minimizes operational errors by automating routine park activities.
- Enhances user experience by ensuring organized ride management and faster service response times.
- Supports system operators with structured processes that improve overall park workflow efficiency.

Future Potential:
Offers a foundation for integrating advanced features such as digital ticketing, real-time ride

analytics, multi-user management, and graphical interfaces for improved usability in future iterations.

2.1.2 Proposed Methodology/ System Design

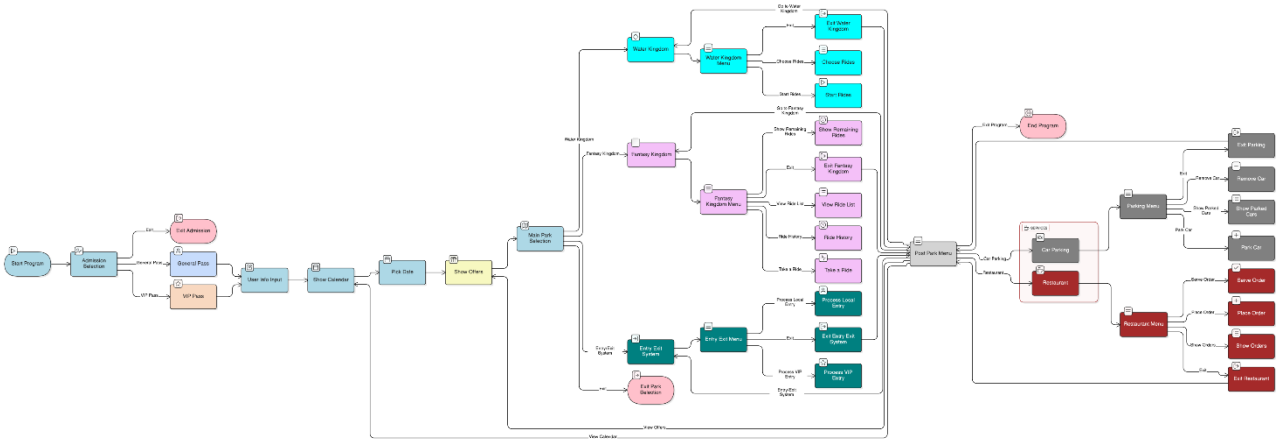


Figure 2.1: Implementation Diagram

Figure 2.1: Overall System Workflow and Module Interaction Diagram

This flowchart represents an **Amusement Park System Management** project where all park operations are controlled through a single integrated flow. The system manages visitor admission, date-based offers, and access to multiple park zones using a shared visitor profile and centralized control. After entry, visitors can navigate between different functional modules such as Fantasy Kingdom, Water Kingdom, entry–exit control, restaurant, and parking, each operating independently but connected to the same main workflow. The design highlights a modular structure in which ride management, service handling, and visitor flow are simulated using basic data structures, allowing the entire park to function as a coordinated and realistic management system.

2.1.3 UI Design

The UI design emphasizes clarity, ease of navigation, and user-friendly interaction to help users efficiently perform amusement-park-related operations such as selecting rides, managing lockers, placing restaurant orders, and handling parking tasks. The interface is entirely text-based, ensuring compatibility across different systems and simplicity in execution.

Main Menu and Navigation:
Users are presented with a structured main menu that allows them to select major modules, including Ride Management, Locker Services, Restaurant Queue, and Parking System. Each module contains clearly defined sub-menus that guide users through available operations such as joining a ride queue, requesting a locker, placing an order, or reserving a parking slot.

Real-Time Status Updates:
After each operation, the system displays updated information in a readable text-based format.

Examples include current ride queue length, locker availability, order position in the restaurant queue, or remaining parking slots. These real-time outputs help users understand the system's current state and make informed decisions.

Instructions and Error Handling:
The UI provides step-by-step prompts to guide users through each workflow. Clear messaging is used to confirm successful actions or notify users of invalid inputs, unavailable resources, or system constraints. Basic error-handling ensures that users are redirected safely to appropriate menus without disrupting the program flow.

2.2 Overall Project Plan

The development of the **Amusement Park Management System** was carried out in a phased and systematic manner to ensure proper requirement understanding, efficient system design, and accurate implementation of data structure concepts. The overall project plan was divided into several stages, each with clearly defined objectives, tasks, and deliverables.

1. Requirement Collection & Field Study (Week 1)

Objective:
To understand real-world amusement park operations and gather authentic system requirements from stakeholders.

Tasks:

- Visit **Fantasy Kingdom** to observe amusement park operations.
- Interact with stakeholders and staff to collect information on:
    - Visitor entry and exit flow.
    - Ride queue management.
    - Locker allocation and usage.
    - Restaurant order handling.
    - Parking management.
- Identify operational problems in manual systems such as long queues and inefficient tracking.
- Translate real-world workflows into system requirements.

Deliverables:

- Stakeholder requirement notes.
- Real-world workflow analysis.
- Initial problem statement and requirement list.

2. Planning & Requirement Analysis (Week 2 – Week 3)

Objective:
To analyze collected data and finalize the project scope based on data structure applicability.

Tasks:

- Analyze functional and non-functional requirements.
- Identify suitable data structures:
    - **Queue** for ride lines, entry gates, and restaurant orders.
    - **Stack** for ride history, locker usage, and plate management.
- Define project objectives aligned with the CSE124 syllabus.
- Finalize system modules and boundaries.
- Select tools and environment (C language, GCC compiler).

Deliverables:

- Finalized requirement specification.
- List of selected data structures and their use cases.
- Project scope document.

3. System Design Phase (Week 4 – Week 5)

Objective:
To design the system architecture and logical flow of modules.

Tasks:

- Design overall system architecture (input, processing, and output modules).
- Create flowcharts for major subsystems:
    - Fantasy Kingdom ride management.
    - Water Kingdom locker and ride system.
    - Restaurant management system.
    - Parking management system.
- Design data flow between modules.
- Plan menu-driven, console-based user interface.

Deliverables:

- System architecture diagrams.
- Flowcharts and logical design documentation.
- UI design outline.

4. Development & Implementation Phase (Week 6 – Week 9)

Objective:
To implement the system using C programming and apply data structure concepts effectively.

Tasks:

- Set up development environment.
- Implement core data structure operations:
    - Enqueue, dequeue for queues.
    - Push, pop for stacks.
- Develop individual modules:
    - Entry and exit management.
    - Ride queue handling.
    - Locker allocation.
    - Restaurant order processing.
    - Parking management.
- Integrate all modules into a single system.
- Handle user inputs and error conditions.

Deliverables:

- Fully functional amusement park modules.
- Integrated C-based system.
- Initial working version of the project.

5. Testing & Debugging Phase (Week 10 – Week 11)

Objective:
To ensure system correctness, stability, and logical accuracy.

Tasks:

- Perform unit testing on individual modules.
- Test queue and stack operations under different scenarios.
- Identify and fix logical and runtime errors.
- Validate system behavior with real-world scenarios.
- Improve user interaction and input validation.

Deliverables:

- Debugged and optimized system.
- Test results and observations.

6. Documentation & Refinement (Week 12 – Week 13)

Objective:
To prepare complete project documentation and refine system performance.

Tasks:

- Write project report chapters:
    - Introduction.
    - Methodology.

    - System design.
    - Implementation.
    - Results and discussion.
- Update code comments and improve readability.
- Ensure alignment with academic standards.

Deliverables:

- Complete project report draft.
- Well-documented source code.

7. Finalization, Presentation & Submission (Week 14 – Week 15)

Objective:
To finalize the project and present it for academic evaluation.

Tasks:

- Prepare presentation slides explaining:
    - Motivation and objectives.
    - Data structure usage.
    - System workflow.
    - Outputs and learning outcomes.
- Conduct final system review.
- Submit project code, documentation, and report.

Deliverables:

- Project presentation slides.
- Final project submission.

Timeline Overview

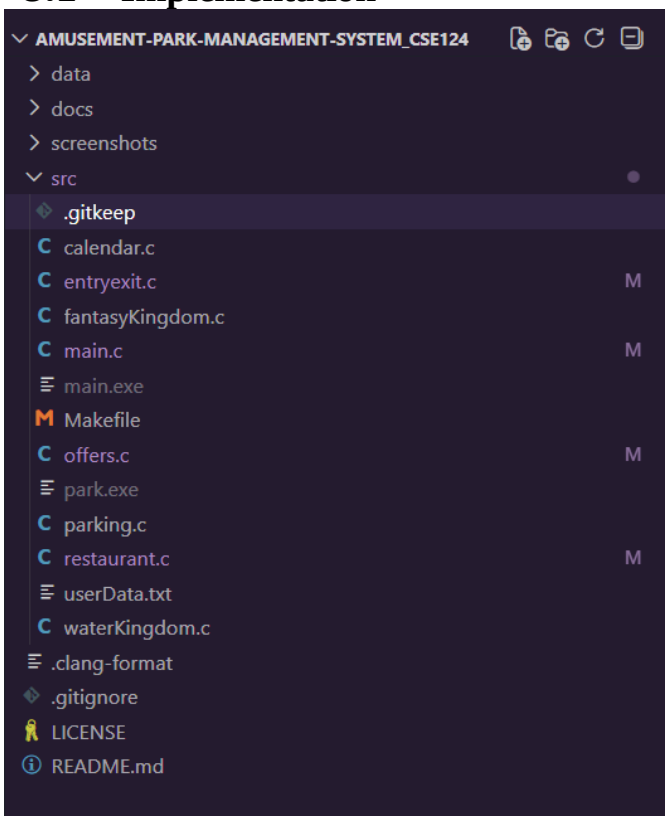
- **Week 1:** Requirement Collection & Field Study
- **Weeks 2–3:** Planning & Requirement Analysis
- **Weeks 4–5:** System Design
- **Weeks 6–9:** Development & Implementation
- **Weeks 10–11:** Testing & Debugging
- **Weeks 12–13:** Documentation & Refinement
- **Weeks 14–15:** Presentation & Submission

Chapter 3

Implementation and Results

This chapter explains the implementation details of each subsystem along with the algorithms, data structures, and code components used to build the system. It also presents testing outcomes, performance analysis, and a detailed discussion of the system's behavior

3.1 Implementation



main.c M

src > main.c > main()

```c
#include <stdio.h>
#include <stdlib.h>

void printDecember2025Calendar();
int chooseDate2025();

void showOffer2025(int date);

void fantasyKingdomWelcoming();
void waterKingdomWelcoming(int age, float height);

void initEntryExitSystem();
void enterAsLocal(char name[], int age, float height);
void enterAsVIP(char name[], int age, float height);
void entryExitSystem();
void restaurant();
void carParking();

extern char visitorNameGlobal[];
extern int visitorAge;
extern float visitorHeight;
extern int childOnly;
extern int childRidePermission;

int main() {
    initEntryExitSystem();

```

main.c M

src > main.c > main()

```c
    int passType;

    printf("============================================================\n");
    printf("                FNTASY KINGDOM AMUSEMENT PARK\n");
    printf("============================================================\n");
    printf("Select Admission Type:\n");
    printf("1) General Pass\n");
    printf("2) VIP Pass\n");
    printf("3) Exit\n");
    printf("> ");

    scanf("%d", &passType);

    int age;
    float height;
    char name[50];

    printf("Enter your name: ");
    scanf(" %[^\n]", name);

    printf("Enter your Age: ");
    scanf("%d", &age);

    printf("Enter your Height (ft): ");
    scanf("%f", &height);

```

main.c M

src > main.c > main()

```c
    if (passType == 1) {
        enterAsLocal(name, age, height);
    } else if (passType == 2) {
        enterAsVIP(name, age, height);
    } else {
        printf("Exiting.\n");
        return 0;
    }

    printDecember2025Calendar();

    int selectedDate = chooseDate2025();
    printf("\nSelected date: December %d, 2025\n", selectedDate);

    showOffer2025(selectedDate);

startParkChoice:

    printf("\n================ PARK SELECTION ================\n");
    printf("1) Fantasy Kingdom\n");
    printf("2) Water Kingdom\n");
    printf("3) Entry/Exit System\n");
    printf("4) Exit Program\n");
    printf("================================================\n");
    printf("> ");
```

main.c M

src > main.c > main()

```c
    int park;
    scanf("%d", &park);

    if (park == 1) {
        fantasyKingdomWelcoming();
    } else if (park == 2) {
        waterKingdomWelcoming(age, height);

    } else if (park == 3) {
        entryExitSystem();
    } else if (park == 4) {
        printf("Goodbye!\n");
        return 0;
    } else {
        printf("Invalid option.\n");
        goto startParkChoice;
    }

postParkMenu:

    printf("\n================================================================\n");
    printf("               WHAT DO YOU WANT TO DO NEXT?\n");
    printf("================================================================\n");
    printf("1) Go to Fantasy Kingdom\n");
    printf("2) Go to Water Kingdom\n");
    printf("3) Entry/Exit System\n");
```

```
106     printf("4) View Calendar\n");
107     printf("5) View Offers\n");
108     printf("6) Go to restaurant\n");
109     printf("7) Park your vehicle\n");
110     printf("8) Exist program\n");
111     printf("=================================================================\n");
112     printf("> ");
113
114     int nextAction;
115     scanf("%d", &nextAction);
116
117     if (nextAction == 1) {
118         fantasyKingdomWelcoming();
119         goto postParkMenu;
120     } else if (nextAction == 2) {
121         waterKingdomWelcoming(age, height);
122         goto postParkMenu;
123     } else if (nextAction == 3) {
124         entryExitSystem();
125         goto postParkMenu;
126     } else if (nextAction == 4) {
127         printDecember2025Calendar();
128         goto postParkMenu;
129     } else if (nextAction == 5) {
130         showOffer2025(selectedDate);
131         goto postParkMenu;
132     } else if (nextAction == 6) {
```

```
132     } else if (nextAction == 6) {
133         restaurant();
134         goto postParkMenu;
135     } else if (nextAction == 7) {
136         carParking();
137     } else if (nextAction == 8) {
138         printf("Goodbye!\n");
139         return 0;
140     }
141
142     else {
143         printf("Invalid option.\n");
144         goto postParkMenu;
145     }
146
147     return 0;
148 }
```

calendar.c

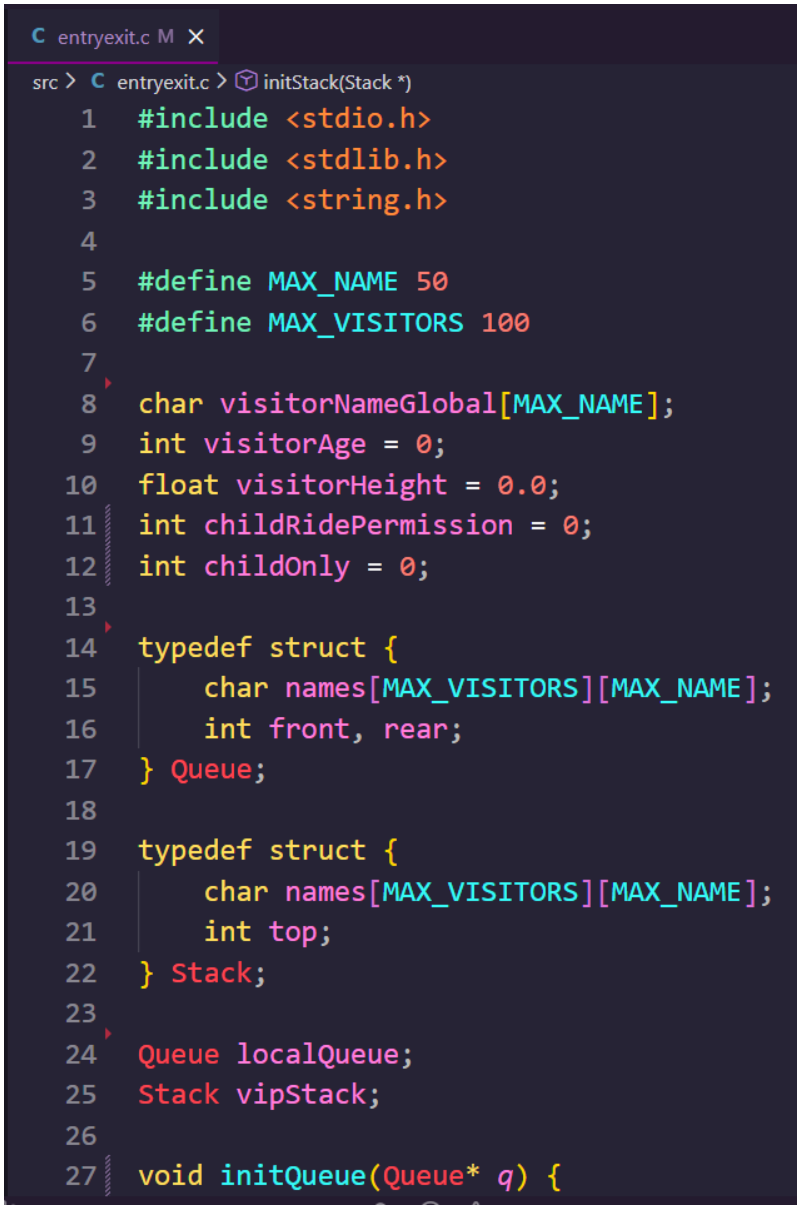
src > calendar.c > printDecember2025Calendar()

```c
#include <stdio.h>

void printDecember2025Calendar() {
    printf("  Please select your suitable date below:\n");
    printf("=============================================\n\n");

    printf("                December 2025\n");
    printf("---------------------------------------------\n");
    printf("Su  Mo  Tu  We  Th  Fr  Sa\n");

    int startDay = 1;
    int day;

    for (int i = 0; i < startDay; i++)
        printf("    ");

    for (day = 1; day <= 31; day++) {
        printf("%2d  ", day);

        if ((day + startDay) % 7 == 0)
            printf("\n");
    }

    printf("\n---------------------------------------------\n");
}
```

calendar.c

src > calendar.c > printDecember2025Calendar()

```c
int chooseDate2025() {
    int date;

    while (1) {
        printf("Select a date (1-31): ");
        if (scanf("%d", &date) != 1) {
            while (getchar() != '\n');
            printf("Invalid input! Please enter a number.\n");
            continue;
        }

        if (date >= 1 && date <= 31)
            return date;

        printf("Invalid date! Please choose between 1 and 31.\n");
    }
}
```


entryexit.c M
src > entryexit.c > initStack(Stack *)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME 50
#define MAX_VISITORS 100
char visitorNameGlobal[MAX_NAME];
int visitorAge = 0;
float visitorHeight = 0.0;
int childRidePermission = 0;
int childOnly = 0;
typedef struct {
char names[MAX_VISITORS][MAX_NAME];
int front, rear;
} Queue;
typedef struct {
char names[MAX_VISITORS][MAX_NAME];
int top;
} Stack;
Queue localQueue;
Stack vipStack;
void initQueue(Queue* q) {

entryexit.c M

src > entryexit.c > initStack(Stack *)

```c
void initQueue(Queue* q) {
    q->front = q->rear = -1;
}

int isQueueFull(Queue* q) {
    return q->rear == MAX_VISITORS - 1;
}

int isQueueEmpty(Queue* q) {
    return q->front == -1;
}

void enqueue(Queue* q, const char* name) {
    if (isQueueFull(q)) {
        printf("Local Queue is full!\n");
        return;
    }

    if (isQueueEmpty(q))
        q->front = 0;

    q->rear++;
    strcpy(q->names[q->rear], name);
}

char* dequeue(Queue* q) {
    if (isQueueEmpty(q))
```

entryexit.c M

src > entryexit.c > ...

```c
char* dequeue(Queue* q) {
    if (isQueueEmpty(q))
        return NULL;

    char* name = q->names[q->front];

    if (q->front == q->rear)
        initQueue(q);
    else
        q->front++;

    return name;
}

void initStack(Stack* s) {
    s->top = -1;
}

int isStackFull(Stack* s) {
    return s->top == MAX_VISITORS - 1;
}

int isStackEmpty(Stack* s) {
    return s->top == -1;
}
```

entryexit.c M

src > entryexit.c > ...

```c
void push(Stack* s, const char* name) {
    if (isStackFull(s)) {
        printf("VIP Stack is full!\n");
        return;
    }
    s->top++;
    strcpy(s->names[s->top], name);
}

char* pop(Stack* s) {
    if (isStackEmpty(s))
        return NULL;

    return s->names[s->top--];
}

void enterAsLocal(char name[], int age, float height) {
    strcpy(visitorNameGlobal, name);
    visitorAge = age;
    visitorHeight = height;

    enqueue(&localQueue, name);

    printf("%s added to Local Queue (ENTRY Gate)\n", name);
}

void enterAsVIP(char name[], int age, float height) {
```

entryexit.c M

src > entryexit.c > ...

```c
void enterAsVIP(char name[], int age, float height) {
    strcpy(visitorNameGlobal, name);
    visitorAge = age;
    visitorHeight = height;

    push(&vipStack, name);

    printf("✓ %s added to VIP Stack (EXIT Gate Entry)\n\n", name);
}

void processLocalEntry() {
    char* name = dequeue(&localQueue);

    if (!name) {
        printf("No local visitors waiting.\n");
        return;
    }
    printf("→ Local Visitor Entered: %s\n", name);
}

void processVIPEntry() {
    char* name = pop(&vipStack);

    if (!name) {
        printf("No VIP visitors waiting.\n");
        return;
```

entryexit.c M

src > entryexit.c > ...

```c
void processVIPEntry() {
    char* name = pop(&vipStack);

    if (!name) {
        printf("No VIP visitors waiting.\n");
        return;
    }
    printf("➡ VIP Visitor Entered: %s\n", name);
}

void entryExitSystem() {
    int choice;

    while (1) {
        printf("\n============================\n");
        printf(" AMUSEMENT PARK ENTRY SYSTEM\n");
        printf("============================\n");
        printf("1. Process Next Local Entry\n");
        printf("2. Process Next VIP Entry\n");
        printf("3. Exit This Module\n");
        printf("> ");

        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

entryexit.c M

src > entryexit.c > ...

```c
void entryExitSystem() {
    while (1) {

        switch (choice) {
            case 1:
                processLocalEntry();
                break;
            case 2:
                processVIPEntry();
                break;
            case 3:
                return;
            default:
                printf("Invalid option!\n");
        }
    }
}

void initEntryExitSystem() {
    initQueue(&localQueue);
    initStack(&vipStack);
}
```

fantasyKingdom.c

src > fantasyKingdom.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAME_LEN 100
#define MAX_RIDE_NAME 64
#define MAX_HISTORY 100
#define CHILD_RIDE_FARE 40

typedef struct QueueNode {
    char visitorName[MAX_NAME_LEN];
    struct QueueNode *next;
} QueueNode;

typedef struct {
    QueueNode *head;
    QueueNode *tail;
    int length;
} Queue;

typedef struct {
    char items[MAX_HISTORY][MAX_RIDE_NAME];
    int top;
} RideStack;

typedef struct {
    char name[MAX_RIDE_NAME];
```

fantasyKingdom.c

src > fantasyKingdom.c > ...

```c
typedef struct {
    int durationMinutes;
    Queue queue;
} Ride;

typedef struct Category {
    char name[40];
    Ride *rides;
    int rideCount;
    struct Category *left;
    struct Category *right;
} Category;

void FK_initQueue(Queue *q) {
    q->head = q->tail = NULL;
    q->length = 0;
}

void FK_enqueue(Queue *q, const char *name) {
    QueueNode *node = malloc(sizeof(QueueNode));
    if (!node) return;

    strncpy(node->visitorName, name, MAX_NAME_LEN - 1);
    node->visitorName[MAX_NAME_LEN - 1] = '\0';
    node->next = NULL;

    if (!q->tail) q->head = q->tail = node;
```

fantasyKingdom.c

src > fantasyKingdom.c > ...

```
45  void FK_enqueue(Queue *q, const char *name) {
54      else {
55          q->tail->next = node;
56          q->tail = node;
57      }
58
59      q->length++;
60  }
61
62  int FK_dequeue(Queue *q, char *outName) {
63      if (!q->head) return 0;
64
65      QueueNode *temp = q->head;
66      strncpy(outName, temp->visitorName, MAX_NAME_LEN);
67      q->head = q->head->next;
68      if (!q->head) q->tail = NULL;
69
70      free(temp);
71      q->length--;
72      return 1;
73  }
74
75  void FK_initStack(RideStack *s) {
76      s->top = -1;
77  }
78
79  void FK_push(RideStack *s, const char *rideName) {
```

fantasyKingdom.c M

src > fantasyKingdom.c > adultRides

```
79  void FK_push(RideStack *s, const char *rideName) {
80      if (s->top < MAX_HISTORY - 1) {
81          s->top++;
82          strncpy(s->items[s->top], rideName, MAX_RIDE_NAME - 1);
83          s->items[s->top][MAX_RIDE_NAME - 1] = '\0';
84      }
85  }
86
87  void FK_printStack(RideStack *s) {
88      if (s->top < 0) {
89          printf("  No rides taken yet.\n");
90          return;
91      }
92      for (int i = s->top; i >= 0; i--) {
93          printf("  %d) %s\n", s->top - i + 1, s->items[i]);
94      }
95  }
96
97  static Ride *adultRides = NULL;
98  static Ride *childrenRides = NULL;
99  static Ride *heritageSites = NULL;
100
101 static int adultCount = 0;
102 static int childrenCount = 0;
103 static int heritageCount = 0;
104
105 static RideStack rideHistory;
```

fantasyKingdom.c M

src > fantasyKingdom.c > visitorAge

```c
static RideStack rideHistory;
static int remainingRides = 12;
static char currentVisitor[MAX_NAME_LEN] = "";

static int completedAdult[50] = {0};
static int completedChildren[50] = {0};
static int completedHeritage[50] = {0};

extern int visitorAge;
extern float visitorHeight;
extern int childRidePermission;
extern char visitorNameGlobal[];

int getLastVisitorName(char *outName, int size) {
    FILE *f = fopen("userData.txt", "r");
    if (!f) return 0;

    char line[1000];
    char last[1000] = "";

    while (fgets(line, sizeof(line), f)) {
        strcpy(last, line);
    }
    fclose(f);

    char *p = strstr(last, "name:");
    if (!p) return 0;
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_createRides(const char * [], int, const int [])

```c
int getLastVisitorName(char *outName, int size) {
    p += 5;
    while (*p == ' ') p++;

    char *q = strstr(p, " age:");
    if (!q) return 0;

    int len = q - p;
    if (len >= size) len = size - 1;

    strncpy(outName, p, len);
    outName[len] = '\0';
    return 1;
}

Ride *FK_createRides(const char *names[], int count, const int durations[]) {
    Ride *arr = malloc(sizeof(Ride) * count);
    for (int i = 0; i < count; i++) {
        strncpy(arr[i].name, names[i], MAX_RIDE_NAME - 1);
        arr[i].name[MAX_RIDE_NAME - 1] = '\0';
        arr[i].durationMinutes = durations[i];
        FK_initQueue(&arr[i].queue);
    }
    return arr;
}
```

fantasyKingdom.c M

src > fantasyKingdom.c > buildFantasyKingdom()

```c
void buildFantasyKingdom() {

    if (adultRides != NULL) return; // already built

    // Adult rides (13)
    const char *adultNames[] = {
        "Roller Coaster", "Magic Carpet", "Bumper Cars", "Flying Disco",
        "Ferris Wheel", "Pirate Ship", "VR Zone", "Horror House",
        "Tornado 360", "Santa Maria", "Wheely Bird", "ZuZu Train",
        "Heritage Coaster"
    };

    int adultDur[] = {6,4,5,5,6,5,8,7,6,6,4,3,7};
    adultCount = 13;

    adultRides = FK_createRides(adultNames, adultCount, adultDur);

    // Children rides
    const char *childNames[] = {
        "Highway Convoy", "Bull Dozer", "Sun & Moon",
        "Zip Around", "Pony Adventure", "Sky Hopper", "Kids Bumper Car"
    };
    int childDur[] = {3,3,2,3,4,3,3};
    childrenCount = 7;

    childrenRides = FK_createRides(childNames, childrenCount, childDur);
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_showRideList()

```c
void buildFantasyKingdom() {

    // Heritage sites
    const char *heritNames[] = {
        "Sitakot Bihar", "Kantajir Mandir", "Ahsan Manzil", "Chunakhola Mosque",
        "National Monument", "Puthia Palace", "Shat Gombuj Mosque", "Paharpur"
    };
    int hDur[] = {0,0,0,0,0,0,0,0};
    heritageCount = 8;

    heritageSites = FK_createRides(heritNames, heritageCount, hDur);

    FK_initStack(&rideHistory);
}

void FK_showRideList() {
    printf("\n================= FANTASY KINGDOM RIDE LIST =================\n");

    printf("\n-- Adult Section --\n");
    for (int i = 0; i < adultCount; i++)
        printf("%2d) %-22s  Queue: %2d  Wait: %d min\n",
               i + 1,
               adultRides[i].name,
               adultRides[i].queue.length,
               adultRides[i].queue.length * adultRides[i].durationMinutes);

    printf("\n-- Children Section --\n");
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_showAdultRides()

```
198  void FK_showRideList() {
210      for (int i = 0; i < childrenCount; i++)
211          printf("%2d) %-22s  Queue: %2d  Wait: %d min  (Fare: %d tk)\n",
212                 i + 1,
213                 childrenRides[i].name,
214                 childrenRides[i].queue.length,
215                 childrenRides[i].queue.length * childrenRides[i].durationMinutes,
216                 CHILD_RIDE_FARE);
217
218      printf("\n-- Heritage Section --\n");
219      for (int i = 0; i < heritageCount; i++)
220          printf("%2d) %-22s  (Visiting Only)\n",
221                 i + 1, heritageSites[i].name);
222
223      printf("================================================================\n");
224  }
225
226  void FK_showAdultRides() {
227      printf("\n================= ADULT RIDES =================\n");
228      for (int i = 0; i < adultCount; i++) {
229          if (completedAdult[i]) {
230              printf("%2d) %-22s  [COMPLETED]\n", i + 1, adultRides[i].name);
231          } else {
232              printf("%2d) %-22s  Queue: %2d  Wait: %d min\n",
233                     i + 1,
234                     adultRides[i].name,
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_showAdultRides()

```
226  void FK_showAdultRides() {
228      for (int i = 0; i < adultCount; i++) {
231          } else {
235                     adultRides[i].queue.length,
236                     adultRides[i].queue.length * adultRides[i].durationMinutes);
237          }
238      }
239      printf("================================================\n");
240  }
241
242  /* Display only Children Rides */
243  void FK_showChildrenRides() {
244      printf("\n================= CHILDREN RIDES =================\n");
245      for (int i = 0; i < childrenCount; i++) {
246          if (completedChildren[i]) {
247              printf("%2d) %-22s  [COMPLETED]\n", i + 1, childrenRides[i].name);
248          } else {
249              printf("%2d) %-22s  Queue: %2d  Wait: %d min  (Fare: %d tk)\n",
250                     i + 1,
251                     childrenRides[i].name,
252                     childrenRides[i].queue.length,
253                     childrenRides[i].queue.length * childrenRides[i].durationMinutes,
254                     CHILD_RIDE_FARE);
255          }
256      }
257      printf("================================================\n");
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_takeAdultRide()

```c
void FK_showHeritageSites() {
    printf("\n================= HERITAGE SITES =================\n");
    for (int i = 0; i < heritageCount; i++) {
        if (completedHeritage[i]) {
            printf("%2d) %-22s  [VISITED]\n", i + 1, heritageSites[i].name);
        } else {
            printf("%2d) %-22s  (Visiting Only)\n", i + 1, heritageSites[i].name);
        }
    }
    printf("==================================================\n");
}

void FK_takeAdultRide() {
    if (visitorAge < 10 || visitorHeight < 4.0) {
        printf("\n=====================================\n");
        printf("[ACCESS DENIED] - NOT READY YET\n");
        printf("=====================================\n");
        printf("You must be:\n");
        printf("  - At least 10 years old\n");
        printf("  - At least 4 feet tall\n");
        printf("\nCurrent Status:\n");
        printf("  - Age: %d years\n", visitorAge);
        printf("  - Height: %.1f feet\n", visitorHeight);
        printf("\nYou can enjoy: Kids Rides & Heritage Sites\n");
        printf("=====================================\n");
        return;
    }
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_takeAdultRide()

```c
void FK_takeAdultRide() {
    if (visitorAge < 10 || visitorHeight < 4.0) {
    }

    int choice;
    FK_showAdultRides();

    printf("\nSelect Adult Ride (1-%d) or 0 to cancel: ", adultCount);
    scanf("%d", &choice);

    if (choice <= 0 || choice > adultCount) return;

    int idx = choice - 1;

    if (completedAdult[idx]) {
        printf("\n[WARNING] You have already completed this ride!\n");
        return;
    }

    if (remainingRides <= 0) {
        printf("No remaining rides left.\n");
        return;
    }

    FK_enqueue(&adultRides[idx].queue, currentVisitor);

    printf("\n=====================================\n");
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_takeChildRide()

```
272 void FK_takeAdultRide() {
310     printf("\n=======================================\n");
311     printf("[COMPLETED] RIDE COMPLETED\n");
312     printf("=======================================\n");
313     printf("Ride: %s\n", adultRides[idx].name);
314     printf("Duration: %d minutes\n", adultRides[idx].durationMinutes);
315     printf("Remaining rides: %d\n", remainingRides - 1);
316     printf("=======================================\n");
317
318     remainingRides--;
319     completedAdult[idx] = 1;
320     FK_push(&rideHistory, adultRides[idx].name);
321
322     printf("\n--- Updated Adult Rides List ---\n");
323     FK_showAdultRides();
324 }
325
326 void FK_takeChildRide() {
327     /* Check if child rides are permitted */
328     if (!childRidePermission) {
329         printf("\n=======================================\n");
330         printf("[ACCESS DENIED]\n");
331         printf("=======================================\n");
332         printf("Child rides are NOT permitted for you.\n");
333         printf("You can enjoy: Adult Rides & Heritage Sites\n");
334         printf("=======================================\n");
335         return;
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_takeChildRide()

```
326 void FK_takeChildRide() {
338     FK_showChildrenRides();
339
340     printf("\nSelect Children Ride (1-%d) or 0 to cancel: ", childrenCount);
341     int choice;
342     scanf("%d", &choice);
343
344     if (choice <= 0 || choice > childrenCount) return;
345
346     int idx = choice - 1;
347
348     if (completedChildren[idx]) {
349         printf("\n[WARNING] You have already completed this ride!\n");
350         return;
351     }
352
353     if (remainingRides <= 0) {
354         printf("No remaining rides left.\n");
355         return;
356     }
357
358     FK_enqueue(&childrenRides[idx].queue, currentVisitor);
359
360     printf("\n=======================================\n");
361     printf("[COMPLETED] RIDE COMPLETED\n");
362     printf("=======================================\n");
```

fantasyKingdom.c M

src > fantasyKingdom.c > FK_visitHeritage()

```
326 void FK_takeChildRide() {
360     printf("\n========================================\n");
361     printf("[COMPLETED] RIDE COMPLETED\n");
362     printf("========================================\n");
363     printf("Ride: %s\n", childrenRides[idx].name);
364     printf("Duration: %d minutes\n", childrenRides[idx].durationMinutes);
365     printf("Fare Paid: %d tk\n", CHILD_RIDE_FARE);
366     printf("Remaining rides: %d\n", remainingRides - 1);
367     printf("========================================\n");
368
369     remainingRides--;
370     completedChildren[idx] = 1;
371     FK_push(&rideHistory, childrenRides[idx].name);
372
373     printf("\n--- Updated Children Rides List ---\n");
374     FK_showChildrenRides();
375 }
376
377 void FK_visitHeritage() {
378     FK_showHeritageSites();
379     printf("\nSelect Heritage Site (1-%d) or 0 to cancel: ", heritageCount);
380     int choice;
381     scanf("%d", &choice);
382
383     if (choice <= 0 || choice > heritageCount) return;
384
385     int idx = choice - 1;
```

fantasyKingdom.c M

src > fantasyKingdom.c > fantasyKingdomWelcoming()

```
377 void FK_visitHeritage() {
387     if (completedHeritage[idx]) {
388         printf("\n[WARNING] You have already visited this site!\n");
389         return;
390     }
391
392     printf("\n========================================\n");
393     printf("[COMPLETED] VISIT COMPLETED\n");
394     printf("========================================\n");
395     printf("Site: %s\n", heritageSites[idx].name);
396     printf("Thank you for visiting!\n");
397     printf("========================================\n");
398
399     completedHeritage[idx] = 1;
400     FK_push(&rideHistory, heritageSites[idx].name);
401
402     printf("\n--- Updated Heritage Sites List ---\n");
403     FK_showHeritageSites();
404 }
405
406 void fantasyKingdomWelcoming() {
407
408     buildFantasyKingdom();
409
410     /* Use current visitor name from main */
411     strncpy(currentVisitor, visitorNameGlobal, MAX_NAME_LEN - 1);
412     currentVisitor[MAX_NAME_LEN - 1] = '\0';
```

fantasyKingdom.c M

src > fantasyKingdom.c > fantasyKingdomWelcoming()

```c
406  void fantasyKingdomWelcoming() {
414      printf("\n=============== WELCOME %s TO FANTASY KINGDOM ==============\n",
415             currentVisitor);
416
417      remainingRides = 12;
418
419      while (1) {
420          printf("\nFantasy Kingdom Menu:\n");
421          printf("1) View Ride List\n");
422          printf("2) Take a Ride\n");
423          printf("3) Show Remaining Rides\n");
424          printf("4) Ride History\n");
425          printf("5) Exit Fantasy Kingdom\n");
426          printf("> ");
427
428          int op;
429          scanf("%d", &op);
430
431          if (op == 1) {
432              printf("\nSelect which Ride List:\n");
433              if (visitorAge >= 10 && visitorHeight >= 4.0)
434                  printf("1) Adult Rides\n");
435              printf("2) Children Rides\n");
436              printf("3) Heritage Sites\n");
437              printf("0) Back to Main Menu\n");
438              printf("> ");
```

fantasyKingdom.c M

src > fantasyKingdom.c > fantasyKingdomWelcoming()

```c
406  void fantasyKingdomWelcoming() {
419      while (1) {
431          if (op == 1) {
439              int t;
440              scanf("%d", &t);
441
442              if (t == 1) {
443                  if (visitorAge >= 10 && visitorHeight >= 4.0)
444                      FK_showAdultRides();
445                  else
446                      printf("\nYou are not eligible to view Adult Rides yet.\n");
447              }
448              else if (t == 2) FK_showChildrenRides();
449              else if (t == 3) FK_showHeritageSites();
450              else if (t == 0) { /* Go back to main menu */ }
451              else printf("\nInvalid option! Please select again.\n");
452          }
453
454          else if (op == 2) {
455              printf("\nSelect Ride Type:\n");
456              if (visitorAge >= 10 && visitorHeight >= 4.0)
457                  printf("1) Adult Ride\n");
458              printf("2) Children Ride\n");
459              printf("3) Heritage Visit\n");
460              printf("0) Back to Main Menu\n");
461              printf("> ");
462              int t;
```

fantasyKingdom.c M

src > fantasyKingdom.c > fantasyKingdomWelcoming()

```c
void fantasyKingdomWelcoming() {
    while (1) {
        else if (op == 2) {
            scanf("%d", &t);

            if (t == 1) FK_takeAdultRide();
            else if (t == 2) FK_takeChildRide();
            else if (t == 3) FK_visitHeritage();
            else if (t == 0) { /* Go back to main menu */ }
            else printf("\nInvalid option! Please select again.\n");
        }

        else if (op == 3)
            printf("\n[INFO] Remaining rides: %d out of 12\n", remainingRides);

        else if (op == 4)
            FK_printStack(&rideHistory);

        else if (op == 5) {
            printf("\nExiting Fantasy Kingdom...\n");
            return;
        }

        else printf("Invalid option.\n");
    }
}
```

offers.c M

src > offers.c > ...

```c
#include <stdio.h>
#include <string.h>

int offerChoice = 0;

char* getDayName(int date) {
    int dayIndex = (date % 7);

    switch (dayIndex) {
        case 1:
            return "Monday";
        case 2:
            return "Tuesday";
        case 3:
            return "Wednesday";
        case 4:
            return "Thursday";
        case 5:
            return "Friday";
        case 6:
            return "Saturday";
        case 0:
            return "Sunday";
        default:
            return "Unknown";
    }
}
```

offers.c M

src > offers.c > ...

```c
void showRegularOffer(int date) {
    printf("------------------------------------------------------------\n");
    printf("  OFFER FOR DECEMBER %d, 2025 (REGULAR)\n", date);
    printf("------------------------------------------------------------\n");
    printf("1) Fantasy Kingdom : 12 rides + Entry = 1000 tk\n");
    printf("2) Water Kingdom   : Unlimited rides + Entry = 1000 tk\n");
    printf("3) Combo Offer     : Fantasy + Water + Entry = 1500 tk\n");
    printf("------------------------------------------------------------\n");
}

void showFamilyDayOffer(int date) {
    printf("------------------------------------------------------------\n");
    printf("  FAMILY DAY OFFER - DECEMBER %d, 2025 (TUESDAY)\n", date);
    printf("------------------------------------------------------------\n");
    printf("1) Fantasy Kingdom : 12 rides + Entry = 500 tk\n");
    printf("2) Water Kingdom   : Unlimited rides = 600 tk\n");
    printf("------------------------------------------------------------\n");
}

void showHolidayOffer(int date) {
    printf("------------------------------------------------------------\n");
    printf("  HOLIDAY OFFER - DECEMBER %d, 2025 (FRIDAY/SATURDAY)\n", date);
    printf("------------------------------------------------------------\n");
    printf("1) Only Entry\n");
    printf("2) Fantasy Kingdom\n");
    printf("3) Water Kingdom\n");
    printf("------------------------------------------------------------\n");
```

offers.c M

src > offers.c > ...

```c
void takeOfferChoice(int maxOption) {
    int choice;
    do {
        printf("Choose an offer: ");
        scanf("%d", &choice);
    } while (choice < 1 || choice > maxOption);

    if (maxOption == 2) {
        offerChoice = choice;
    } else {
        offerChoice = choice;
    }
}

void showOffer2025(int date) {
    char* day = getDayName(date);

    printf("============================================================\n");
    printf("                OFFERS FOR DECEMBER %d, 2025\n", date);
    printf("============================================================\n");
    printf(" Day: %s\n\n", day);

    if (strcmp(day, "Tuesday") == 0) {
        showFamilyDayOffer(date);
        takeOfferChoice(2);
    } else if (strcmp(day, "Friday") == 0 || strcmp(day, "Saturday") == 0) {
        showHolidayOffer(date);
```

offers.c M

src > offers.c > ...

```c
void showOffer2025(int date) {
    char* day = getDayName(date);

    printf("============================================================\n");
    printf("                OFFERS FOR DECEMBER %d, 2025\n", date);
    printf("============================================================\n");
    printf(" Day: %s\n\n", day);

    if (strcmp(day, "Tuesday") == 0) {
        showFamilyDayOffer(date);
        takeOfferChoice(2);
    } else if (strcmp(day, "Friday") == 0 || strcmp(day, "Saturday") == 0) {
        showHolidayOffer(date);
        takeOfferChoice(3);
    } else {
        showRegularOffer(date);
        takeOfferChoice(3);
    }
}
```

parking.c

src > parking.c > parkCar()

```c
#include <stdio.h>
#include <stdlib.h>


struct Car {
    int carNumber;
    struct Car* next;
};

struct Car* front = NULL;
struct Car* rear = NULL;


void parkCar() {
    struct Car* newCar = (struct Car*)malloc(sizeof(struct Car));

    if (newCar == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter car number: ");
    scanf("%d", &newCar->carNumber);

    newCar->next = NULL;

    if (rear == NULL) {
```

parking.c

src > parking.c > parkCar()

```c
void parkCar() {
    if (rear == NULL) {
        front = rear = newCar;
    } else {
        rear->next = newCar;
        rear = newCar;
    }

    printf("Car %d parked successfully.\n", newCar->carNumber);
}


void removeCar() {
    if (front == NULL) {
        printf("Parking is empty. No car to remove.\n");
        return;
    }

    struct Car* temp = front;
    printf("Car %d left the parking.\n", temp->carNumber);

    front = front->next;

    if (front == NULL) {
        rear = NULL;
    }
```

parking.c

src > parking.c > parkCar()

```c
void removeCar() {
    free(temp);
}

void showParking() {
    if (front == NULL) {
        printf("Parking is empty.\n");
        return;
    }

    struct Car* temp = front;
    printf("\nCars currently parked (Front -> Rear):\n");

    while (temp != NULL) {
        printf("Car %d\n", temp->carNumber);
        temp = temp->next;
    }
}

void carParking() {
    int choice;

    while (1) {
        printf("\n==============================\n");
        printf("      AMUSEMENT PARK PARKING\n");
        printf("==============================\n");
```

parking.c

src > parking.c > parkCar()

```c
void carParking() {
    while (1) {
        printf("\n==============================\n");
        printf("      AMUSEMENT PARK PARKING\n");
        printf("==============================\n");
        printf("1. Park Car\n");
        printf("2. Remove Car\n");
        printf("3. Show Parked Cars\n");
        printf("4. Exit Parking\n");
        printf("> ");

        scanf("%d", &choice);

        switch (choice) {
            case 1:
                parkCar();
                break;

            case 2:
                removeCar();
                break;

            case 3:
                showParking();
                break;

            case 4:
```

parking.c

src > parking.c > parkCar()

```
72  void carParking() {
75      while (1) {
87          switch (choice) {
100             case 4:
101                 printf("Exiting Car Parking System...\n");
102                 return;
103
104             default:
105                 printf("Invalid option!\n");
106         }
107     }
108 }
109
```

restaurant.c M

src > restaurant.c > restaurant()

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAX 5
5
6  struct OrderStack {
7      char orders[MAX][30];
8      int top;
9  };
10
11 struct OrderStack stack;
12
13 void initializeStack() {
14     stack.top = -1;
15 }
16
17 void placeOrder() {
18     if (stack.top == MAX - 1) {
19         printf("\nRestaurant is full! Cannot take more orders.\n");
20         return;
21     }
22
23     stack.top++;
24     printf("Enter food name: ");
25     scanf(" %[^\n]", stack.orders[stack.top]);
26
27     printf("Order added: %s\n", stack.orders[stack.top]);
```

restaurant.c M

src > restaurant.c > restaurant()

```c
void placeOrder() {
}

void serveOrder() {
    if (stack.top == -1) {
        printf("\nNo orders to serve.\n");
        return;
    }

    printf("Serving order: %s\n", stack.orders[stack.top]);
    stack.top--;
}

void showOrders() {
    if (stack.top == -1) {
        printf("\nNo current orders.\n");
        return;
    }

    printf("\nCurrent Orders (Top to Bottom):\n");
    for (int i = stack.top; i >= 0; i--) {
        printf("- %s\n", stack.orders[i]);
    }
}

void restaurant() {
    int choice;
```

restaurant.c M

src > restaurant.c > restaurant()

```c
void restaurant() {
    int choice;
    initializeStack();

    while (1) {
        printf("\n==============================\n");
        printf("     AMUSEMENT PARK RESTAURANT\n");
        printf("==============================\n");
        printf("1. Place Order\n");
        printf("2. Serve Order\n");
        printf("3. Show Orders\n");
        printf("4. Exit Restaurant\n");
        printf("> ");

        scanf("%d", &choice);

        switch (choice) {
            case 1:
                placeOrder();
                break;

            case 2:
                serveOrder();
                break;

            case 3:
```

restaurant.c M

src > restaurant.c > restaurant()

```c
void restaurant() {
    while (1) {
        switch (choice) {
            case 3:
                showOrders();
                break;

            case 4:
                printf("\nExiting Restaurant...\n");
                return;

            default:
                printf("\nInvalid option!\n");
        }
    }
}
```

waterKingdom.c

src > waterKingdom.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

struct userLine {
    int userSerial;
    struct userLine* next;
};

struct userLine* start = NULL;

struct userLine* createNode(int serial) {
    struct userLine* newNode = (struct userLine*)malloc(sizeof(struct userLine));

    newNode->userSerial = serial;
    newNode->next = NULL;

    return newNode;
}

void insertInUserLine(int serial) {
    struct userLine* newNode = (struct userLine*)malloc(sizeof(struct userLine));

    newNode->userSerial = serial;
    newNode->next = NULL;

    if (start == NULL) {
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void insertInUserLine(int serial) {

    if (start == NULL) {
        start = newNode;
    } else {
        struct userLine* i = start;

        while (i->next != NULL) {
            i = i->next;
        }
        i->next = newNode;
    }
}

void deleteFromTheUserLine() {
    if (start == NULL) {
        printf("There's nothing to delete \n");
    } else {
        start = start->next;
    }
}

void printFamilyPoolLine() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
```

```c
void printFamilyPoolLine() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}

void clearUserLine() {
    struct userLine* cur = start;
    while (cur != NULL) {
        struct userLine* tmp = cur;
        cur = cur->next;
        free(tmp);
    }
    start = NULL;
}

void familyPoolFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void familyPoolFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
    insertInUserLine(6);
    insertInUserLine(7);
    insertInUserLine(8);

    printf("\nFamily Pool is currently full with 200 people, sorry you have to wait\n\n");
    printFamilyPoolLine();

    int totalFamilyPoolMember = 200;

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalFamilyPoolMember--;
        printf("One member left the family pool!\n");
        printf("Current Family Pool member status: %d\n", totalFamilyPoolMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered the pool\n", i + 1);
        totalFamilyPoolMember++;
        printf("Current Family Pool member status: %d\n", totalFamilyPoolMember);
```

```c
// waterKingdom.c
// src > C waterKingdom.c > ...
67  void familyPoolFunction() {
83      for (int i = 0; i < 5; i++) {
92          printf("Current Family Pool member status: %d\n", totalFamilyPoolMember);
93          Sleep(1000);
94          i < 4 ? printFamilyPoolLine() : printf("");
95      }
96      printf("\nyou are in!\n\n");
97  }
98
99  void printWavePoolLine() {
100     struct userLine* i = start;
101
102     while (i != NULL) {
103         i->userSerial == 8 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
104         i = i->next;
105     }
106     printf("\n");
107 }
108
109 void wavePoolFunction() {
110     clearUserLine();
111     insertInUserLine(1);
112     insertInUserLine(2);
113     insertInUserLine(3);
114     insertInUserLine(4);
115     insertInUserLine(5);
116     insertInUserLine(6);
```

```c
// waterKingdom.c
// src > C waterKingdom.c > ...
109 void wavePoolFunction() {
117     insertInUserLine(7);
118     insertInUserLine(8);
119     insertInUserLine(9);
120
121     int totalWavePoolMember = 100;
122     printf("\nWave Pool is currently full with %d people, sorry you have to wait\n\n", totalWavePoolMember);
123     printWavePoolLine();
124
125     for (int i = 0; i < 8; i++) {
126         Sleep(3000);
127         totalWavePoolMember--;
128         printf("One member left the Wave Pool!\n");
129         printf("Current Wave Pool member status: %d\n", totalWavePoolMember);
130         Sleep(1000);
131         deleteFromTheUserLine();
132         printf("User with serial number %d entered the pool\n", i + 1);
133         totalWavePoolMember++;
134         printf("Current Wave Pool member status: %d\n", totalWavePoolMember);
135         Sleep(1000);
136         i < 7 ? printWavePoolLine() : printf("");
137     }
138     printf("\nyou are in!\n");
139 }
```

```c
void printDancingZoneLine() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}

void dancingZoneFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);

    int totalDancingZoneMember = 50;
    printf("\nDancing Zone is currently full with %d people, sorry you have to wait\n\n", totalDancingZoneMember);
    printDancingZoneLine();

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalDancingZoneMember--;
        printf("One member left the Dancing Zone!\n");
```

```c
void dancingZoneFunction() {
    for (int i = 0; i < 5; i++) {
        printf("Current Dancing Zone member status: %d\n", totalDancingZoneMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered the pool\n", i + 1);
        totalDancingZoneMember++;
        printf("Current Dancing Zone member status: %d\n", totalDancingZoneMember);
        Sleep(1000);
        i < 4 ? printDancingZoneLine() : printf("");
    }
    printf("\nyou are in!\n");
}

void printGreerSlideLine() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 1 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}

void greerSlideFunction() {
    clearUserLine();
    insertInUserLine(1);
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void greerSlideFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);

    int totalDancingZoneMember = 30;
    printf("\nGreer Slide is currently full with %d people, sorry you have to wait\n\n", totalDancingZoneMember);
    printGreerSlideLine();

    for (int i = 0; i < 1; i++) {
        Sleep(3000);
        totalDancingZoneMember--;
        printf("One member left the Greer Slide!\n");
        printf("Current Greer Slide member status: %d\n", totalDancingZoneMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered the pool\n", i + 1);
        totalDancingZoneMember++;
        printf("Current Greer Slide member status: %d\n", totalDancingZoneMember);
        Sleep(1000);
        i < 0 ? printGreerSlideLine() : printf("");
    }
    printf("\nyou are in!\n");
}
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void printMultiSlideLine() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}

void multiSlideFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
    insertInUserLine(6);
    insertInUserLine(7);

    int totalDancingZoneMember = 30;
    printf("\nMulti Slide is currently full with %d people, sorry you have to wait\n\n", totalDancingZoneMember);
    printMultiSlideLine();

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalDancingZoneMember--;
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void multiSlideFunction() {
    for (int i = 0; i < 5; i++) {
        printf("One member left the Multi Slide!\n");
        printf("Current Multi Slide member status: %d\n", totalDancingZoneMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered Multi Slide\n", i + 1);
        totalDancingZoneMember++;
        printf("Current Multi Slide member status: %d\n", totalDancingZoneMember);
        Sleep(1000);
        i < 4 ? printMultiSlideLine() : printf("");
    }
    printf("\nyou are in!\n");
}

void printYellowFly() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void yellowFlyFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
    insertInUserLine(6);
    insertInUserLine(7);

    int totalYellowFlyMember = 30;
    printf("\nYellow Fly is currently full with %d people, sorry you have to wait\n\n", totalYellowFlyMember);
    printYellowFly();

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalYellowFlyMember--;
        printf("One member left the Yellow Fly!\n");
        printf("Current Yellow Fly member status: %d\n", totalYellowFlyMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered Yellow Fly\n", i + 1);
        totalYellowFlyMember++;
        printf("Current Yellow Fly member status: %d\n", totalYellowFlyMember);
        Sleep(1000);
        i < 4 ? printYellowFly() : printf("");
    }
```

```c
// waterKingdom.c
// src > C waterKingdom.c > ...
void yellowFlyFunction() {
    for (int i = 0; i < 5; i++) {
    printf("\nyou are in!\n");
}

void printBlueTunnel() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}

void blueTunnelFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
    insertInUserLine(6);
    insertInUserLine(7);

    int totalBlueTunnelMember = 30;
    printf("\nBlue Tunnel is currently full with %d people, sorry you have to wait\n\n", totalBlueTunnelMember);
```

```c
// waterKingdom.c
// src > C waterKingdom.c > ...
void blueTunnelFunction() {
    printf("\nBlue Tunnel is currently full with %d people, sorry you have to wait\n\n", totalBlueTunnelMember);
    printBlueTunnel();

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalBlueTunnelMember--;
        printf("One member left the Blue Tunnel!\n");
        printf("Current Blue Tunnel member status: %d\n", totalBlueTunnelMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered Blue Tunnel\n", i + 1);
        totalBlueTunnelMember++;
        printf("Current Blue Tunnel member status: %d\n", totalBlueTunnelMember);
        Sleep(1000);
        i < 4 ? printBlueTunnel() : printf("");
    }
    printf("\nyou are in!\n");
}

void printRedTunnel() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void printRedTunnel() {
    printf("\n");
}

void redTunnelFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
    insertInUserLine(6);
    insertInUserLine(7);

    int totalRedTunnelMember = 30;
    printf("\nRed Tunnel is currently full with %d people, sorry you have to wait\n\n", totalRedTunnelMember);
    printRedTunnel();

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalRedTunnelMember--;
        printf("One member left the Red Tunnel!\n");
        printf("Current Red Tunnel member status: %d\n", totalRedTunnelMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered Red Tunnel\n", i + 1);
        totalRedTunnelMember++;
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void redTunnelFunction() {
    for (int i = 0; i < 5; i++) {
        printf("Current Red Tunnel member status: %d\n", totalRedTunnelMember);
        Sleep(1000);
        i < 4 ? printRedTunnel() : printf("");
    }
    printf("\nyou are in!\n");
}

void printLazyRiverLine() {
    struct userLine* i = start;

    while (i != NULL) {
        i->userSerial == 5 ? printf("%d (You are here) \n", i->userSerial) : printf("%d\n", i->userSerial);
        i = i->next;
    }
    printf("\n");
}

void lazyRiverFunction() {
    clearUserLine();
    insertInUserLine(1);
    insertInUserLine(2);
    insertInUserLine(3);
    insertInUserLine(4);
    insertInUserLine(5);
    insertInUserLine(6);
```

waterKingdom.c

src > waterKingdom.c > ...

```c
void lazyRiverFunction() {
    insertInUserLine(7);

    int totalLazyRiverMember = 30;
    printf("\nLazy River is currently full with %d people, sorry you have to wait\n\n", totalLazyRiverMember);
    printLazyRiverLine();

    for (int i = 0; i < 5; i++) {
        Sleep(3000);
        totalLazyRiverMember--;
        printf("One member left the Lazy River!\n");
        printf("Current Lazy River member status: %d\n", totalLazyRiverMember);
        Sleep(1000);
        deleteFromTheUserLine();
        printf("User with serial number %d entered Lazy River\n", i + 1);
        totalLazyRiverMember++;
        printf("Current Lazy River member status: %d\n", totalLazyRiverMember);
        Sleep(1000);
        i < 4 ? printLazyRiverLine() : printf("");
    }
    printf("\nyou are in!\n");
}

int deleteRideFromTheRideList(int waterKingdomUserRideChoises[], int size, int rideNumber) {
    int deletingNumbersIdx = -1;

    for (int i = 0; i < size; i++) {
```

waterKingdom.c

src > waterKingdom.c > deleteRideFromTheRideList(int [], int, int)

```c
int deleteRideFromTheRideList(int waterKingdomUserRideChoises[], int size, int rideNumber) {
    for (int i = 0; i < size; i++) {
        if (waterKingdomUserRideChoises[i] == rideNumber) {
            deletingNumbersIdx = i;
            break;
        }
    }

    if (deletingNumbersIdx == -1) {
        return size;
    }

    for (int i = deletingNumbersIdx; i < size - 1; i++) {
        waterKingdomUserRideChoises[i] = waterKingdomUserRideChoises[i + 1];
    }

    return size - 1;
}

void printLeftRides(int totalRideLeft, int waterKingdomUserRideChoises[]) {
    printf("Ride left: ");

    for (int i = 0 ; i < totalRideLeft ; i++) {
        if (waterKingdomUserRideChoises[i] == 1) {
            printf("| 1) Family Pool |");
        } else if (waterKingdomUserRideChoises[i] == 2) {
```

```c
// waterKingdom.c
// src > C waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
void printLeftRides(int totalRideLeft, int waterKingdomUserRideChoises[]) {
    for (int i = 0 ; i < totalRideLeft ; i++) {
        if (waterKingdomUserRideChoises[i] == 1) {
        } else if (waterKingdomUserRideChoises[i] == 2) {
            printf("| 2) Wave Pool |");
        } else if (waterKingdomUserRideChoises[i] == 3) {
            printf("| 3) Dancing Zone |");
        } else if (waterKingdomUserRideChoises[i] == 4) {
            printf("| 4) Greer Slide |");
        } else if (waterKingdomUserRideChoises[i] == 5) {
            printf("| 5) Multi Slide |");
        } else if (waterKingdomUserRideChoises[i] == 6) {
            printf("| 6) Yellow Fly |");
        } else if (waterKingdomUserRideChoises[i] == 7) {
            printf("| 7) Blue Tunnel |");
        } else if (waterKingdomUserRideChoises[i] == 8) {
            printf("| 8) Red Tunnel |");
        } else if (waterKingdomUserRideChoises[i] == 9) {
            printf("| 9) Lazy River |");
        }
    }

    printf("\n\n");
}

void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
```

```c
// waterKingdom.c
// src > C waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
    int rideNumber;
    chooseAgainToStartRiding:
        printf("Choose the ride number you want to go \n");
        printf("Or type 0 to exit\n");
        printf("> ");
        scanf("%d", &rideNumber);

        if (rideNumber == 0) {
            return;
        }

        if (rideNumber == 1) {
            int count = 0;
            for (int i = 0; i < 9; i++) {
                if (waterKingdomUserRideChoises[i] == 1) {
                    count++;
                    break;
                }
            }

            if (count == 1) {
                familyPoolFunction();
                int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
                printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
                goto chooseAgainToStartRiding;
```

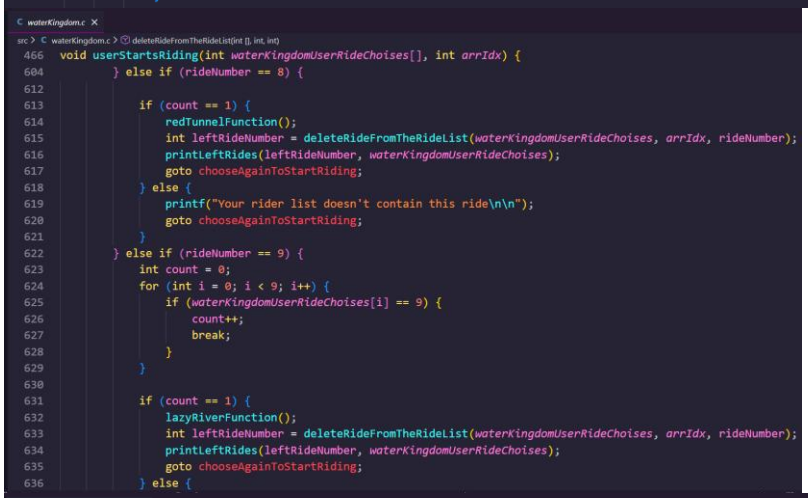
```c
// waterKingdom.c
// src > C waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
        if (rideNumber == 1) {
            if (count == 1) {
                goto chooseAgainToStartRiding;
            } else {
                printf("Your rider list doesn't contain this ride\n\n");
                goto chooseAgainToStartRiding;
            }
        } else if (rideNumber == 2) {
            int count = 0;
            for (int i = 0; i < 9; i++) {
                if (waterKingdomUserRideChoises[i] == 2) {
                    count++;
                    break;
                }
            }

            if (count == 1) {
                wavePoolFunction();
                int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
                printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
                goto chooseAgainToStartRiding;
            } else {
                printf("Your rider list doesn't contain this ride\n\n");
                goto chooseAgainToStartRiding;
            }
        } else if (rideNumber == 3) {
```

```
waterKingdom.c
src > waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
466  void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
514          } else if (rideNumber == 3) {
515              int count = 0;
516              for (int i = 0; i < 9; i++) {
517                  if (waterKingdomUserRideChoises[i] == 3) {
518                      count++;
519                      break;
520                  }
521              }
522
523              if (count == 1) {
524                  dancingZoneFunction();
525                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
526                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
527                  goto chooseAgainToStartRiding;
528              } else {
529                  printf("Your rider list doesn't contain this ride\n\n");
530                  goto chooseAgainToStartRiding;
531              }
532          } else if (rideNumber == 4) {
533              int count = 0;
534              for (int i = 0; i < 9; i++) {
535                  if (waterKingdomUserRideChoises[i] == 4) {
536                      count++;
537                      break;
538                  }
539              }
```

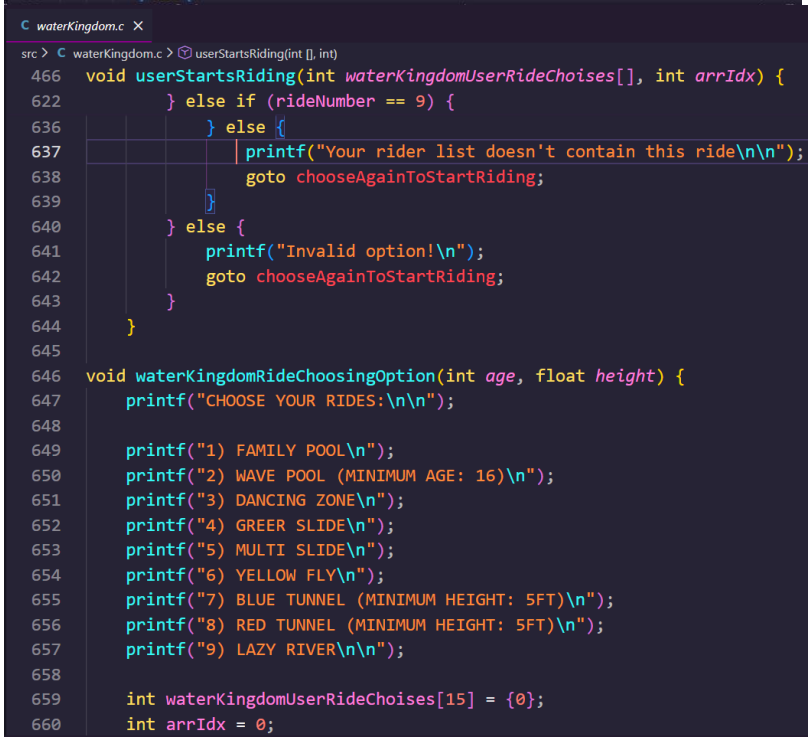
```
waterKingdom.c
src > waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
466  void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
532          } else if (rideNumber == 4) {
540
541              if (count == 1) {
542                  greerSlideFunction();
543                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
544                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
545                  goto chooseAgainToStartRiding;
546              } else {
547                  printf("Your rider list doesn't contain this ride\n\n");
548                  goto chooseAgainToStartRiding;
549              }
550          } else if (rideNumber == 5) {
551              int count = 0;
552              for (int i = 0; i < 9; i++) {
553                  if (waterKingdomUserRideChoises[i] == 5) {
554                      count++;
555                      break;
556                  }
557              }
558
559              if (count == 1) {
560                  multiSlideFunction();
561                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
562                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
563                  goto chooseAgainToStartRiding;
564              } else {
```

```
waterKingdom.c
src > waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
466  void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
550          } else if (rideNumber == 5) {
564              } else {
565                  printf("Your rider list doesn't contain this ride\n\n");
566                  goto chooseAgainToStartRiding;
567              }
568          } else if (rideNumber == 6) {
569              int count = 0;
570              for (int i = 0; i < 9; i++) {
571                  if (waterKingdomUserRideChoises[i] == 6) {
572                      count++;
573                      break;
574                  }
575              }
576
577              if (count == 1) {
578                  yellowFlyFunction();
579                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
580                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
581                  goto chooseAgainToStartRiding;
582              } else {
583                  printf("Your rider list doesn't contain this ride\n\n");
584                  goto chooseAgainToStartRiding;
585              }
586          } else if (rideNumber == 7) {
587              int count = 0;
588              for (int i = 0; i < 9; i++) {
```

```
waterKingdom.c
src > waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
466  void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
586          } else if (rideNumber == 7) {
588              for (int i = 0; i < 9; i++) {
589                  if (waterKingdomUserRideChoises[i] == 7) {
590                      count++;
591                      break;
592                  }
593              }
594
595              if (count == 1) {
596                  blueTunnelFunction();
597                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
598                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
599                  goto chooseAgainToStartRiding;
600              } else {
601                  printf("Your rider list doesn't contain this ride\n\n");
602                  goto chooseAgainToStartRiding;
603              }
604          } else if (rideNumber == 8) {
605              int count = 0;
606              for (int i = 0; i < 9; i++) {
607                  if (waterKingdomUserRideChoises[i] == 8) {
608                      count++;
609                      break;
610                  }
611              }
```



```
waterKingdom.c
src > waterKingdom.c > deleteRideFromTheRideList(int [], int, int)
466  void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
604          } else if (rideNumber == 8) {
612
613              if (count == 1) {
614                  redTunnelFunction();
615                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
616                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
617                  goto chooseAgainToStartRiding;
618              } else {
619                  printf("Your rider list doesn't contain this ride\n\n");
620                  goto chooseAgainToStartRiding;
621              }
622          } else if (rideNumber == 9) {
623              int count = 0;
624              for (int i = 0; i < 9; i++) {
625                  if (waterKingdomUserRideChoises[i] == 9) {
626                      count++;
627                      break;
628                  }
629              }
630
631              if (count == 1) {
632                  lazyRiverFunction();
633                  int leftRideNumber = deleteRideFromTheRideList(waterKingdomUserRideChoises, arrIdx, rideNumber);
634                  printLeftRides(leftRideNumber, waterKingdomUserRideChoises);
635                  goto chooseAgainToStartRiding;
636              } else {
```



```
waterKingdom.c
src > waterKingdom.c > userStartsRiding(int [], int)
466  void userStartsRiding(int waterKingdomUserRideChoises[], int arrIdx) {
622          } else if (rideNumber == 9) {
636              } else {
637                  printf("Your rider list doesn't contain this ride\n\n");
638                  goto chooseAgainToStartRiding;
639              }
640          } else {
641              printf("Invalid option!\n");
642              goto chooseAgainToStartRiding;
643          }
644      }
645
646  void waterKingdomRideChoosingOption(int age, float height) {
647      printf("CHOOSE YOUR RIDES:\n\n");
648
649      printf("1) FAMILY POOL\n");
650      printf("2) WAVE POOL (MINIMUM AGE: 16)\n");
651      printf("3) DANCING ZONE\n");
652      printf("4) GREER SLIDE\n");
653      printf("5) MULTI SLIDE\n");
654      printf("6) YELLOW FLY\n");
655      printf("7) BLUE TUNNEL (MINIMUM HEIGHT: 5FT)\n");
656      printf("8) RED TUNNEL (MINIMUM HEIGHT: 5FT)\n");
657      printf("9) LAZY RIVER\n\n");
658
659      int waterKingdomUserRideChoises[15] = {0};
660      int arrIdx = 0;
```

```c
void waterKingdomRideChoosingOption(int age, float height) {
    int waterKingdomUserRideChoise;

    chooseAgain:
        if (arrIdx >= 0 && arrIdx < 9) {
            printf("IF YOU ARE DONE WITH CHOOSING, TYPE 0\n");
            printf("> ");
            scanf("%d", &waterKingdomUserRideChoise);

            switch (waterKingdomUserRideChoise) {
                case 0:
                    goto userDoneWithChoosing;
                case 2:
                    if (age >= 16) {
                        for (int i = 0; i < arrIdx; i++) {
                            if (waterKingdomUserRideChoises[i] == waterKingdomUserRideChoise) {
                                printf("YOU'VE ALREADY SELECTED THIS OPTION");
                                goto chooseAgain;
                            }
                        }
                        waterKingdomUserRideChoises[arrIdx] = waterKingdomUserRideChoise;
                        arrIdx++;
                        printf("OPTION SELECTED!\n");
                        goto chooseAgain;
                    } else {
                        printf("You are not eligible for this ride, minimum age required is 16\n");
                        goto chooseAgain;
```

Loading...

```c
void waterKingdomRideChoosingOption(int age, float height) {
        if (arrIdx >= 0 && arrIdx < 9) {
            switch (waterKingdomUserRideChoise) {
                    } else {
                        goto chooseAgain;
                    }
                case 7:
                case 8:
                    if (height >= 5) {
                        for (int i = 0; i < arrIdx; i++) {
                            if (waterKingdomUserRideChoises[i] == waterKingdomUserRideChoise) {
                                printf("YOU'VE ALREADY SELECTED THIS OPTION");
                                goto chooseAgain;
                            }
                        }
                        waterKingdomUserRideChoises[arrIdx] = waterKingdomUserRideChoise;
                        arrIdx++;
                        printf("OPTION SELECTED!\n");
                        goto chooseAgain;
                    } else {
                        printf("You are not eligible for this ride, minimum height required is 5ft\n");
                        goto chooseAgain;
                    }
                case 1:
                case 3:
                case 4:
                case 5:
```

```c
void waterKingdomRideChoosingOption(int age, float height) {
        if (arrIdx >= 0 && arrIdx < 9) {
            switch (waterKingdomUserRideChoise) {
                case 5:
                case 6:
                case 9:
                    for (int i = 0; i < arrIdx; i++) {
                        if (waterKingdomUserRideChoises[i] == waterKingdomUserRideChoise) {
                            printf("YOU'VE ALREADY SELECTED THIS OPTION\n");
                            goto chooseAgain;
                        }
                    }
                    waterKingdomUserRideChoises[arrIdx] = waterKingdomUserRideChoise;
                    arrIdx++;
                    printf("OPTION SELECTED!\n");
                    goto chooseAgain;

                default:
                    printf("INVALID OPTION\n");
                    goto chooseAgain;
            }

            userDoneWithChoosing:
                printf("Rides you choosed: ");
                for (int i = 0; i < arrIdx; i++) {
                    if (waterKingdomUserRideChoises[i] == 1) {
                        printf("| 1) Family Pool |");
```

```c
void waterKingdomRideChoosingOption(int age, float height) {
        if (arrIdx >= 0 && arrIdx < 9) {
            for (int i = 0; i < arrIdx; i++) {
                } else if (waterKingdomUserRideChoises[i] == 2) {
                    printf("| 2) Wave Pool |");
                } else if (waterKingdomUserRideChoises[i] == 3) {
                    printf("| 3) Dancing Zone |");
                } else if (waterKingdomUserRideChoises[i] == 4) {
                    printf("| 4) Greer Slide |");
                } else if (waterKingdomUserRideChoises[i] == 5) {
                    printf("| 5) Multi Slide |");
                } else if (waterKingdomUserRideChoises[i] == 6) {
                    printf("| 6) Yellow Fly |");
                } else if (waterKingdomUserRideChoises[i] == 7) {
                    printf("| 7) Blue Tunnel |");
                } else if (waterKingdomUserRideChoises[i] == 8) {
                    printf("| 8) Red Tunnel |");
                } else if (waterKingdomUserRideChoises[i] == 9) {
                    printf("| 9) Lazy River |");
                }
            }
            printf("\n\n");
        } else {
            printf("You've choosen maximum number of rides, can't choose anymore\n");
            goto userDoneWithChoosing;
        }

```

```c
void waterKingdomRideChoosingOption(int age, float height) {
    userStartsRiding(waterKingdomUserRideChoises, arrIdx);
}

void waterKingdomWelcoming(int age, float height) {
    printf("\n");
    printf(" __        __   _                          \n");
    printf(" \\ \\      / /__| | ___ ___  _ __ ___   ___ \n");
    printf("  \\ \\ /\\ / / _ \\ |/ __/ _ \\| '_ ` _ \\ / _ \\ \n");
    printf("   \\ V  V /  __/ | (_| (_) | | | | | |  __/ \n");
    printf("    \\_/\\_/ \\___|_|\\___\\___/|_| |_| |_|\\___|  \n");
    printf("                                            \n");

    printf("  _____     \n");
    printf(" |_   _|__  \n");
    printf("   | |/ _ \\ \n");
    printf("   | | (_) |\n");
    printf("   |_|\\___/ \n");
    printf("            \n");
    printf(" __        __    _              _  ___                 _                        \n");
    printf(" \\ \\      / /_ _| |_ ___ _ __  | |/ (_)_ __   __ _  __| | ___  _ __ ___  \n");
    printf("  \\ \\ /\\ / / _` | __/ _ \\ '__| | ' /| | '_ \\ / _` |/ _` |/ _ \\| '_ ` _ \\ \n");
    printf("   \\ V  V / (_| | ||  __/ |    | . \\| | | | | (_| | (_| | (_) | | | | | |\n");
    printf("    \\_/\\_/ \\__,_|\\__\\___|_|    |_|\\_\\_|_| |_|\\__, |\\__,_|\\___/|_| |_| |_|\n");
    printf("                                              |___/                          \n");
```

```c
void waterKingdomWelcoming(int age, float height) {
    printf("\n");
    printf(" __        __   _                          \n");
    printf(" \\ \\      / /__| | ___ ___  _ __ ___   ___ \n");
    printf("  \\ \\ /\\ / / _ \\ |/ __/ _ \\| '_ ` _ \\ / _ \\ \n");
    printf("   \\ V  V /  __/ | (_| (_) | | | | | |  __/ \n");
    printf("    \\_/\\_/ \\___|_|\\___\\___/|_| |_| |_|\\___|  \n");
    printf("                                            \n");

    printf("  _____     \n");
    printf(" |_   _|__  \n");
    printf("   | |/ _ \\ \n");
    printf("   | | (_) |\n");
    printf("   |_|\\___/ \n");
    printf("            \n");
    printf(" __        __    _              _  ___                 _                        \n");
    printf(" \\ \\      / /_ _| |_ ___ _ __  | |/ (_)_ __   __ _  __| | ___  _ __ ___  \n");
    printf("  \\ \\ /\\ / / _` | __/ _ \\ '__| | ' /| | '_ \\ / _` |/ _` |/ _ \\| '_ ` _ \\ \n");
    printf("   \\ V  V / (_| | ||  __/ |    | . \\| | | | | (_| | (_| | (_) | | | | | |\n");
    printf("    \\_/\\_/ \\__,_|\\__\\___|_|    |_|\\_\\_|_| |_|\\__, |\\__,_|\\___/|_| |_| |_|\n");
    printf("                                              |___/                          \n");

    waterKingdomRideChoosingOption(age, height);
}
```

3.2 Performance Analysis

The system's runtime behavior aligns closely with the expected performance of the linear data structures used. Queue operations—implemented using linked lists—support **O(1)** enqueue and dequeue, while queue traversal for ride-status display runs in **O(n)**. This ensures efficient handling of visitor flow even as queue sizes increase. The waiting-time calculation model (queue.length × durationMinutes) provides a clear and consistent estimate for both adult and children rides, whereas heritage sites incur negligible processing cost due to their visit-only nature.

The Entry/Exit subsystem also performs efficiently through constant-time operations in both the array-based local queue and the VIP stack. Memory usage remains low because queue nodes are dynamically allocated

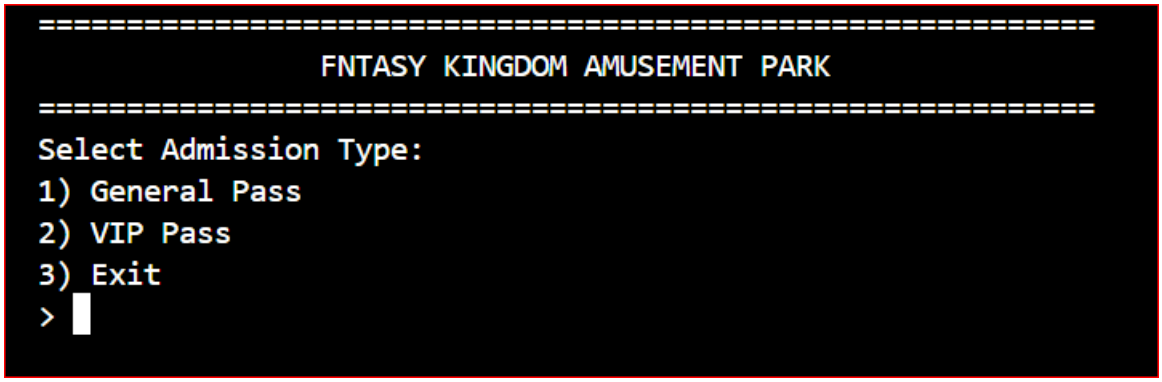
and deallocated on demand, and all static structures operate within predefined limits. Since the system runs in a single-threaded environment and uses Sleep() to simulate real-time behavior, most perceived delays originate from controlled simulation timing rather than computational overhead.

Category-wise Observations

- **Adult Section:** Longer ride durations increase predicted wait times; performance remains stable with O(1) queue updates.
- **Children Section:** Shorter ride durations reduce effective waiting time; permission checks naturally limit load.
- **Heritage Section:** Processes in constant time with no queue-based operations, imposing minimal performance cost.

Overall, the system demonstrates reliable, predictable performance. Its primary computational cost arises from queue traversal during display operations, while core queue and stack operations remain constant time. The implementation is lightweight, memory-efficient, and appropriate for the intended operational scale of the amusement park simulation.

3.3 Results and Discussion



```
> 1
Enter your name: Mahadi
Enter your Age: 22
Enter your Height (ft): 5.7
Mahadi added to Local Queue (ENTRY Gate)
  Please select your suitable date below:
==========================================

                December 2025
------------------------------------------
Su  Mo  Tu  We  Th  Fr  Sa
     1   2   3   4   5   6
 7   8   9  10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30  31
------------------------------------------
Select a date (1-31): 13

Selected date: December 13, 2025
```

```
============================================================
               OFFERS FOR DECEMBER 13, 2025
============================================================
 Day: Saturday

------------------------------------------------------------
  HOLIDAY OFFER - DECEMBER 13, 2025 (FRIDAY/SATURDAY)
------------------------------------------------------------
1) Fantasy Kingdom
2) Water Kingdom
------------------------------------------------------------
Choose an offer: 1

=============== PARK SELECTION ===============
1) Fantasy Kingdom
2) Entry/Exit System
3) Exit Program
==============================================
> 1
```

```
> 1

============== WELCOME Mahadi TO FANTASY KINGDOM ==============

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
>
```

```
Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
> 1

Select which Ride List:
1) Adult Rides
2) Children Rides
3) Heritage Sites
0) Back to Main Menu
> 1

================== ADULT RIDES ==================
 1) Roller Coaster          Queue:  0  Wait: 0 min
 2) Magic Carpet            Queue:  0  Wait: 0 min
 3) Bumper Cars             Queue:  0  Wait: 0 min
 4) Flying Disco            Queue:  0  Wait: 0 min
 5) Ferris Wheel            Queue:  0  Wait: 0 min
 6) Pirate Ship             Queue:  0  Wait: 0 min
 7) VR Zone                 Queue:  0  Wait: 0 min
 8) Horror House            Queue:  0  Wait: 0 min
 9) Tornado 360             Queue:  0  Wait: 0 min
10) Santa Maria             Queue:  0  Wait: 0 min
11) Wheely Bird             Queue:  0  Wait: 0 min
12) ZuZu Train              Queue:  0  Wait: 0 min
13) Heritage Coaster        Queue:  0  Wait: 0 min
================================================
```

```
Select Ride Type:
1) Adult Ride
2) Children Ride
3) Heritage Visit
0) Back to Main Menu
> 2

========================================
[ACCESS DENIED]
========================================
Child rides are NOT permitted for you.
You can enjoy: Adult Rides & Heritage Sites
========================================

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
>
```

```
Enter your name: Mahadi
Enter your Age: 22
Enter your Height (ft): 5.7
1) Adult Ride
2) Children Ride
3) Heritage Visit
0) Back to Main Menu
> 1

================== ADULT RIDES ==================
 1) Roller Coaster          Queue:  0  Wait: 0 min
 2) Magic Carpet            Queue:  0  Wait: 0 min
 3) Bumper Cars             Queue:  0  Wait: 0 min
 4) Flying Disco            Queue:  0  Wait: 0 min
 5) Ferris Wheel            Queue:  0  Wait: 0 min
 6) Pirate Ship             Queue:  0  Wait: 0 min
 7) VR Zone                 Queue:  0  Wait: 0 min
 8) Horror House            Queue:  0  Wait: 0 min
 9) Tornado 360             Queue:  0  Wait: 0 min
10) Santa Maria             Queue:  0  Wait: 0 min
11) Wheely Bird             Queue:  0  Wait: 0 min
12) ZuZu Train              Queue:  0  Wait: 0 min
13) Heritage Coaster        Queue:  0  Wait: 0 min
================================================

Select Adult Ride (1-13) or 0 to cancel: 3

========================================
[COMPLETED] RIDE COMPLETED
========================================
Ride: Bumper Cars
Duration: 5 minutes
Remaining rides: 11
========================================
```

```
--- Updated Adult Rides List ---

================= ADULT RIDES =================
 1) Roller Coaster          Queue:  0  Wait: 0 min
 2) Magic Carpet            Queue:  0  Wait: 0 min
 3) Bumper Cars             [COMPLETED]
 4) Flying Disco            Queue:  0  Wait: 0 min
 5) Ferris Wheel            Queue:  0  Wait: 0 min
 6) Pirate Ship             Queue:  0  Wait: 0 min
 7) VR Zone                 Queue:  0  Wait: 0 min
 8) Horror House            Queue:  0  Wait: 0 min
 9) Tornado 360             Queue:  0  Wait: 0 min
10) Santa Maria             Queue:  0  Wait: 0 min
11) Wheely Bird             Queue:  0  Wait: 0 min
12) ZuZu Train              Queue:  0  Wait: 0 min
13) Heritage Coaster        Queue:  0  Wait: 0 min
================================================

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
>
```

```
------------------------------------------------

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
> 3

[INFO] Remaining rides: 11 out of 12

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
>
```

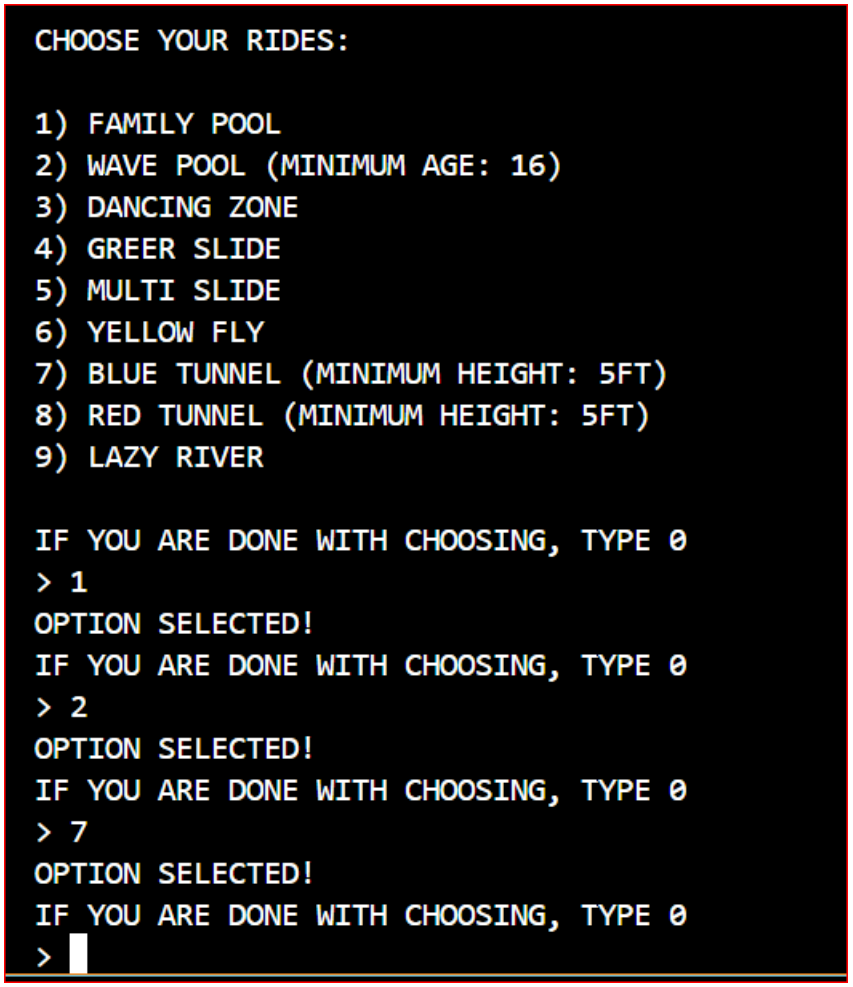
```
[INFO] Remaining rides: 11 out of 12

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
> 4
  1) Bumper Cars

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
>
```

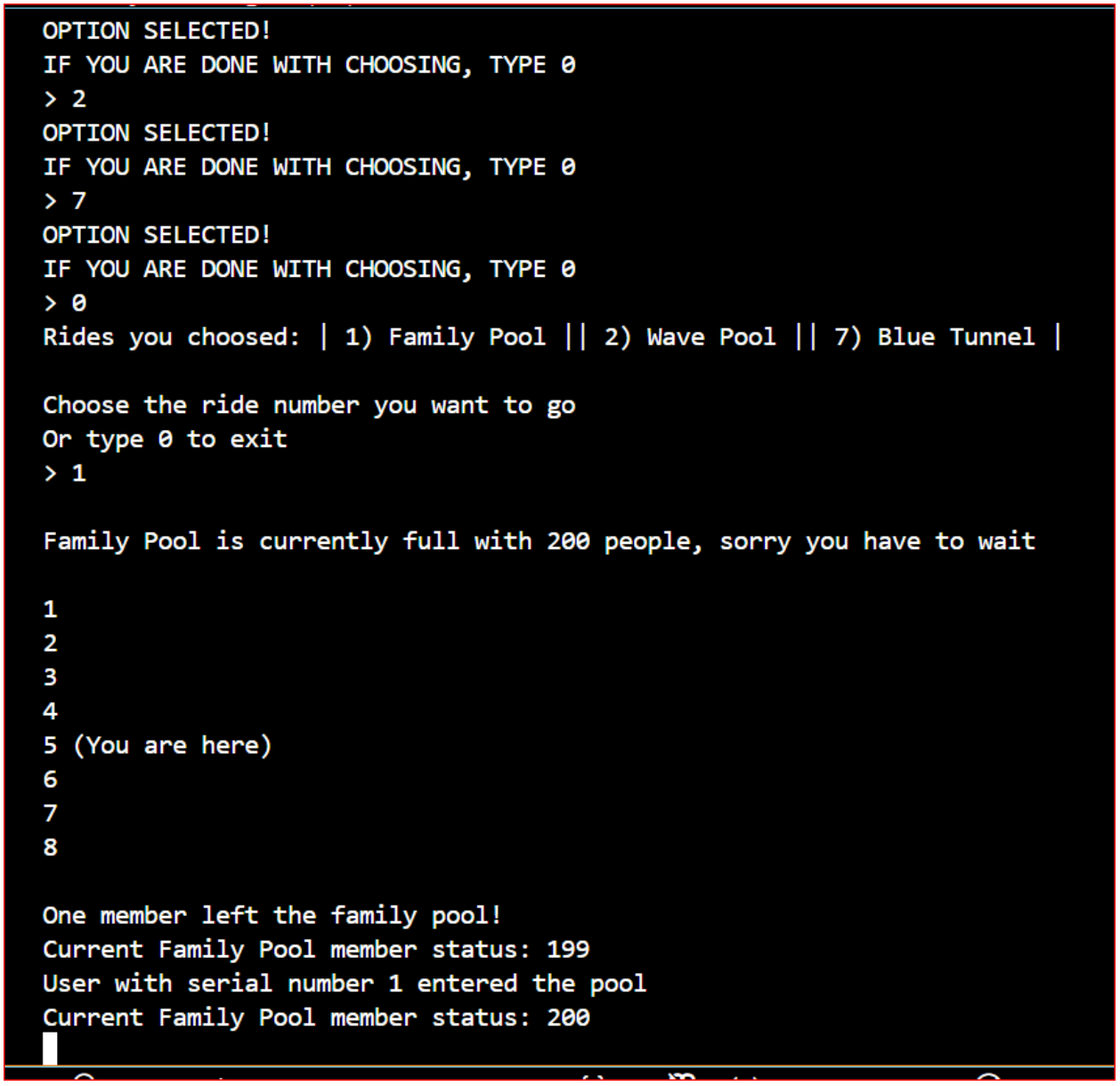
```
Select Ride Type:
1) Adult Ride
2) Children Ride
3) Heritage Visit
0) Back to Main Menu
> 3

================= HERITAGE SITES ==================
 1) Sitakot Bihar           (Visiting Only)
 2) Kantajir Mandir         (Visiting Only)
 3) Ahsan Manzil            (Visiting Only)
 4) Chunakhola Mosque       (Visiting Only)
 5) National Monument       (Visiting Only)
 6) Puthia Palace           (Visiting Only)
 7) Shat Gombuj Mosque      (Visiting Only)
 8) Paharpur                (Visiting Only)
===================================================

Select Heritage Site (1-8) or 0 to cancel: 3

========================================
[COMPLETED] VISIT COMPLETED
========================================
Site: Ahsan Manzil
Thank you for visiting!
========================================
```

```
--- Updated Heritage Sites List ---

================== HERITAGE SITES ==================
 1) Sitakot Bihar           (Visiting Only)
 2) Kantajir Mandir         (Visiting Only)
 3) Ahsan Manzil            [VISITED]
 4) Chunakhola Mosque       (Visiting Only)
 5) National Monument       (Visiting Only)
 6) Puthia Palace           (Visiting Only)
 7) Shat Gombuj Mosque      (Visiting Only)
 8) Paharpur                (Visiting Only)
====================================================

Fantasy Kingdom Menu:
1) View Ride List
2) Take a Ride
3) Show Remaining Rides
4) Ride History
5) Exit Fantasy Kingdom
>
```

```
 Fantasy Kingdom Menu:
 1) View Ride List
 2) Take a Ride
 3) Show Remaining Rides
 4) Ride History
 5) Exit Fantasy Kingdom
 > 5

 Exiting Fantasy Kingdom...

 ================================================================
                 WHAT DO YOU WANT TO DO NEXT?
 ================================================================
 1) Go to Fantasy Kingdom
 2) Go to Water Kingdom
 3) Entry/Exit System
 4) View Calendar
 5) View Offers
 6) Go to restaurant
 7) Park your vehicle
 8) Exist program
 ================================================================
 > 2
```


CHOOSE YOUR RIDES:
1) FAMILY POOL
2) WAVE POOL (MINIMUM AGE: 16)
3) DANCING ZONE
4) GREER SLIDE
5) MULTI SLIDE
6) YELLOW FLY
7) BLUE TUNNEL (MINIMUM HEIGHT: 5FT)
8) RED TUNNEL (MINIMUM HEIGHT: 5FT)
9) LAZY RIVER


CHOOSE YOUR RIDES:
1) FAMILY POOL
2) WAVE POOL (MINIMUM AGE: 16)
3) DANCING ZONE
4) GREER SLIDE
5) MULTI SLIDE
6) YELLOW FLY
7) BLUE TUNNEL (MINIMUM HEIGHT: 5FT)
8) RED TUNNEL (MINIMUM HEIGHT: 5FT)
9) LAZY RIVER
IF YOU ARE DONE WITH CHOOSING, TYPE 0
> 1
OPTION SELECTED!
IF YOU ARE DONE WITH CHOOSING, TYPE 0
> 2
OPTION SELECTED!
IF YOU ARE DONE WITH CHOOSING, TYPE 0
> 7
OPTION SELECTED!
IF YOU ARE DONE WITH CHOOSING, TYPE 0
>

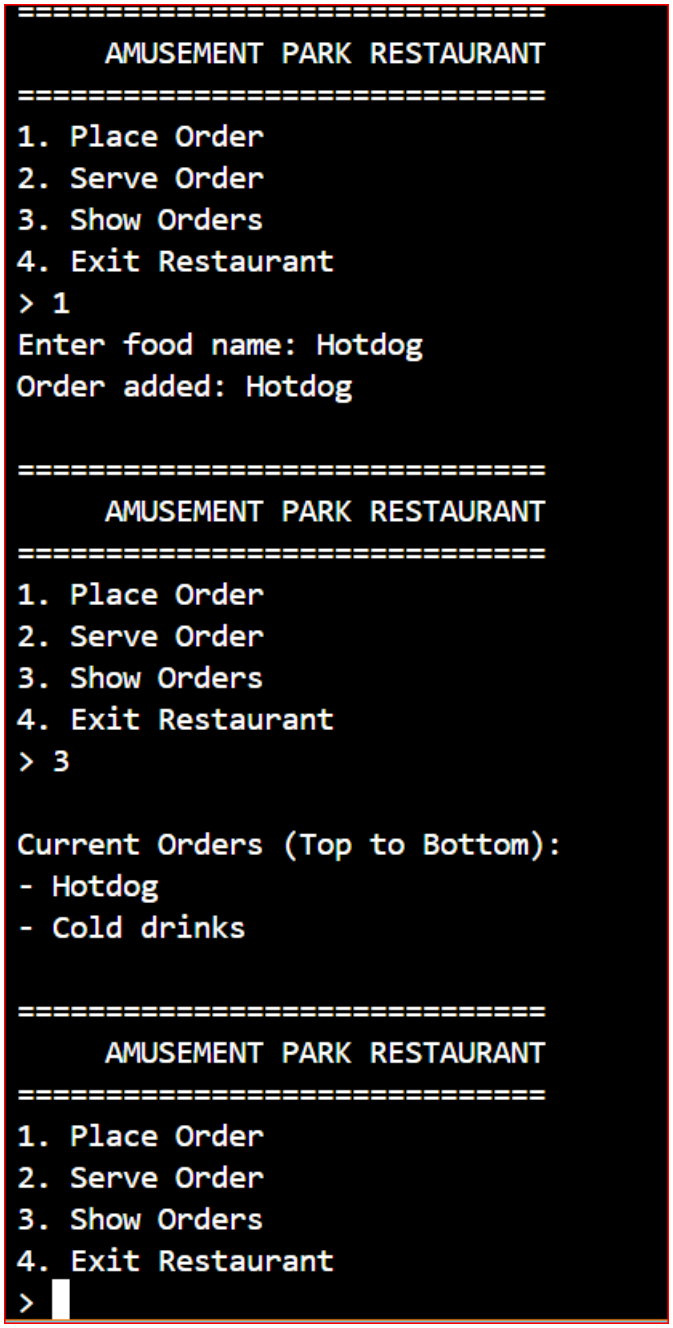

OPTION SELECTED!
IF YOU ARE DONE WITH CHOOSING, TYPE 0
> 2
OPTION SELECTED!
IF YOU ARE DONE WITH CHOOSING, TYPE 0
> 7
OPTION SELECTED!
IF YOU ARE DONE WITH CHOOSING, TYPE 0
> 0
Rides you choosed: | 1) Family Pool || 2) Wave Pool || 7) Blue Tunnel |
Choose the ride number you want to go
Or type 0 to exit
> 1
Family Pool is currently full with 200 people, sorry you have to wait
1
2
3
4
5 (You are here)
6
7
8
One member left the family pool!
Current Family Pool member status: 199
User with serial number 1 entered the pool
Current Family Pool member status: 200

```
One member left the family pool!
Current Family Pool member status: 199
User with serial number 4 entered the pool
Current Family Pool member status: 200
5 (You are here)
6
7
8

One member left the family pool!
Current Family Pool member status: 199
User with serial number 5 entered the pool
Current Family Pool member status: 200

you are in!

Ride left: | 2) Wave Pool || 7) Blue Tunnel |

Choose the ride number you want to go
Or type 0 to exit
>
```

```
Choose the ride number you want to go
Or type 0 to exit
> 2

Wave Pool is currently full with 100 people, sorry you have to wait

1
2
3
4
5
6
7
8 (You are here)
9

One member left the Wave Pool!
Current Wave Pool member status: 99
User with serial number 1 entered the pool
Current Wave Pool member status: 100
2
3
4
5
6
7
8 (You are here)
9
```

```
Choose the ride number you want to go
Or type 0 to exit
> 7

Blue Tunnel is currently full with 30 people, sorry you have to wait

1
2
3
4
5 (You are here)
6
7

One member left the Blue Tunnel!
Current Blue Tunnel member status: 29
User with serial number 1 entered Blue Tunnel
Current Blue Tunnel member status: 30
2
3
4
5 (You are here)
6
7
```

```
Choose the ride number you want to go
Or type 0 to exit
> 0

========================================================================
                WHAT DO YOU WANT TO DO NEXT?
========================================================================
1) Go to Fantasy Kingdom
2) Go to Water Kingdom
3) Entry/Exit System
4) View Calendar
5) View Offers
6) Go to restaurant
7) Park your vehicle
8) Exist program
========================================================================
> 6

==============================
     AMUSEMENT PARK RESTAURANT
==============================
1. Place Order
2. Serve Order
3. Show Orders
4. Exit Restaurant
>
```


==============================
AMUSEMENT PARK RESTAURANT
==============================
1. Place Order
2. Serve Order
3. Show Orders
4. Exit Restaurant
> 1
Enter food name: Hotdog
Order added: Hotdog
==============================
AMUSEMENT PARK RESTAURANT
==============================
1. Place Order
2. Serve Order
3. Show Orders
4. Exit Restaurant
> 3
Current Orders (Top to Bottom):
- Hotdog
- Cold drinks
==============================
AMUSEMENT PARK RESTAURANT
==============================
1. Place Order
2. Serve Order
3. Show Orders
4. Exit Restaurant
>

```
Exiting Restaurant...

=================================================================
               WHAT DO YOU WANT TO DO NEXT?
=================================================================
1) Go to Fantasy Kingdom
2) Go to Water Kingdom
3) Entry/Exit System
4) View Calendar
5) View Offers
6) Go to restaurant
7) Park your vehicle
8) Exist program
=================================================================
> 7

==============================
      AMUSEMENT PARK PARKING
==============================
1. Park Car
2. Remove Car
3. Show Parked Cars
4. Exit Parking
>
```

```
==============================
      AMUSEMENT PARK PARKING
==============================
1. Park Car
2. Remove Car
3. Show Parked Cars
4. Exit Parking
> 1
Enter car number: 232
Car 232 parked successfully.

==============================
      AMUSEMENT PARK PARKING
==============================
1. Park Car
2. Remove Car
3. Show Parked Cars
4. Exit Parking
> 2
Car 232 left the parking.

==============================
      AMUSEMENT PARK PARKING
==============================
1. Park Car
2. Remove Car
3. Show Parked Cars
4. Exit Parking
> 3
Parking is empty.
```

```
==============================
      AMUSEMENT PARK PARKING
==============================
1. Park Car
2. Remove Car
3. Show Parked Cars
4. Exit Parking
> 4
Exiting Car Parking System...
```

Chapter 4

Engineering Standards and Mapping

This chapter evaluates the project through the lens of engineering standards, societal and environmental impacts, and ethical considerations. It also maps the system to relevant Course Outcomes, Program Outcomes, Complex Engineering Problems, and Engineering Activities.

4.1 Impact on Society, Environment and Sustainability

The development of the Amusement Park Management System demonstrates how algorithmic design and structured problem-solving directly contribute to social efficiency, operational safety, and sustainable resource usage. As amusement parks serve thousands of visitors in a single day, inefficient management systems often generate overcrowding, confusion, service delays, and unnecessary waste. By introducing a data-driven automation framework, this project showcases how computational models—built upon queues, stacks, linked lists, and heaps—can significantly streamline operations and improve overall societal experience.

The automation of core workflows reduces human dependency in high-stress environments, lowering the chances of manual error and increasing consistency in service delivery. Ride queue optimization improves fairness, transparency, and visitor safety by ensuring deterministic ordering and predictable wait times. The systematic locker and parking assignment models minimize resource contention, reduce operational conflicts, and support smoother visitor flow throughout the facility.

From a broader perspective, this project illustrates the potential of technology to enhance the quality of recreational spaces. Large-scale public environments benefit greatly from structured engineering solutions, enabling them to accommodate growing populations while maintaining service quality. Thus, the system not only solves a technical problem but also contributes to societal well-being by promoting efficient, enjoyable, and safe experiences for all visitors

4.1.1 Impact on Life

An efficient amusement park management system directly improves visitors' quality of life by minimizing the time spent waiting in queues, searching for lockers, or navigating crowded parking lots. Long waiting times are not just inconvenient; they contribute to psychological stress, fatigue, and reduced enjoyment—particularly for families, elderly visitors, and children. By reducing operational inefficiencies, the system creates a more structured and pleasant environment in which individuals can focus on enjoyment rather than logistical concerns.

Beyond visitor experiences, the system significantly benefits employees by reducing the cognitive load associated with manual queue tracking, inconsistent locker distribution, and subjective offer calculations. This allows staff to concentrate on supervision, safety, emergency response, and hospitality—activities that genuinely enhance visitor experience.

In short, improving the operational flow of an amusement park results in measurable quality-of-life improvements for all stakeholders involved.

4.1.2 Impact on Society & Environment

Digital transformation of amusement parks can have wide-ranging societal implications. Organized queues reduce crowding, which contributes to improved public safety—an important concern in high-density environments. Automated decision-making removes human bias and promotes equitable treatment for all visitors, reinforcing social fairness and transparency.

Environmentally, the elimination of paper-based tickets, manual logs, physical queue slips, and paper-based order systems significantly reduces waste generation. Over time, such reductions in paper and ink usage diminish the environmental footprint of daily operations. Furthermore, optimized resource allocation—such as efficient parking slot usage—reduces unnecessary fuel consumption associated with searching for parking, indirectly lowering carbon emissions.

Thus, the system supports both social and environmental sustainability through efficiency, fairness, and waste reduction.

4.1.3 Ethical Aspects

Ethical considerations are central to the design of this system, particularly in areas involving fairness, transparency, and responsible data handling. The queue-based components of the Fantasy Kingdom and Water Kingdom enforce strict **FIFO behavior**, ensuring that all visitors are served in the order they arrive. VIP processing, implemented through a separate stack structure, represents an intentional and explicitly communicated priority mechanism rather than hidden or arbitrary favoritism. Similarly, age- and height-based ride eligibility in the Water Kingdom is handled algorithmically, ensuring consistent and unbiased enforcement of safety rules.

The system also treats visitor information ethically. Although the prototype stores only minimal, non-sensitive data (such as name, age, and height) and does not retain permanent profiles, it processes all inputs solely for operational purposes. No unnecessary data is collected, stored, or shared, which aligns with responsible data-minimization principles.

By automating key decision points—such as admission sequencing, ride eligibility, queue movement, and service selection—the system reduces opportunities for subjective human interference. This promotes fairness and reduces the risk of inconsistent judgment by staff. Overall, the system demonstrates how ethical engineering practices can be embedded directly into algorithmic logic, fostering transparency, trust, and safe operation within the amusement park environment.

4.1.4 Sustainability Plan

The sustainability of this system is supported by its modular structure and lightweight data-structure–based design. Each subsystem—Calendar, Offers, Entry/Exit, Fantasy Kingdom, and Water Kingdom—is implemented independently, allowing future enhancements without altering the core architecture. Features

such as database integration, graphical interfaces, real-time analytics, or IoT-based ride monitoring can be added incrementally, ensuring long-term adaptability.

Operational sustainability is also achieved through efficient resource usage. The system relies on simple linear data structures with minimal memory overhead, enabling deployment on low-power or legacy hardware. By replacing manual processes with automated digital workflows, the system reduces paper usage and minimizes operational redundancy. This contributes to both environmental responsibility and long-term maintainability.

Overall, the design supports continued growth, scalability, and efficient operation, ensuring that the system remains viable and adaptable as amusement-park requirements evolve over time.

4.2 Project Management and Team Work

The development of this project was carried out through collaborative teamwork involving five members. At the initial stage, **one team member proposed the core project idea(Mahadi Rahman Jihad-008)**, identified the problem domain, and initiated the overall concept of modeling amusement park operations using Data Structure principles. Following this, **all team members were equally involved** in supporting and developing the project.

Each member actively participated in **collecting data, visiting the selected location, and observing real-world operational workflows**. The information gathered from these activities was shared among the entire team, enabling a common understanding of the system requirements and practical constraints. Based on these shared insights, the team collectively analyzed how real amusement park processes could be represented using queues, linked lists, and stack logic.

Throughout the project, responsibilities were shared evenly, and decisions were made through group discussion and mutual agreement. Regular communication ensured coordination, consistency, and alignment with project objectives. This collaborative approach strengthened teamwork, responsibility sharing, and collective problem-solving skills, and it strongly aligns with **CO3**, reflecting effective team-based project execution similar to professional engineering environments.

4.2.1 Requirement analysis

Requirement analysis was conducted through direct field study and stakeholder interaction to understand real-world amusement park operations and translate them into a Data-Structure-based system. For this purpose, the project team visited **Fantasy Kingdom Amusement Park, Savar, Dhaka, Bangladesh**, as it represents a large-scale, real-life amusement park environment suitable for this study.

Before conducting the field visit, the team **formally applied for permission through the university** by submitting an official application explaining the academic purpose of the visit. Upon receiving approval from the university authority, the team proceeded with the site visit. This ensured that the data collection process followed proper academic and ethical guidelines.

During the visit, the team **individually met and communicated with the Fantasy Kingdom Manager and the Water Kingdom Manager** to gain insights into daily operational workflows. These discussions focused on ride queue handling, visitor flow management, restaurant service order, locker usage, and parking arrangements. The managers provided practical explanations of how visitors are currently managed, common challenges faced during peak hours, and areas where structured automation could improve efficiency.

In addition to discussions, the team conducted **direct observation** of publicly accessible operational areas, including ride queues, Water Kingdom attractions, food service areas, and parking zones. These observations helped identify operational bottlenecks and understand how real-life processes naturally follow Data Structure behavior such as FIFO queues and LIFO-based resource handling.

DAFFODIL INTERNATIONAL UNIVERSITY

Department of Computer Science & Engineering (CSE)
DSC, Birulia, Savar, Dhaka–1216, Bangladesh

Date: 25 Nov. 2025

To
The Managing Authority
Fantasy Kingdom Amusement Park
Ashulia, Savar, Dhaka

Subject: Request for Permission to Collect Operational Data

Respected Sir/Madam,

I, **Mahadi Rahman Jihad**, Student ID **251-15-008**, on behalf of **Team 5 Dots**, studying B.Sc. in CSE at Daffodil International University, am enrolled in **Data Structure Lab (CSE 124)** under the supervision of **Mr. Md. Jakaria Zobair**.

For my academic project titled **"Amusement Park Management System"**, I respectfully request permission to collect non-sensitive operational information from Fantasy Kingdom, including:

- Ride information (name, capacity, duration, queue type)
- Ticket types and pricing
- Visitor flow
- Daily ride usage
- Maintenance schedule

No personal or confidential data will be collected. All collected information will be used strictly for academic purposes.

I kindly request your approval to collect this information. Your cooperation will be sincerely appreciated.

Sincerely,

Mahadi Rahman Jihad
B.Sc. in CSE, FSIT
Daffodil International University
Email: 251-15-008@diu.edu.bd
Phone: +880 1750-382019

Approved & Verified By

Course Teacher	**Department Office (CSE)**
Mr. Md. Jakaria Zobair	Authorized Officer:
Lecturer, CSE	
DIU	Signature & Seal: 25/11/25
Signature: Jakaria 25/11/2025	

Scanned with CamScanner

Figure 4.1 Application

Stakeholder Identification

Based on field visits and discussions, the following stakeholders were identified:

- **Person-1: Fantasy Kingdom Manager**
  Provided information regarding ride queue management, visitor flow, and peak-time operational challenges.
- **Person-2: Water Kingdom Manager**
  Explained Water Kingdom operations, eligibility rules, crowd control, and ride scheduling processes.
- **Person-3: Visitors**
  Primary users of the system who experience ride queues, restaurant service, locker usage, and parking facilities.
- **Person-4: Operational Staff**
  Includes ride operators, restaurant staff, and parking attendants who directly interact with visitors and follow operational workflows.

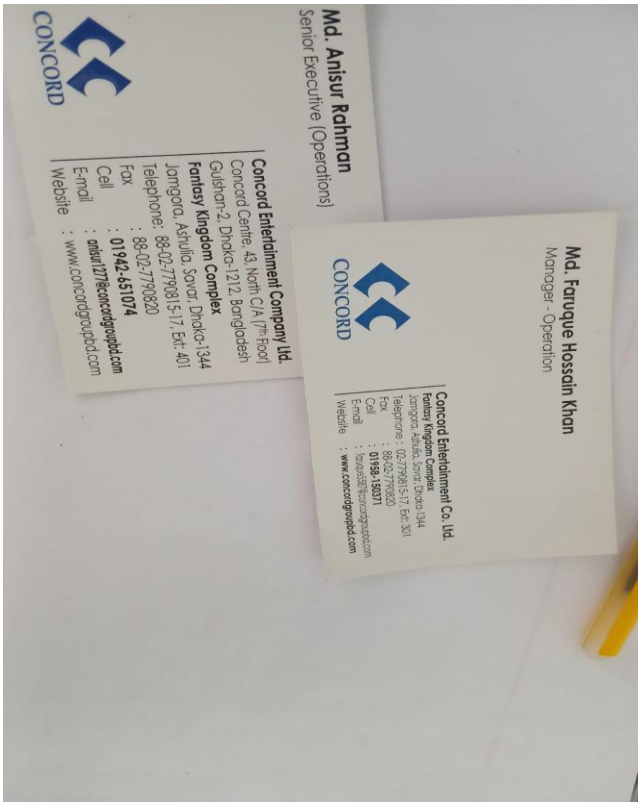


Figure 4.2 : Stakeholder Name

- **Person-5: Project Team (System Designers)**
  Responsible for requirement analysis, system modeling, and mapping real-world workflows to appropriate Data Structures.

Figure 4.3 : Visit

Figure 4.4: Entrance slip

Understanding stakeholder roles helped define system priorities, functional behavior, and technical constraints.

Permission and Ethical Compliance

The field study was conducted **after receiving formal permission from the university authority**. All observations were limited to **publicly accessible areas** of the amusement park. No confidential documents, internal records, or personal data were collected. Photographs taken during the visit were used solely as **academic evidence of requirement analysis** and do not interfere with park operations or privacy.
website to visite:[https://fantasykingdom.net/]

Identified Requirements

Based on stakeholder discussion and observation, the following requirements were identified:

Functional Requirements

- FIFO-based ride queue management for Fantasy Kingdom
- Visitor eligibility checking (age/height) for Water Kingdom rides
- Structured restaurant order handling
- Sequential parking allocation
- Real-time queue status and operational flow simulation

Technical Requirements

- Implementation using **C programming language**
- Use of **queues, linked lists, and stack logic**
- Pointer-based dynamic memory management
- Efficient enqueue and dequeue operations
- Modular system design for independent subsystems

These requirements directly influenced the system architecture and Data Structure selection used in the project.

4.3 Complex Engineering Problem

This amusement park management system satisfies the criteria of a Complex Engineering Problem (CEP) as defined by the accreditation guidelines. The system involves multiple competing constraints, interdependent modules, abstract data models, and performance-critical decision-making. Solving such a problem requires deep knowledge of data structures, algorithm design, and operational modeling—central characteristics of complex engineering problems.

4.3.1 Mapping of Program Outcome

In this section, have provided a mapping of the problem and provided solution with targeted Program Outcomes (PO's).

Table 4.1: Justification of Program Outcomes

CO'	PO's	Justification
CO1	PO1	This Course Outcome supports PO3 by enabling students to design structured, computational solutions using fundamental data structures to address real-life problems with complex engineering characteristics and demonstrating a core competency in engineering design and development.
CO2	PO2	This Course Outcome supports PO2 by engaging students in identifying and analyzing real world problems, formulating appropriate abstract data types, and applying engineering principles and modern tools to develop and validate efficient solutions and thereby demonstrating structured problem-solving and analytical capabilities.
CO3	PO3	This Course Outcome supports PO9 by requiring students to collaboratively apply data structures in solving real-world problems, fostering teamwork, communication, and mutual contribution in diverse and multidisciplinary teams—skills essential for professional engineering practice.

4.3.2 Complex Problem Solving

This mapping demonstrates how the system addresses complex problem-solving criteria through its design and implementation.

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	EP2 Range of Conflicting Requirements	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of Applicable Codes	EP6 Extent of Stake-holder Involvement	EP7 Inter-dependence
√	√	N/A	N/A	N/A	√	N/A

EP1 - Depth of Knowledge

This task required a deep understanding of fundamental data structures such as queues, stacks, and linked lists, along with efficient memory handling. These concepts were essential for accurately modeling real-world amusement park workflows, including visitor flow management, ride scheduling, and dynamic state changes. Proper use of these structures ensured realistic simulations, optimized performance, and reliable handling of complex, time-dependent interactions within the system.

EP2 - Range of Conflicting Requirements

The design process involved balancing fairness, processing speed, and resource limitations while implementing queue management, locker allocation, and order processing systems. This required resolving

practical trade-offs to ensure efficient operation, equitable user experiences, and optimal use of available resources under real-world constraints.

EP6 – Extent of Stakeholder Involvement

The project was shaped through an on-site visit, with the entire system design guided by the practical needs of visitors, staff, and management. System functionality was carefully planned to address the expectations of all stakeholder groups simultaneously, ensuring a balanced solution that supported user experience, operational efficiency, and administrative control.

4.3.3 Engineering Activities

This table shows how the project reflects various engineering activities through its workflow, design choices, and implementation.

Table 4.3: Mapping with complex engineering activities.

EA1 Range of resources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society and environment	EA5 Familiarity
N/A	N/A	N/A	N/A	N/A

Chapter 5

Conclusion

This chapter summarizes the overall achievements of the project and reflects on its academic and practical significance. It also highlights the system's limitations and proposes directions for future improvement and extension.

5.1 Summary

The Amusement Park Management System developed in this project demonstrates how fundamental linear data structures can be effectively translated into a functional, multi-module software solution. Queues, stacks, and linked lists—typically studied as abstract concepts—were applied directly to automate key amusement-park operations such as visitor entry sequencing, ride management, ride history tracking, menu handling, and category-based navigation. This practical translation reflects the central objective of the Data Structure course: enabling students to convert theoretical ADTs into solutions that address real operational problems.

Each subsystem—Entry/Exit, Fantasy Kingdom, Water Kingdom, Offers, and Calendar—was designed and implemented independently, then integrated into a unified workflow. This modular approach ensures clarity, maintainability, and scalability while mirroring industry-standard software engineering practices. By structuring the system around well-defined data-handling units, the project achieves predictable behavior, reduced complexity, and ease of future enhancement.

From an academic standpoint, the project successfully meets the intended course outcomes by demonstrating problem analysis, selecting appropriate data structures, and applying algorithmic reasoning to real scenarios. From an engineering perspective, the system illustrates how performance, memory efficiency, and reliability influence design decisions in practical implementations.

In summary, the project not only fulfills its functional goals but also provides meaningful hands-on experience with data structures, showing their relevance and applicability in designing efficient, real-world software systems.

5.2 Limitation

Although the system successfully models core amusement-park operations using linear data structures, several limitations arise from the project's scope, simplified simulation logic, and the constraints of a console-based C environment. First, the system operates entirely in volatile memory; aside from minimal temporary name reading, there is **no persistent database or long-term storage**, meaning visitor history, ride statistics, and operational analytics are not retained across sessions.

Second, the console interface limits user experience and operational clarity. Queue lengths, ride statuses, and attraction categories are presented purely as text, with no graphical visualization of real-time crowd

movement. This restricts situational awareness and does not reflect the responsiveness required in an actual amusement-park control system.

A significant limitation is the absence of concurrency. Real parks involve many actions occurring simultaneously—multiple visitors joining lines, completing rides, or switching between zones. Because the implementation processes all events **sequentially**, it cannot model overlapping activities, peak-time behavior, or dynamic load conditions. The use of Sleep() for simulation further reinforces a single-threaded, time-blocking workflow.

Scalability is another constraint. While queues, stacks, and linked lists scale conceptually, the current implementation is tuned for moderate visitor volume. Running thousands of visitors or large interconnected ride networks would require redesigning memory management, adding dynamic allocation safeguards, and optimizing traversal operations.

Finally, several modules employ simplified operational logic. Water Kingdom ride flow uses a basic queue simulation rather than a full multi-ride ecosystem with lifeguard rules or real safety constraints. The Entry/Exit system uses fixed-size arrays and does not support advanced ticket rules or automated conflict resolution. Similarly, the offer engine is intentionally simple and does not incorporate surge pricing, holiday analytics, or business intelligence.

These limitations do not diminish the system's academic value; instead, they highlight the difference between a course-level prototype and an industry-oriented amusement-park management platform while identifying clear directions for future expansion.

5.3 Future Work

The current system provides a functional foundation, but several enhancements can transform it into a fully operational and industry-ready amusement park management solution. Future improvements align across four major development areas: **data management**, **interface enhancement**, **technical capabilities**, and **intelligent automation**.

1. Persistent Storage and Data Analytics

One of the highest-priority improvements is the introduction of long-term storage mechanisms.
Potential extensions include:

- File-based logging for ride history, queue statistics, and visitor profiles
- Integration of relational databases (e.g., MySQL/SQLite) for historical analytics
- Automated generation of daily, weekly, and seasonal operational reports
- Storage-driven insights for business planning and resource allocation

Persistent data would allow the system to move beyond real-time simulation toward real operational decision support.

2. Graphical User Interface (GUI) and Multi-User Support

Replacing the console interface with a GUI would significantly improve usability and operational visibility.
Future enhancements may include:

- Desktop or web-based dashboards showing real-time queue lengths and ride status
- Visual mapping of Fantasy Kingdom and Water Kingdom attractions
- Dynamic menus for ride restrictions, offers, and visitor categories
- Multi-user access enabling different staff members to manage separate modules simultaneously

A GUI would make the system scalable for actual amusement park use.

3. Technical Enhancements and System Architecture Upgrades

To increase realism and performance, several technical improvements can be considered:

- **Multi-threading or event-driven simulation** to model multiple visitors acting concurrently
- **Refined memory management** for handling large visitor volumes
- **Introduction of advanced data structures:**
    - Hash tables for quick visitor lookup
    - Balanced trees for scheduling and resource allocation
    - Graphs for park navigation, routing, and congestion detection

These upgrades would bring the system closer to large-scale commercial implementations.

4. Integration With Emerging Technologies

Modern amusement parks leverage a range of technologies that can be incorporated into future versions:

- RFID-based ticketing and visitor tracking
- IoT-enabled sensors for locker availability and ride capacity monitoring
- Automated parking gates and barrier systems
- Mobile applications for queue updates, ride reservations, and digital ticket management

These additions would enable real-time responsiveness and enhance visitor experience.

5. Advanced Decision-Making and Optimization

The system may eventually include intelligent algorithms that support operational planning:

- Dynamic pricing based on day, time, and crowd levels
- Predictive analytics for ride demand forecasting
- Automated crowd management alerts
- Adaptive scheduling for high-demand attractions

Such features would elevate the system from an operational tool to an intelligent park management platform.

5.3.1 Conclusion

In summary, while the current prototype fulfills its academic objectives, it also serves as a strong base for future expansion. With targeted improvements in storage, user interface, concurrency, algorithms, and technology integration, the system can evolve into a sophisticated, real-world amusement park management solution.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[3] E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2nd ed. Silicon Press, 1993.

[4] Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th ed. Addison-Wesley, 2011.

[5] M. Weiss, *Data Structures and Algorithm Analysis in C*, 2nd ed. Addison-Wesley, 1997.

[6] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Wiley, 2012.

[7] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Pearson, 2012.

[8] D. Knuth, *The Art of Computer Programming*, Vol. 1–3, Addison-Wesley, 1997.

[9] R. Larson and J. Odoni, *Urban Operations Research*. Prentice-Hall, 1981.
(**Used for queueing systems, ride flow modeling, and operational analysis**)

[10] M. M. El-Taha and S. Stidham Jr., *Sample-Path Analysis of Queueing Systems*. Springer, 1999.
(**Used for understanding FIFO & priority queues**)

[11] R. Graham, R. L. Graham, and M. Grigni, *Priority Queues and Heaps*, MIT OpenCourseWare Lecture Notes, 2000.

[12] Pressman, R. S. and B. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. McGraw-Hill, 2014.

[13] Sommerville, I., *Software Engineering*, 10th ed. Pearson, 2015.

[14] K. E. Case, R. G. Desrochers, and J. M. Roberts, "Queue management strategies in recreational facilities," *Journal of Operations Management*, vol. 22, no. 3, pp. 185–199, 2004.

[15] R. Shumaker and M. Wilson, “Digital transformation in amusement parks: A survey of smart systems,” *International Journal of Entertainment Technology*, vol. 7, no. 4, pp. 42–58, 2021.

[16] T. Nguyen and L. Vo, “Priority-based resource allocation in public service environments,” *IEEE Access*, vol. 8, pp. 19022–19035, 2020.

[17] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, 2006.

[18] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Prentice Hall, 1988