FACULTY OF COMPUTING

UNIVERSITI TEKNOLOGI MALAYSIA

DATA STRUCTURE & ALGORITHM (SECJ2013)

SEMESTER 1 2025/2026

Mini Project Documentation
PKU Queueing System

By
Ahmad Munif bin Baharum (051221-11-0677) – Leader
Abdurrafiq bin Zakaria (050304-06-0447)
Najmuddin bin Kamarudin (051201-06-0211)

SECTION 05

Lecturer:
Dr. Pang Yee Yong

Date Submission:

16 January 2026
For lecturer use:

Description	Mark Distribution	Mark
Project Report: - System analysis - Design - Program code	 10 15 25	
Presentation & Demo	25	
System Prototype	25	
TOTAL	**100**	

Table of Contents

PART 1: INTRODUCTION

1.1 Synopsis Project

The **PKU Queueing System** is a C++ prototype developed to manage patient flow at Pusat Kesihatan UTM, where an automated process is vital for handling high daily volumes for General and Dental services. Every patient is assigned a unique, sequential three-digit ID beginning at **000**, ensuring a clear and transparent record of arrival. To handle medical urgency effectively, the system operates on a two-level triage logic, which is **Emergency** and **Normal**, where the "Call Next" function strictly prioritizes Emergency patients. This ensures that critical cases are attended to immediately, bypassing the standard arrival order until the Emergency queue is cleared.

Internally, the system demonstrates the practical application of fundamental data structures by using a **Linked List-based Queue (FIFO)** to manage the dynamic waiting line without wasting memory. The integration of **Priority Queue logic** allows the program to sort patients by their triage level, effectively moving Emergency patients to the front of the service line based on their urgency rather than just their arrival time. Additionally, a **Stack (LIFO)** is utilized to maintain a **History Log** of treated patients, enabling staff to trace back and review the most recent cases in reverse chronological order. This combination of structures provides a robust solution for simulating real-world medical queue management while maintaining administrative oversight.

1.2 Objective of the project

- **Building a Real-World Tool:** We aimed to create a C++ system that actually feels like a clinic environment, using a familiar 000-style ID sequence to track every visitor from the moment they walk in.

- **Managing the Crowd Smarter:** By using **Linked Lists**, we made sure the system stays light and fast. It grows as more patients arrive and shrinks as they are treated, so it never wastes memory.

- **Prioritizing What Matters:** We built in **Priority Queue logic** because, in a clinic, some cases can't wait. This ensures Emergency patients get care immediately, while others wait their turn fairly.

- **Remembering the Past:** We used a **Stack** to act as the system's "short-term memory," allowing staff to instantly see who they just treated without digging through old files.

PART 2: SYSTEM ANALYSIS AND DESIGN

2.1 System Requirements

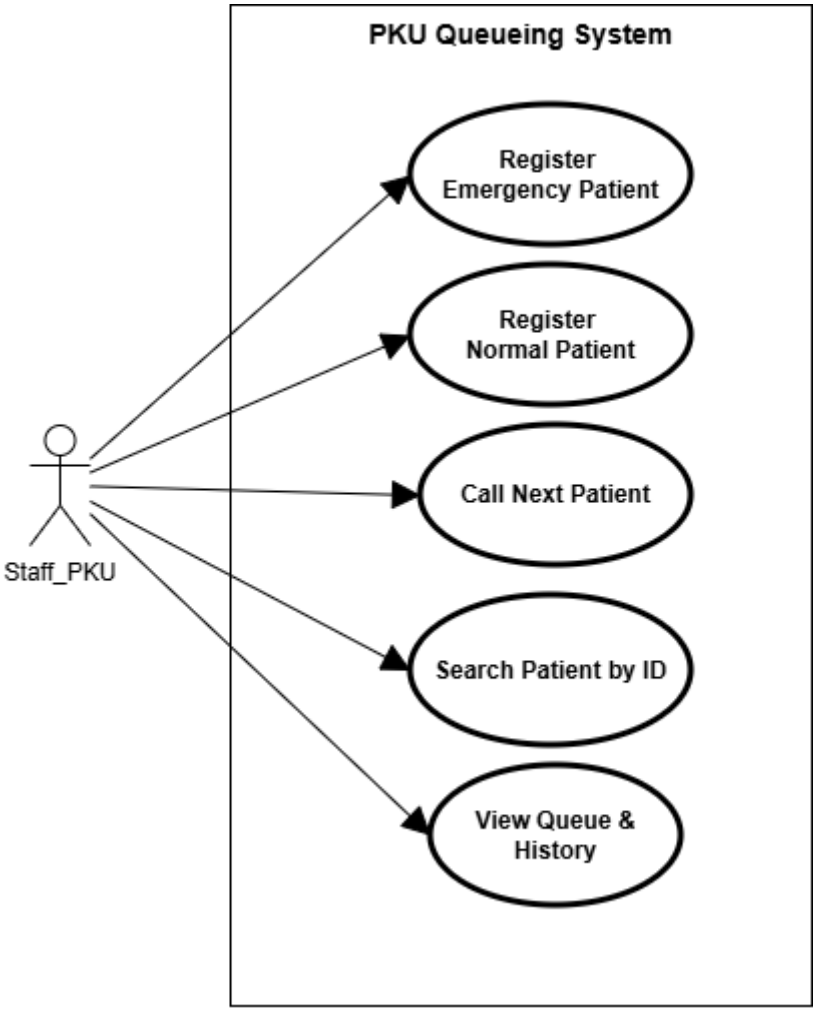

Figure 2.1 PKU Queueing System Use Case

Use Case description for PKU Queueing System

Actor	Task
Staff_PKU	Allows the staff to add patients' info to the queue

Detailed description for each Use Case

Use Case	Purpose
Register Emergency Patient	To generate a unique ID and prioritize urgent cases by placing them at the front of the queue.
Register Normal Patient	To generate a unique ID and store standard patient info at the end of the queue.
Call Next Patient	To dequeue the next patient and automatically archive the previous one into the History Stack and history.txt.
Search Patient by ID	To find a specific patient currently waiting in memory by performing a Sequential Search.
View Queue & History	To display the real-time priority list (Queue) and the session records (Stack).

2.2 System Design

2.2.1 Class Diagram

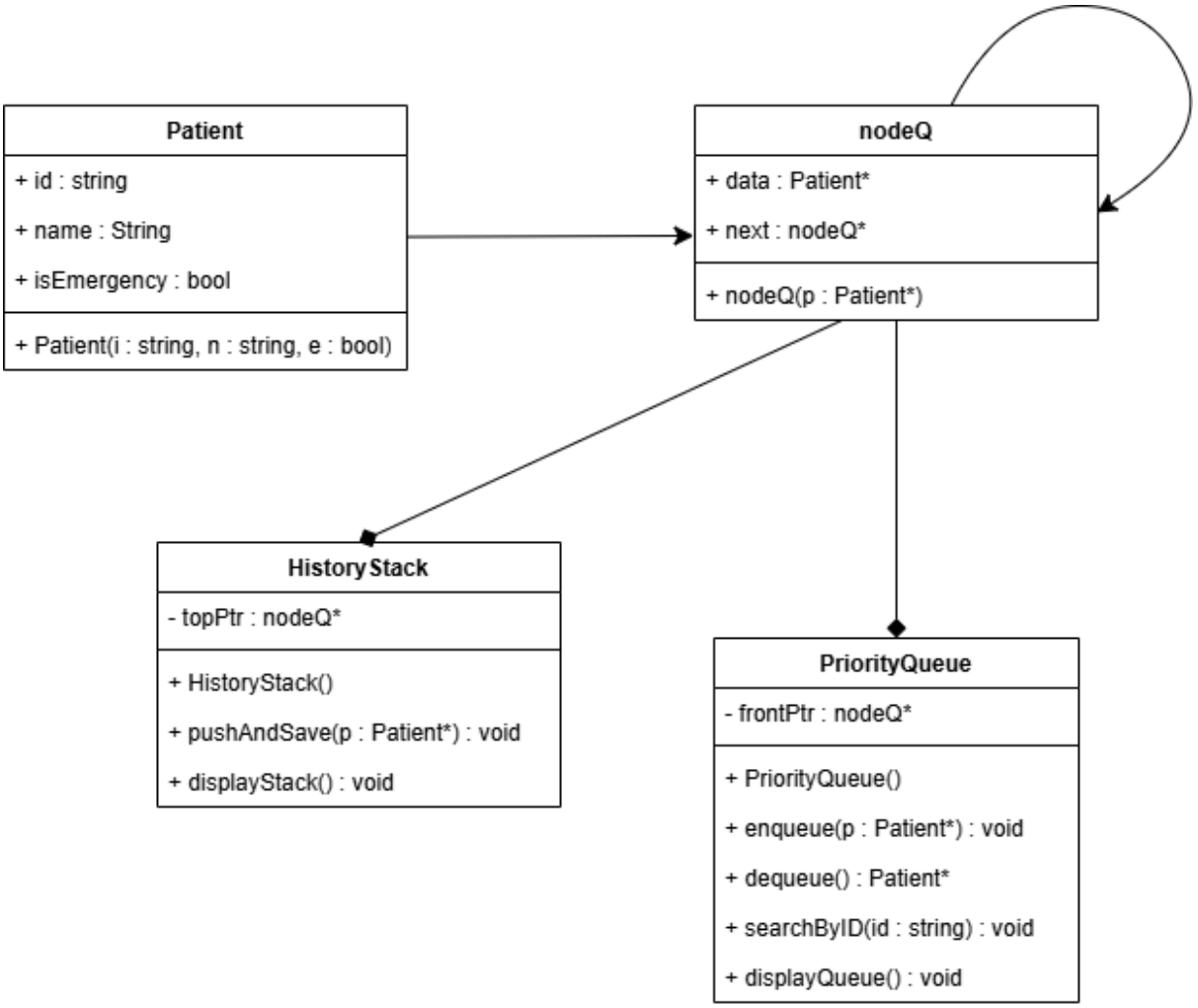

Figure 2.2 Class Diagram of PKU Queueing System

2.2.2 Flow Chart

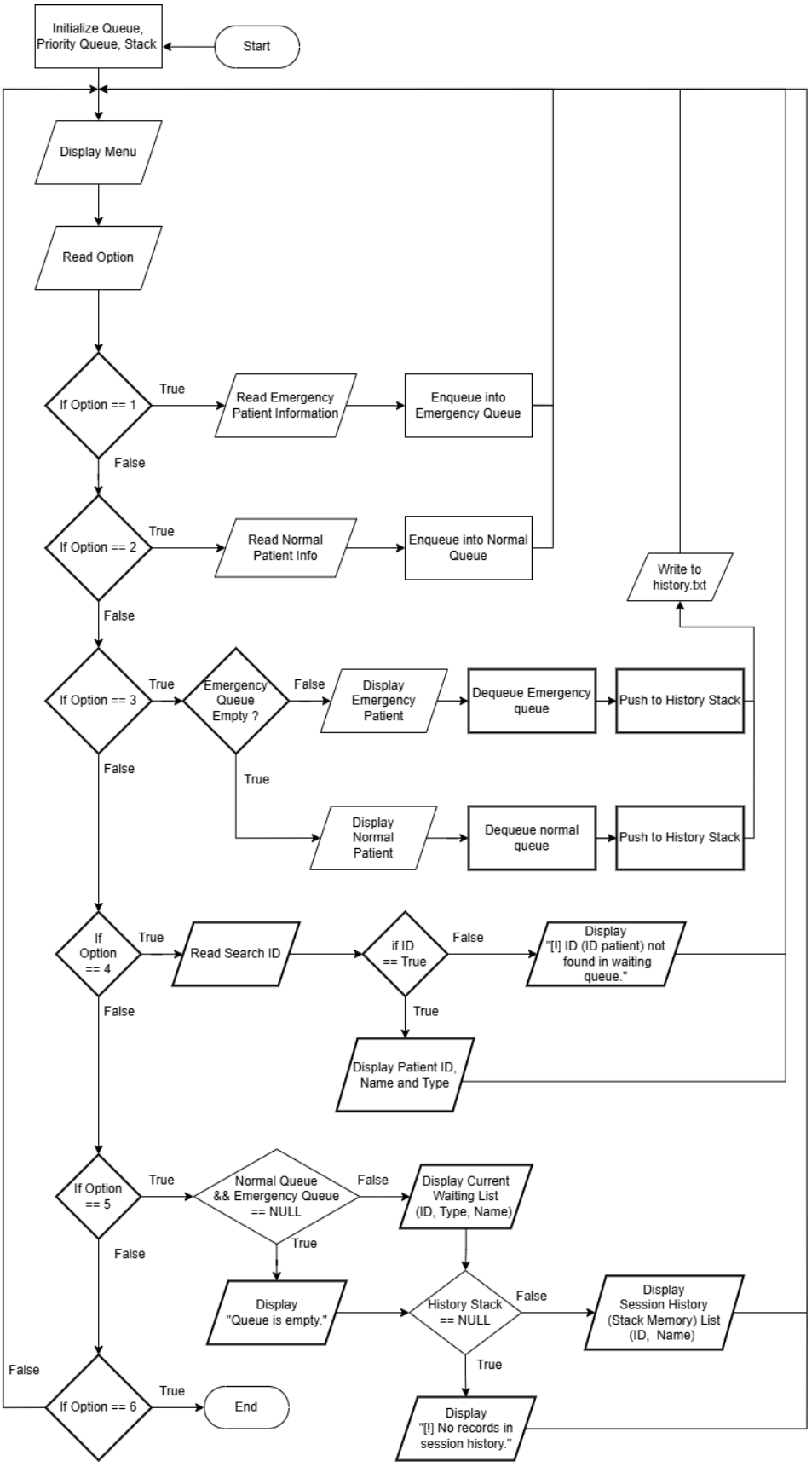

Figure 2.3 Flowchart

PART 3: SYSTEM PROTOTYPE

Menu Section

```
=== Welcome to PKU Hospital Queue Management System ===

==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection:
```

Case 1 : Register Patients

The staff can add their patients by choosing whether option 1 (Emergency case) or option 2 (Normal case). They must register the patient by adding their Name. Also, the ID for that new patient will be automatically assigned by a unique ID starting from ID: 0000 and increasing by one. For example, the second patient will be automatically assigned by ID: 0001 and so on.

Patient 1 (Normal)

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 2

Patient ID: 000
Enter Patient Name: Abdurrafiq Bin Zakaria
```

Patient 2 (Emergency)

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 1

Patient ID: 001
Enter Patient Name: Daniel Iman Haqimie Bin Yusuff
```

Patient 3 (Normal)

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 2

Patient ID: 002
Enter Patient Name: Ahmad Munif Bin Baharum
```

Patient 4 (Emergency)

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 1

Patient ID: 003
Enter Patient Name: Najmuddin Bin Kamarudin
```

Case 2 : Option 4 (Search Patient by ID)

The staff can also search the ID of patients that still in the queue and have not been called yet. They can search them by inserting their ID, and the program will show their ID, Name,a nd type of triage level if that ID exists in the program, and it is otherwise if the ID is not found in the program (queue).

Found

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 4
Enter ID: 002

[Found] ID: 002 | Name: Ahmad Munif Bin Baharum | Type: Normal
```

Not Found

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 4
Enter ID: 006

[!] ID 006 not found in waiting queue.
```

Case 3 : Option 3 (Call Next Patient)

When the patient has been called, the data in the queue of that patient will be dequeued (erased/removed), and it will be pushed to the history stack to be written in the file “history.txt”. Also, the program will automatically call the patient with the highest priority first, rather than follow the sequence.

```
==================================
NOW SERVING: [001] Daniel Iman Haqimie Bin Yusuff
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 3
[System] Record for Daniel Iman Haqimie Bin Yusuff archived to history.txt

==================================
NOW SERVING: [003] Najmuddin Bin Kamarudin
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection:
```

```
==================================
NOW SERVING: [003] Najmuddin Bin Kamarudin
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 3
[System] Record for Najmuddin Bin Kamarudin archived to history.txt

==================================
NOW SERVING: [000] Abdurrafiq Bin Zakaria
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection:
```

```
==================================
NOW SERVING: [000] Abdurrafiq Bin Zakaria
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 3
[System] Record for Abdurrafiq Bin Zakaria archived to history.txt

==================================
NOW SERVING: [002] Ahmad Munif Bin Baharum
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection:
```

```
==================================
NOW SERVING: [002] Ahmad Munif Bin Baharum
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 3
[System] Record for Ahmad Munif Bin Baharum archived to history.txt

[!] No more patients in queue.

==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection:
```

Case 4 : Option 5 (View Queue & History)

This option 5, you will display all the remaining queues and the current past event (history). If the queue has been called. That queue data will be pushed to the history stack, and the data in queue now have been deleted.

Before using Option 3

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 5

--- CURRENT WAITING LIST ---
ID: 001 | [EMERGENCY] | NAME: Daniel Iman Haqimie Bin Yusuff
ID: 003 | [EMERGENCY] | NAME: Najmuddin Bin Kamarudin
ID: 000 | [NORMAL]    | NAME: Abdurrafiq Bin Zakaria
ID: 002 | [NORMAL]    | NAME: Ahmad Munif Bin Baharum

[!] No records in session history.
```

After using Option 3

```
==================================
NOW SERVING: None
==================================
1. Register Emergency Patient
2. Register Normal Patient
3. Call Next Patient
4. Search Patient by ID
5. View Queue & History
6. Exit
Selection: 5

Queue is empty.

--- SESSION HISTORY (Stack Memory) ---
ID: 002 | Ahmad Munif Bin Baharum
ID: 000 | Abdurrafiq Bin Zakaria
ID: 003 | Najmuddin Bin Kamarudin
ID: 001 | Daniel Iman Haqimie Bin Yusuff
```

Case 5 : All the history stored in history.txt

All the enqued data from “Call Next Patient.” will be stored in this history.txt file. If they have new data to be updated, this file will not be erased because we use ios::app. So, the staff can review all the patients on that day without losing any patients when they close the program.

history.txt

```
1   ID: 001 | Type: EMERGENCY | Name: Daniel Iman Haqimie Bin Yusuff
2   ID: 003 | Type: EMERGENCY | Name: Najmuddin Bin Kamarudin
3   ID: 000 | Type: NORMAL    | Name: Abdurrafiq Bin Zakaria
4   ID: 002 | Type: NORMAL    | Name: Ahmad Munif Bin Baharum
5
```

PART 4: DEVELOPMENT ACTIVITIES

Meeting Date	Members Participate in the meeting	Activity	Task for each member	Task Achieved (Yes/No)
31/12/2025	ALL	Project briefing & idea discussion	Najmuddin: Draft project synopsis & objectives Rafiq: Identify PKU problems and system scope Munif: Propose data structures (Queue, Stack, Priority Queue)	Yes
5/1/2026	ALL	System requirements analysis	Najmuddin: Write system requirements Rafiq: Use case identification & descriptions Munif: Actor definition and task mapping	Yes
8/1/2026	ALL	System design discussion	Najmuddin: Class diagram design Rafiq: Flowchart design Munif: Validate system logic and data flow	Yes
12/1/2026	ALL	Prototype development	Najmuddin: Implement Queue & Linked List in C++	Yes

			Rafiq: Implement Priority Queue (Emergency cases) Munif: Implement Stack & history log feature	
15/1/2026	ALL	Final review & report preparation	Najmuddin: Integrate code & testing Rafiq: Documentation & diagram polishing Munif: Final system explanation & report review	Yes

PART 5: APPENDIX

```
//Source Code
#include <iostream>
#include <string>
#include <fstream>
#include <stdio.h>

using namespace std;

// Global counter for automatic ID (000, 001...)
int patientCounter = 0;

string generateID() {
    char buffer[10];
    sprintf(buffer, "%03d", patientCounter++);
    return string(buffer);
}

// --- DATA STRUCTURE: PATIENT ---
class Patient {
public:
    string id, name;
    bool isEmergency;
    Patient(string i, string n, bool e) : id(i), name(n), isEmergency(e) {}
};

// --- CHAPTER 9: NODE ---
struct nodeQ {
    Patient* data;
    nodeQ* next;
    nodeQ(Patient* p) : data(p), next(nullptr) {}
};

// --- CHAPTER 8: HISTORY STACK (With Automatic File Saving) ---
class HistoryStack {
private:
    nodeQ* topPtr;
public:
    HistoryStack() : topPtr(nullptr) {}

    void pushAndSave(Patient* p) {
        if (p == nullptr) return;

        nodeQ* newNode = new nodeQ(p);
        newNode->next = topPtr;
```

```
        topPtr = newNode;

        ofstream outFile("history.txt", ios::app);
        if (outFile.is_open()) {
            string typeLabel = (p->isEmergency) ? "EMERGENCY" : "NORMAL   ";
            outFile << "ID: " << p->id << " | Type: " << typeLabel << " | Name: " << p->name << endl;
            outFile.close();
        }
        cout << "[System] Record for " << p->name << " archived to history.txt\n";
    }

    void displayStack() {
        if (!topPtr) { cout << "\n[!] No records in session history.\n"; return; }
        cout << "\n--- SESSION HISTORY (Stack Memory) ---\n";
        nodeQ* temp = topPtr;
        while (temp) {
            cout << "ID: " << temp->data->id << " | " << temp->data->name << endl;
            temp = temp->next;
        }
    }
};

// --- CHAPTER 9: PRIORITY QUEUE ---
class PriorityQueue {
private:
    nodeQ *frontPtr;
public:
    PriorityQueue() : frontPtr(nullptr) {}

    void enqueue(Patient* p) {
        nodeQ* newNode = new nodeQ(p);
        if (p->isEmergency) {
            if (!frontPtr || !frontPtr->data->isEmergency) {
                newNode->next = frontPtr;
                frontPtr = newNode;
            } else {
                nodeQ* temp = frontPtr;
                while (temp->next != nullptr && temp->next->data->isEmergency)
                    temp = temp->next;
                newNode->next = temp->next;
                temp->next = newNode;
            }
        } else {
            if (!frontPtr) frontPtr = newNode;
            else {
                nodeQ* temp = frontPtr;
                while (temp->next != nullptr) temp = temp->next;
                temp->next = newNode;
```

```
        }
      }
    }

    Patient* dequeue() {
      if (!frontPtr) return nullptr;
      nodeQ* temp = frontPtr;
      Patient* p = temp->data;
      frontPtr = frontPtr->next;
      delete temp;
      return p;
    }

    void searchByID(string searchID) {
      nodeQ* temp = frontPtr;
      bool found = false;
      while (temp != nullptr) {
        if (temp->data->id == searchID) {
          cout << "\n[Found] ID: " << temp->data->id << " | Name: " << temp->data->name;
          cout << " | Type: " << (temp->data->isEmergency ? "Emergency" : "Normal") << endl;
          found = true;
          break;
        }
        temp = temp->next;
      }
      if (!found) cout << "\n[!] ID " << searchID << " not found in waiting queue.\n";
    }

    void displayQueue() {
      if (!frontPtr) { cout << "\nQueue is empty.\n"; return; }
      cout << "\n--- CURRENT WAITING LIST ---\n";
      nodeQ* temp = frontPtr;
      while (temp) {
        string type = (temp->data->isEmergency) ? "[EMERGENCY]" : "[NORMAL]   ";
          cout << "ID: " << temp->data->id << " | " << type << " | NAME: " << temp->data->name <<
endl;
        temp = temp->next;
      }
    }
};

// --- MAIN MENU ---
int main() {
    PriorityQueue pkuQueue;
    HistoryStack history;
    Patient* currentServing = nullptr;
    int choice;
```

```
    cout << "=== Welcome to PKU Hospital Queue Management System ===\n";

    while (true) {
        cout << "\n====================================";
            if (currentServing) cout << "\nNOW SERVING: [" << currentServing->id << "] " <<
currentServing->name;
        else cout << "\nNOW SERVING: None";
        cout << "\n====================================";
        cout << "\n1. Register Emergency Patient";
        cout << "\n2. Register Normal Patient";
        cout << "\n3. Call Next Patient";
        cout << "\n4. Search Patient by ID";
        cout << "\n5. View Queue & History";
        cout << "\n6. Exit";
        cout << "\nSelection: ";

        if (!(cin >> choice)) { cin.clear(); cin.ignore(100, '\n'); continue; }
        if (choice == 6) break;

        string name, sid, assignedID;
        switch (choice) {
            case 1: // Emergency
                assignedID = generateID();
                cout << "\nPatient ID: " << assignedID << endl;
                cout << "Enter Patient Name: ";
                cin.ignore(); getline(cin, name);
                pkuQueue.enqueue(new Patient(assignedID, name, true));
                break;
            case 2: // Normal
                assignedID = generateID();
                cout << "\nPatient ID: " << assignedID << endl;
                cout << "Enter Patient Name: ";
                cin.ignore(); getline(cin, name);
                pkuQueue.enqueue(new Patient(assignedID, name, false));
                break;
            case 3:
                if (currentServing != nullptr) {
                    history.pushAndSave(currentServing);
                }
                currentServing = pkuQueue.dequeue();
                if (!currentServing) cout << "\n[!] No more patients in queue.\n";
                break;
            case 4:
                cout << "Enter ID: "; cin >> sid;
                pkuQueue.searchByID(sid);
                break;
            case 5:
                pkuQueue.displayQueue();
```

```
            history.displayStack();
            break;
        }
    }
    cin.ignore();
    return 0;
}
```