PRISON MANAGEMENT SYSTEM

Name	Roll no
Muhammad Hashim Abbasi	242187
Abdul Rahman	242179
Abu Bakar Awan	242233

Submitted to: Ma’am Aisha Sattar Choudhry

Air University, Islamabad

21 December 2025

ABSTRACT

The Prison Management System is a computerized solution designed to streamline the management of prison operations. It replaces traditional manual record-keeping with an organized and efficient digital system, helping staff manage prisoners, officers, and cells in a secure and structured manner. The system maintains comprehensive prisoner records, including personal information, crime details, cells assign, and supervising officers. Officers' duties and cell capacities are tracked to ensure smooth daily operations, while a visitor's queue is managed to regulate prison visits efficiently. By using data structures such as linked lists, stacks, and queues, the system allows dynamic addition, deletion, searching, and update records. File handling ensures all data is stored persistently, reducing the chances of loss or errors. Overall, the system enhances accuracy, security, and efficiency in prison management, minimizes paperwork, and supports informed decision-making by prison authorities.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Introduction

Prison management is a critical task that requires accurate record-keeping and efficient coordination of multiple entities such as prisoners, officers, cells, and visitors. Manual management is prone to errors, delays, and miscommunication, which can lead to serious operational issues. To address these challenges, a **Prison Management System (PMS)** has been developed using **C++**.

This system provides a **computerized solution** for managing prison operations, including prisoner registration, officer management, cell allocation, and visitor scheduling. It leverages **data structures such as linked lists, stacks, and queues** to ensure efficient data handling. The system also maintains **persistent storage** using text files, allowing administrators to save and retrieve data reliably.

Key features of the system include:

- **Prisoner Management:** Add, edit, delete, search, and display prisoner records while automatically managing unique IDs.
- **Officer Management:** Maintain a list of officers along with their rank and assigned prisoners.
- **Cell Management:** Allocate prisoners to cells and monitor occupancy to prevent overfilling.
- **Visitor Queue Management:** Schedule and process visitors in a **first-in-first-out (FIFO)** manner.
- **User Interface:** A simple and intuitive console-based UI with colored output for better readability.
- **Data Integrity:** Validation checks for IDs, cell capacity, and officer existence ensure consistent data.

Overall, the system is designed to **automate administrative tasks**, reduce human error, and provide a clear and organized view of all prison operations.

1.2 Problem Statement

Managing a prison is a complex and sensitive task that involves keeping track of multiple entities, including prisoners, officers, cells, and visitors. Traditional manual methods for handling these operations are time-consuming, error-prone, and inefficient. Common problems in manual prison management include:

- **Tracking Prisoners:** Maintaining accurate records of prisoner details, crimes, sentences, cell assignments, and responsible officers is challenging without an organized system. Mistakes can lead to overcrowding or mismanagement.
- **Officer Management:** Assigning prisoners to officers and keeping track of officer responsibilities requires careful monitoring. Manual methods can lead to confusion or uneven workload distribution.
- **Cell Allocation:** Cells have limited capacities, and improper allocation can result in overcrowding or underutilization. Ensuring that prisoners are assigned only to available cells is difficult without automated checks.
- **Visitor Management:** Visitors must be scheduled and processed in a **first-come, first-served** manner. Manual queues are prone to errors, delays, and mismanagement.
- **Data Integrity and Storage:** Maintaining historical records manually is inefficient. Losing or misplacing files can result in permanent loss of critical information.
- **ID Management:** Prisoners need unique IDs for identification. Without a proper system, deleted IDs may be wasted, leading to inefficiency in future assignments.

The **objective of this project** is to develop a **computerized Prison Management System** that automates these processes. The system ensures accurate record-keeping, efficient allocation of prisoners to cells and officers, orderly visitor management, and secure storage of all data in files. Additionally, the system recycles prisoner IDs and validates all input to prevent errors, making prison administration **more reliable, organized, and efficient**.

1.3 Objectives

The main objective of the **Prison Management System** is to provide a **computerized and efficient solution** for managing all aspects of prison operations. The system is designed to address the challenges of manual management and improve accuracy, organization, and efficiency.

1. **Efficient Prisoner Management:**
    - Maintain detailed records of prisoners, including their personal information, crimes, sentences, assigned cells, and officers.
    - Allow administrators to add, edit, delete, search, and display prisoner information with ease.
    - Automatically manage unique prisoner IDs and recycle deleted IDs to optimize ID usage.
2. **Officer Management:**
    - Maintain a structured record of officers along with their ranks.
    - Assign prisoners to officers and display officer information to monitor responsibilities.
3. **Cell Management:**
    - Manage cells with their capacity and current occupancy.
    - Ensure that prisoners are assigned only to available cells, preventing overcrowding.
    - Update cell occupancy dynamically when prisoners are added or removed.
4. **Visitor Queue Management:**
    - Handle visitors in a **first-in-first-out (FIFO)** order using a queue.
    - Add, process, and display visitors efficiently, linking them to the respective prisoners.
5. **Data Persistence:**
    - Store all prisoners, officers, and cell data in text files to ensure information is **saved permanently**.
    - Load and display saved data from files whenever needed.
6. **User-Friendly Interface:**
    - Provide a console-based interface with color-coded messages for better readability.
    - Simplify operations with intuitive menus and input prompts for administrators.
7. **Data Validation and Integrity:**
    - Validate all inputs, including IDs, cell capacity, and officer existence, to prevent errors.
    - Ensure that all operations maintain consistent and accurate data across the system.

Overall, the system aims to **automate administrative tasks**, reduce human error, and provide an organized framework for **prison operations management**.

1.4 Problem Solution

The **Prison Management System** provides a computerized solution to the challenges of manual prison administration. The system integrates multiple modules to efficiently manage prisoners, officers, cells, and visitors while ensuring data accuracy and integrity. The solution can be described as follows:

1. **Automated Prisoner Management:**
   - The system allows administrators to **add, edit, delete, search, and display** prisoner records.
   - Each prisoner is assigned a **unique ID**, and deleted IDs are **recycled using a stack**, ensuring optimal use of identifiers.
   - Prisoners are assigned to cells and officers, with validation checks to prevent assignment errors.
2. **Officer Management:**
   - Administrators can **add and view officers** along with their ranks.
   - The system ensures that prisoners are assigned only to **existing officers**, preventing mismatches.
3. **Cell Management:**
   - Cells are managed with their **capacity and current occupancy**.
   - The system prevents **overcrowding** by checking cell availability before assigning prisoners.
   - Occupancy is updated automatically when prisoners are added or removed.
4. **Visitor Queue Management:**
   - Visitors are managed in a **first-in-first-out (FIFO) queue**, ensuring orderly processing.
   - The system allows adding visitors, processing them in order, and displaying the queue.
   - Each visitor is linked to a specific prisoner, maintaining accurate records of visits.
5. **Persistent Data Storage:**
   - All data for prisoners, officers, and cells is **saved in text files**, enabling permanent record keeping.
   - Administrators can **load and view file content** directly from the system.

6. **User-Friendly Interface:**
    - The system uses a **console-based UI with color-coded messages** to enhance clarity.
    - Menus are intuitive, allowing administrators to navigate easily between modules.
7. **Data Validation and Integrity:**
    - Input validation ensures that only valid IDs, cell numbers, and officer IDs are used.
    - The system prevents errors such as assigning prisoners to full cells or non-existent officers.

1.5 Methodology

The **Prison Management System** is implemented using **C++** with a focus on **linked lists, stack, queue, and file handling**. The system is structured into modules to manage prisoners, officers, cells, and visitors. The methodology describes the **design, data structures, and workflow** used to solve the problems identified in prison administration.

1.5.1 System Design

The system follows a **modular design**, dividing functionality into distinct components:

- **Prisoner Management Module**
- **Officer Management Module**
- **Cell Management Module**
- **Visitor Queue Module**
- **File Handling Module**
- **User Interface Module**

This separation allows for **easy maintenance and scalability**, as each module handles a specific set of operations.

1.5.2. Data Structures Used

- **Linked Lists:**
  Used to store **dynamic data** such as prisoners, officers, and cells. Each entity has a **node structure** with relevant attributes and a pointer to the next node. Enables **efficient addition, deletion, and traversal** of records without predefining the size.
- **Stack (ID Stack):**

Used to **recycle prisoner IDs**. When a prisoner is deleted, their ID is pushed to the stack. When a new prisoner is added, the system first checks the stack to reuse an ID before generating a new one.

- **Queue (Visitor Queue):**

  Implements **FIFO** for visitor management. Visitors are enqueued as they arrive and dequeued when processed, maintaining proper visiting order.

- **Files:**

  prisoners.txt, officers.txt, and cells.txt are used for **persistent storage**. Save Data() appends new records to the respective file. Load Data() reads and displays saved records when required.

1.5.3. Module Workflow

- **Login System:**

  Provides a secure login using a **username and password**. Password input is masked with stars (*) using _getch().

- **Prisoner Management:**

  Add: Validates cell availability and officer existence, assigns ID, updates cell occupancy, and saves data to file. **Edit:** Allows modification of prisoner details. **Delete:** Removes prisoner, decrements cell occupancy, and recycles ID. **Search & Display:** Retrieves prisoner information, including assigned officer and cell details.

- **Officer Management:**

  Add Officer: Checks for unique ID and saves officer details to file. **Display Officers:** Lists all officers with their ranks.

- **Cell Management:**

  Add Cell: Creates new cells with specific capacities. **Display Cells:** Shows cell occupancy and capacity to monitor availability.

- **Visitor Queue Management:**

  Add Visitor: Enqueues visitor to the queue with linked prisoner ID. **Process Visitor:** Dequeues the first visitor for processing. **Display Queue:** Shows all pending visitors in order.

- **File Viewing:**

  Administrators can view the **contents of saved files** for prisoners, officers, and cells.

1.5.4. User Interface

The system uses a **console-based interface** with **ANSI color codes** for clear distinction of messages: success (green), error (red), and information (cyan). Menus are **centered manually** for better readability. Input validation ensures **robustness against incorrect or invalid data**.

1.5.5 Data Flow

1. User logs in → system validates credentials.
2. Administrator selects a module (Prisoners, Officers, Cells, Visitors, or File Viewer).
3. Module operations (Add/Edit/Delete/Search/Display) are performed using linked lists.
4. For prisoners, **cell and officer checks** ensure valid assignment.
5. For visitors, **queue operations** maintain order.
6. All additions are saved to files for persistence.
7. Deleted prisoner IDs are **pushed to the stack**, ready for reuse

1.5.5. Integration

Prisoner List interacts with **Cell List** and **Officer List** to validate assignments. **Visitor Queue** is linked to prisoner IDs for proper tracking. File handling ensures that **all operations are persistent**, even after program termination.

In conclusion, the methodology combines **object-oriented programming, dynamic data structures, file handling, and a user-friendly interface** to automate prison management. The system ensures **accuracy, efficiency, and reliability** in handling all administrative operations.

Figure 1.1 Flow Chart:

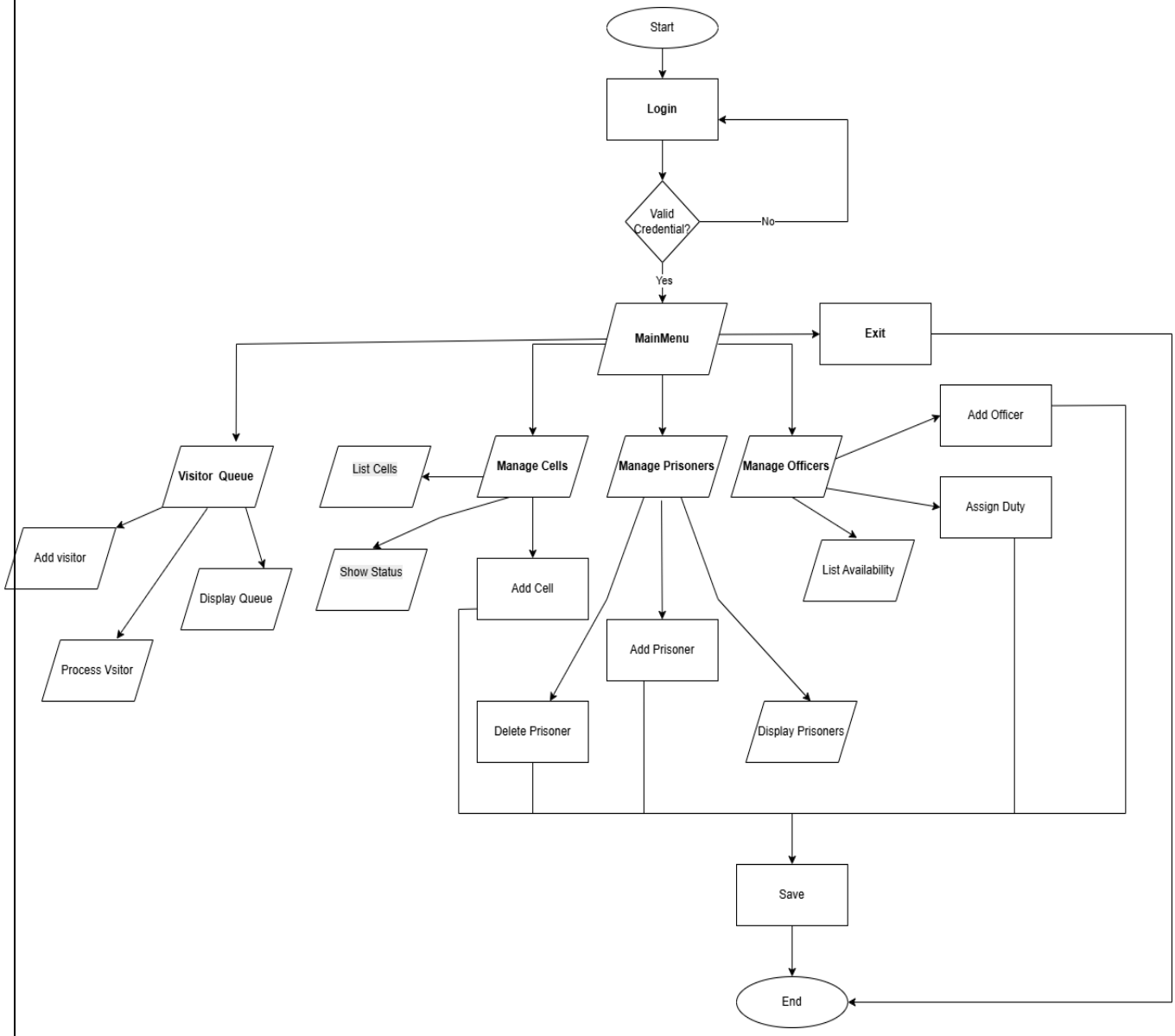


Figure 1.2 ERD

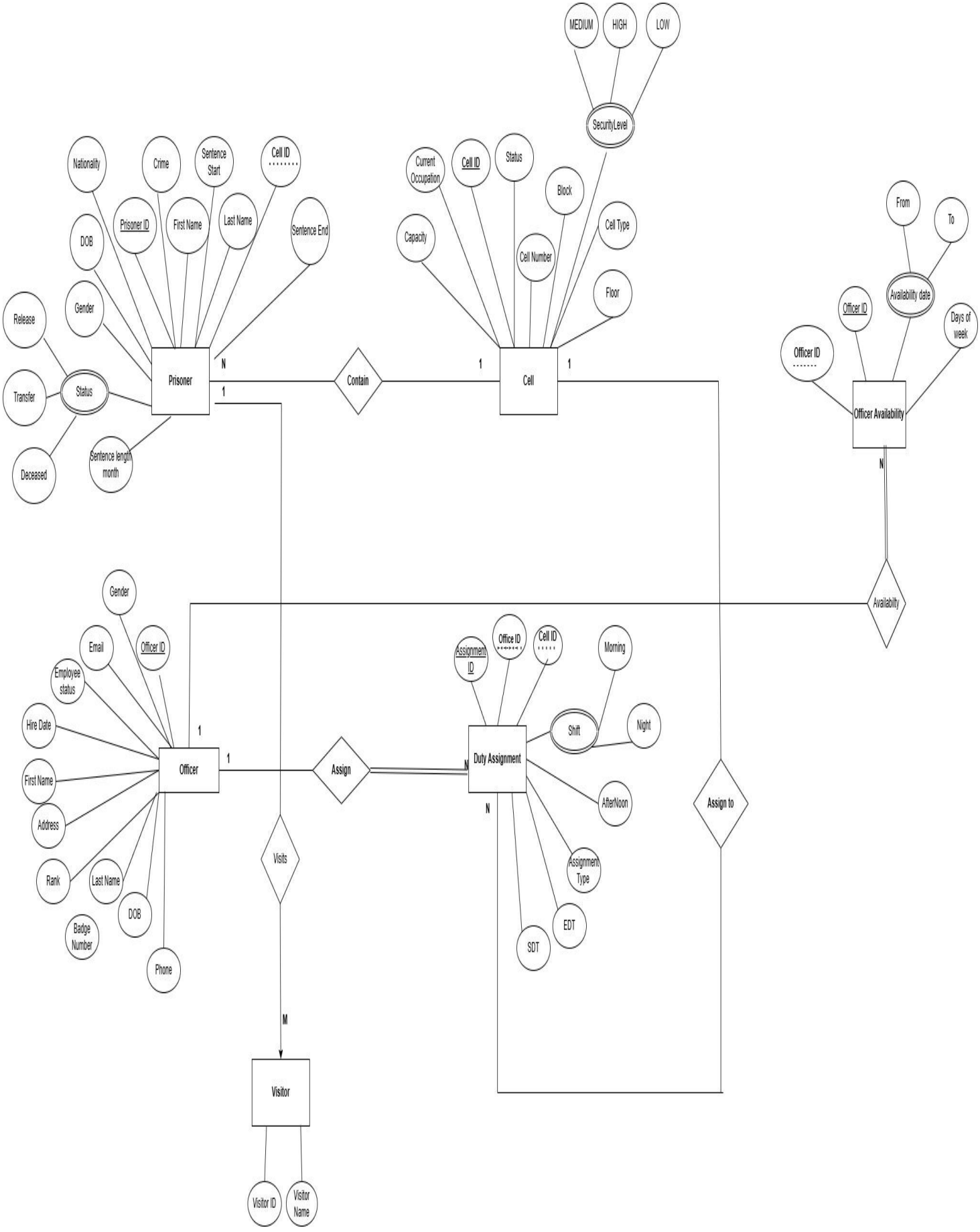
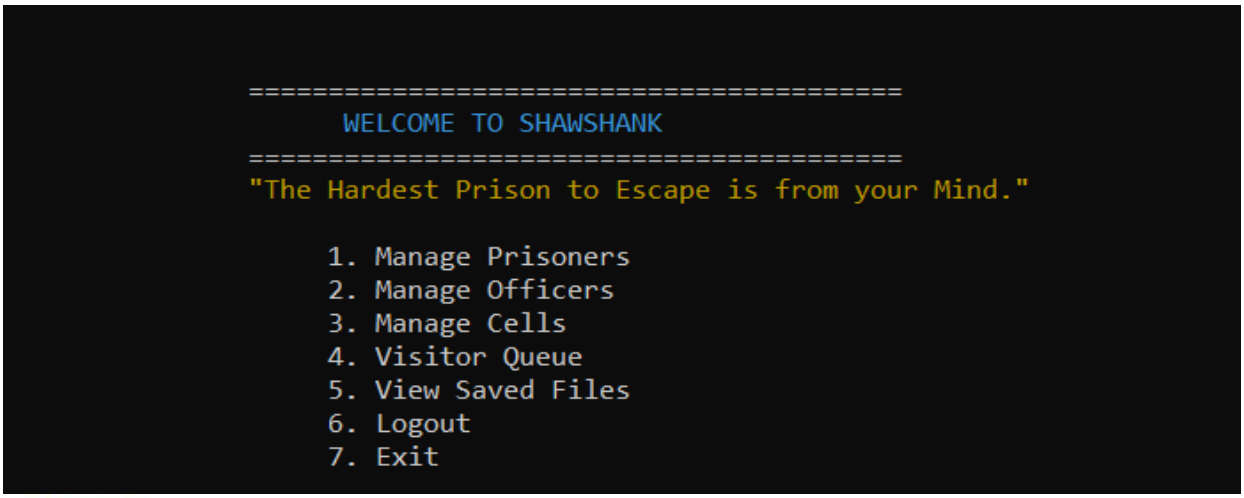

MEDIUM
HIGH
LOW
SecurityLevel
Nationality
Crime
Sentence Start
Cell ID
Prisoner ID
First Name
Last Name
Sentence End
DOB
Gender
Release
Transfer
Status
Deceased
Sentence length month
Prisoner
N
1
Contain
1
1
Cell
Current Occupation
Cell ID
Status
Block
Capacity
Cell Number
Cell Type
Floor
From
To
Availability date
Officer ID
Days of week
Officer ID
Officer Availability
N
Availability
Gender
Email
Officer ID
Employee status
Hire Date
First Name
Address
Rank
Last Name
DOB
Badge Number
Phone
1
Officer
1
Assign
N
Duty Assignment
N
Assignment ID
Office ID
Cell ID
Morning
Shift
Night
AfterNoon
Assignment Type
EDT
SDT
Assign to
Visits
M
Visitor
Visitor ID
Visitor Name

FIGURE 1.3 CODE & OUTPUT SNIPPETS




```
void menu() {
    while (true) {
        system("cls");
        printWelcome();

        cout << "                              1. Manage Prisoners" << endl;
        cout << "                              2. Manage Officers" << endl;
        cout << "                              3. Manage Cells" << endl;
        cout << "                              4. Visitor Queue" << endl;
        cout << "                              5. View Saved Files" << endl;
        cout << "                              6. Logout" << endl;
        cout << "                              7. Exit" << endl;
        cout << "\n";
        cout << "                              Enter choice: ";

        int choice;
        if (!(cin >> choice)) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "\n";
            printError("Invalid input! Please enter a number.");
            system("timeout /t 1 >nul");
            continue;
        }

        switch (choice) {
        case 1: prisonerMenu(); break;
        case 2: officerMenu(); break;
        case 3: cellMenu(); break;
        case 4: visitorMenu(); break;
        case 5: viewFilesMenu(); break;
        case 6: return; // Logout
        case 7: exit(0); // Exit
        default: cout << "\n"; printError("Invalid choice!"); system("timeout /t 1 >nul");
        }
    }
}
```

```
void cellMenu() {
    while (true) {
        system("cls");
        printHeader("MANAGE CELLS");
        cout << "                                    1. Add Cell" << endl;
        cout << "                                    2. Display All" << endl;
        cout << "                                    3. Back" << endl;
        cout << "\n";
        cout << "                                    Choice: ";

        int choice;
        if (!(cin >> choice)) {
            cin.clear(); cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }

        if (choice == 3) return;

        if (choice == 1) {
            int id, cap;
            cout << "\n";
            cout << "                                        ID: "; cin >> id;
            cout << "                                        Capacity: "; cin >> cap;
            cells.addCell(id, cap);
        }
        else if (choice == 2) {
            cells.display();
        }
        system("pause");
    }
}
```

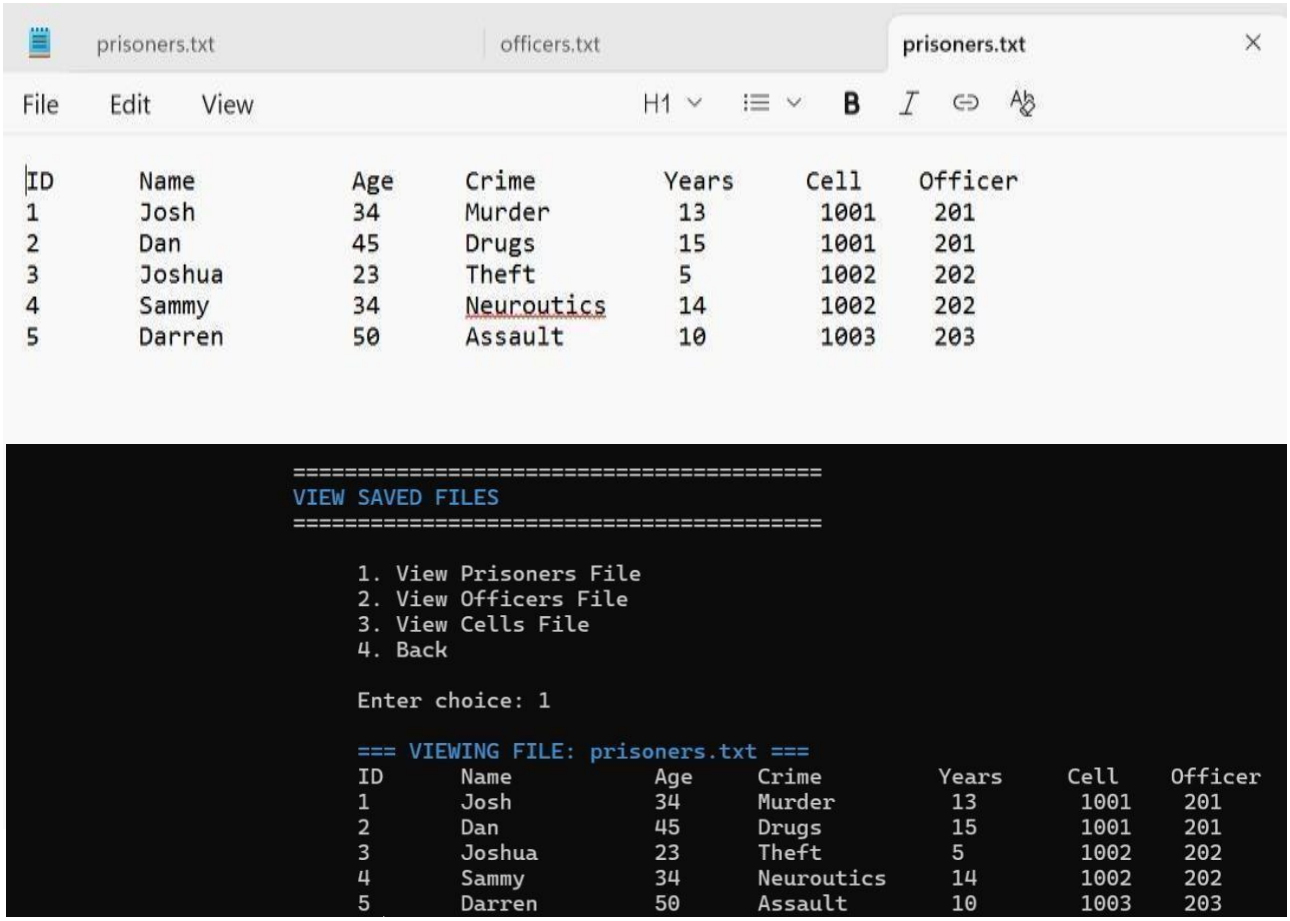
```
      ==========================================
      MANAGE CELLS
      ==========================================

            1. Add Cell
            2. Display All
            3. Back

            Choice: 1

            ID: 1234
            Capacity: 3
            [SUCCESS] Cell added successfully.
            [SUCCESS] Data saved to file!
inue
```

FIGURE 1.4 File Handling



CONCLUSION:

The **Prison Management System** efficiently automates prison operations, including prisoner, officer, cell, and visitor management. Using **linked lists, stacks, queues, and file handling**, the system ensures accurate record-keeping, proper ID management, and orderly visitor scheduling. Its **user-friendly interface** and input validation reduce errors and simplify administration. Overall, the system provides a **reliable, organized, and efficient solution** for modern prison management.