EXTRENAL EXPERT INPUT – DATA STRUCTURE (CSES003)

CENTRE FOR PROFESSIONAL ENHANCEMENT, LPU

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology

Computer Science Engineering

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

Under the guidance of

Mr. Sudeep Chowhan

On the Project Titled

PHARMACY MANAGEMENT SYSTEM

Submitted By

Name: SHIV NARAYAN DWIVEDI

Registration Number: 12314985

Signature: [illegible]

Academic Session: 2025 – 2026

TO WHOM SO EVER IT MAY CONCERN

I, Shiv Narayan Dwivedi, with Registration No. 12314985, hereby declare that the work done by me on “Pharmacy Management System” is a record of original work done by me under the guidance of faculty for the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering.

Name:	Shiv Narayan Dwivedi (12314985)
Signature:	शिवनारायण
Dated:	23/02/2026 .

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Lovely Professional University for providing me the opportunity to work on this project titled “Pharmacy Management System”.

I would like to thank my faculty members for their valuable guidance support and encouragement throughout the development of this project. Their suggestions helped me understand the concepts clearly and complete the project successfully.

I am also thankful to my friends and family for their continuous motivation and support during the completion of this project. This project has helped me gain practical knowledge and improve my understanding of Data Structures and their real-world applications.

Name: Shiv Narayan Dwivedi

Registration Number: 12314985

TABLE OF CONTENTS

INTRODUCTION

The Pharmacy Management System is a software application designed to manage and maintain medicine records efficiently in a digital format. In traditional pharmacies, inventory is often maintained manually using registers or spreadsheets, which can lead to errors such as incorrect stock calculations, duplicate entries, and difficulty in tracking expiry dates.

With the advancement of technology, it has become essential to adopt computerized systems to improve efficiency and accuracy. The Pharmacy Management System provides a simple and effective solution for managing medicine records digitally. It allows users to add new medicines, update existing records, search medicines, and generate bills after purchase.

This project is developed using C++ programming language and demonstrates the practical implementation of fundamental data structures. Linked List is used to store medicine records dynamically, while Queue is used to maintain purchase history in sequential order. These data structures help in managing data efficiently without requiring fixed memory size.

The system is designed to be user-friendly and menu-driven so that users can easily perform operations without requiring technical knowledge. The main goal of this project is to simplify pharmacy operations, reduce manual errors, and improve record management efficiency.

OBJECTIVES

The primary objective of the Pharmacy Management System is to develop a software solution that can efficiently manage medicine inventory using data structures.

The specific objectives of this project are:

- To design a system that maintains medicine records digitally
- To demonstrate the practical application of data structures in real-world scenarios
- To reduce manual record keeping errors
- To provide an efficient way to add, search, and update medicine records
- To generate bills automatically after purchase
- To track stock levels and provide alerts for low stock medicines
- To monitor expiry dates of medicines
- To improve efficiency and accuracy in pharmacy operations

- To create a simple and user-friendly interface

PROBLEM STATEMENT

Managing medicine inventory manually is inefficient and prone to errors. Pharmacists often face challenges such as maintaining accurate stock records, monitoring expiry dates, and avoiding duplicate entries.

Manual systems require more time and effort, and there is a higher chance of mistakes that can lead to medicine shortages or wastage. Incorrect stock tracking can also affect customer service and pharmacy operations.

The aim of this project is to develop a computerized system that provides an automated solution for managing medicine records. The system ensures accurate data storage, easy retrieval of information, and efficient inventory management.

SCOPE OF PROJECT

The scope of this project includes managing medicine inventory, tracking stock levels, generating bills, and monitoring expiry dates using a menu-driven interface.

The system is suitable for small pharmacies or as a learning tool for students studying data structures. It demonstrates how programming concepts can be used to build real-world applications.

Although the current system is designed as a standalone application, it can be extended in the future by integrating database support, graphical user interface, and multi-user functionality.

FEATURES

The Pharmacy Management System provides several features that help in managing medicine inventory efficiently. Each feature is designed to perform a specific task and improve the overall functionality of the system.

Add Medicine

The Add Medicine feature allows the user to enter details of new medicines into the system. The user provides information such as medicine ID, name, price, quantity, and expiry date.

The system checks for duplicate entries before adding a new medicine to ensure data accuracy. This feature helps maintain proper records and prevents duplication of data.

Show All Medicines

This feature displays all medicines available in the system along with their complete details. It allows users to view the entire inventory at once and check stock levels.

This feature is useful for verifying records and monitoring available medicines.

Search Medicine

The Search Medicine feature allows users to find a specific medicine by entering its name. The system searches through the stored records and displays relevant details such as price, quantity, and expiry date.

This feature helps users quickly retrieve information without searching manually.

Update Medicine

The Update Medicine feature allows users to modify existing medicine details such as price, quantity, and expiry date. This is useful when new stock arrives or when there is a change in price. Updating records ensures that the system always contains accurate and up-to-date information.

Purchase Medicine

This feature simulates the sale of medicines. When a medicine is purchased, the system reduces the stock quantity and generates a bill displaying medicine name, quantity purchased, price per unit, and total amount.

The purchase history is stored in a queue to maintain transaction order.

Low Stock Alert

The Low Stock Alert feature identifies medicines whose quantity is less than or equal to 5 and displays them as low stock. This helps users restock medicines before they run out.

Expiry Check

This feature allows users to check the expiry date of a medicine by entering its ID. It helps prevent selling expired medicines and ensures safety.

Dashboard

The Dashboard provides summary information such as total number of medicines and total number of purchases. It gives a quick overview of system status.

DATA STRUCTURES USED

Data structures play a crucial role in the development of the Pharmacy Management System. They help in organizing and managing data efficiently so that operations like insertion, deletion, searching, and updating can be performed easily.

In this project, two main data structures are used: Linked List and Queue. These structures are chosen because they are dynamic, efficient, and suitable for real-world applications where data size can change frequently.

Linked List

Linked List is used as the primary data structure to store medicine records in the system. Each medicine is represented as a node that contains fields such as medicine ID, name, price, quantity, and expiry date. Each node in the linked list contains a pointer that connects it to the next node, forming a chain-like structure. This allows the system to store records dynamically without requiring a fixed memory size. The linked list structure makes it easy to add new medicines, update existing records, and remove medicines when necessary. Since nodes are not stored in contiguous memory locations, the system can grow dynamically as new medicines are added.

Linked List is particularly useful in this project because the number of medicines is not fixed, and it allows efficient insertion and deletion operations without shifting other elements. Another advantage of using linked list is that it reduces memory wastage since memory is allocated only when needed. The system traverses the linked list to search for medicines, update stock, and display records.

Queue

Queue is used to store purchase history. When a medicine is purchased, its name is added to the queue. Queue follows FIFO (First In First Out) principle, meaning the first purchase is recorded first. This represents real-world billing flow where transactions occur in sequence. Queue helps maintain order and provides a simple way to track purchases.

SYSTEM DESIGN

The Pharmacy Management System is designed using a menu-driven approach that allows users to interact with the system easily. The system provides a simple interface where users can select different options to perform operations such as adding medicines, searching records, updating stock, and purchasing medicines.

The main goal of the system design is to ensure simplicity, efficiency, and ease of use while demonstrating the practical implementation of data structures. The system is structured in a way that separates user interaction, processing logic, and data storage, making it organized and easy to understand.

Menu Driven Interface

The system follows a menu-driven interface where a list of options is displayed on the screen. The user selects the desired operation by entering the corresponding choice number. This approach makes the system user-friendly and easy to navigate, even for users who are not technically experienced. The menu-driven design ensures smooth interaction between the user and the system.

System Architecture

The system architecture consists of three main components:

1. User Interface

The User Interface is responsible for displaying menu options and receiving input from the user. It acts as a communication layer between the user and the system. Through the interface, users can perform tasks such as adding medicines, searching records, updating details, and generating bills. The interface is designed to be simple and clear so that users can easily understand and operate the system.

2. Processing Unit

The Processing Unit is the core component of the system that performs all logical operations. It processes user input and executes the required tasks such as inserting new records, updating existing records, searching for medicines, and calculating total bill amount. The processing unit ensures that all operations are carried out efficiently and accurately. It also handles validation checks such as preventing duplicate entries and verifying stock availability before purchase.

3. Data Storage

The Data Storage component is responsible for storing medicine records and purchase history. In this system, Linked List is used to store medicine details dynamically, while Queue is used to maintain purchase history. Linked List allows dynamic memory allocation, making it suitable for storing an unknown number of medicine records. Queue helps maintain purchase records in sequential order following FIFO principle.

Design Goals

The system is designed with the following goals in mind:

- To provide a simple and user-friendly interface
- To ensure efficient data management
- To reduce manual errors
- To demonstrate the use of data structures
- To maintain organized and structured data flow

Benefits of the Design

The menu-driven design makes the system easy to use and understand. The separation of components improves clarity and helps maintain organized structure.

The use of linked list and queue ensures efficient data handling while allowing the system to grow dynamically. Overall, the system design provides a clear workflow and supports smooth execution of operations.

MODULES DESCRIPTION

The Pharmacy Management System is divided into several modules, each responsible for performing a specific function. Dividing the system into modules helps improve organization, readability, and maintainability of the system.

Each module works independently but collectively contributes to the overall functionality of the system.

Add Medicine Module

The Add Medicine module allows the user to enter details of new medicines into the system. The user provides information such as medicine ID, name, price, quantity, and expiry date.

Before adding a new medicine, the system checks for duplicate entries to ensure that medicines with the same ID or name are not added again. This helps maintain data accuracy and prevents duplication of records.

This module is essential for maintaining updated inventory records and allows the system to dynamically store medicine details using linked list.

Show All Medicines Module

The Search Medicine module allows users to search for a specific medicine by entering its name. The system traverses the linked list and compares the entered name with stored records.

If the medicine is found, the system displays details such as price, quantity, and expiry date. This module improves accessibility and helps users quickly retrieve information.

Update Medicine Module

The Update Medicine module allows users to modify existing medicine details such as price, quantity, and expiry date. This module ensures that records remain accurate and up-to-date. It is useful when new stock arrives or when price changes occur.

Purchase Medicine Module

The Purchase Medicine module simulates the sale of medicines. The user enters medicine name and quantity, and the system checks stock availability. If sufficient stock is available, the quantity is reduced and a bill is generated showing total price. The purchase history is stored in queue.

Low Stock Alert Module

This module checks medicines whose quantity is less than or equal to 5 and displays them as low stock. It helps users restock medicines on time.

Expiry Check Module

This module allows users to check expiry date of medicines by entering their ID. It helps prevent the sale of expired medicines and ensures safety.

Dashboard Module

The Dashboard module displays summary information such as total number of medicines and total number of purchases. It provides a quick overview of system activity.

WORKING OF SYSTEM

The Pharmacy Management System begins by displaying a menu with various options. The user selects an operation by entering the corresponding choice number. When a medicine is added, its details are stored in the linked list. The linked list allows dynamic storage of records and helps manage inventory efficiently.

When a purchase is made, the system checks stock availability. If sufficient quantity is available, the stock is reduced and a bill is generated showing total amount. The purchase history is stored in queue. The low stock alert feature helps monitor medicines with low quantity so they can be restocked in time. The expiry check feature helps prevent selling expired medicines.

The dashboard provides summary information such as total medicines and purchases, helping users monitor overall system activity. The system continues running until the user selects the exit option.

ADVANTAGES

The Pharmacy Management System provides several advantages that make it useful for managing medicine inventory efficiently. The system helps reduce manual work and improves accuracy in maintaining records. One of the main advantages is that it provides a simple and user-friendly interface, making it easy for users to operate without requiring technical knowledge. The menu-driven design allows users to perform operations easily by selecting options.

The system reduces manual errors that often occur in traditional record-keeping methods. By maintaining records digitally, it ensures accurate stock tracking and prevents duplication of data. Another advantage is efficient inventory management. The system allows users to monitor stock levels and provides alerts for low stock medicines, ensuring timely restocking. The billing feature helps generate bills automatically after purchase, saving time and effort. The project also demonstrates the practical application of data structures, helping students understand how linked list and queue can be used in real-world applications.

FUTURE ENHANCEMENTS

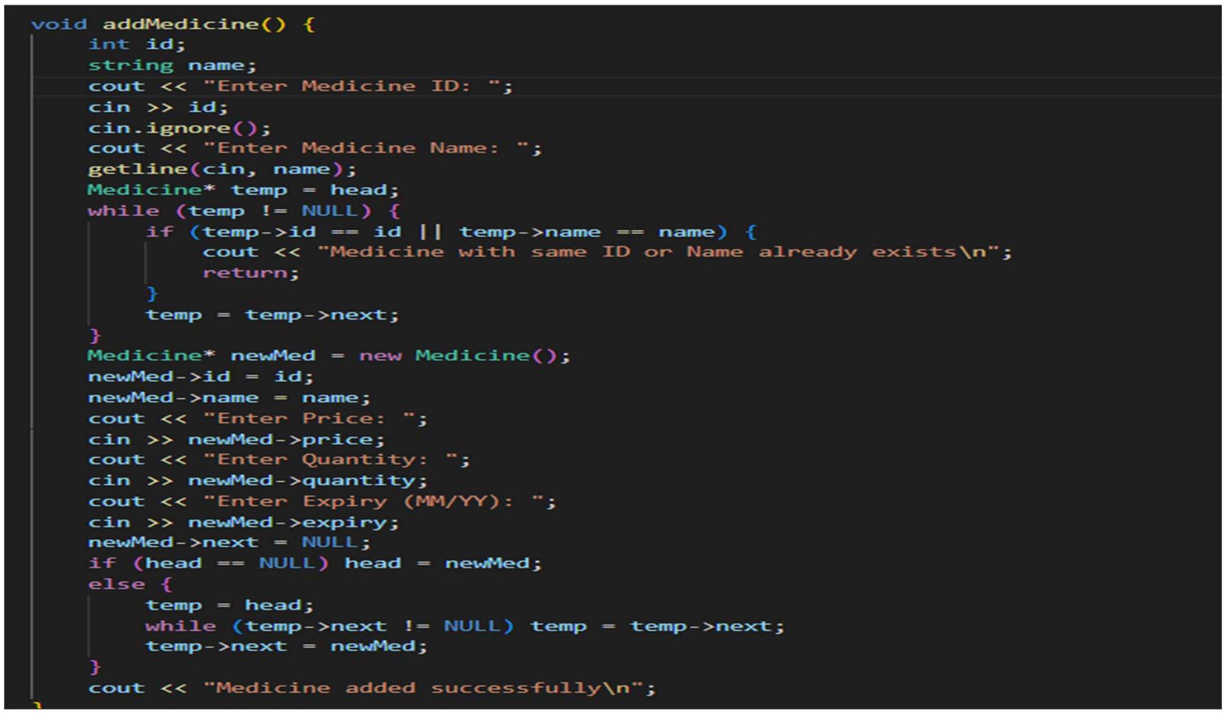
The Pharmacy Management System can be improved further by adding new features and technologies. One possible enhancement is adding file handling or database integration to store data permanently. This will allow records to be saved even after the program is closed. Another improvement could be adding a graphical user interface to make the system more interactive and visually appealing. A login authentication system can be added to improve security and restrict access to authorized users only.

CONCLUSION

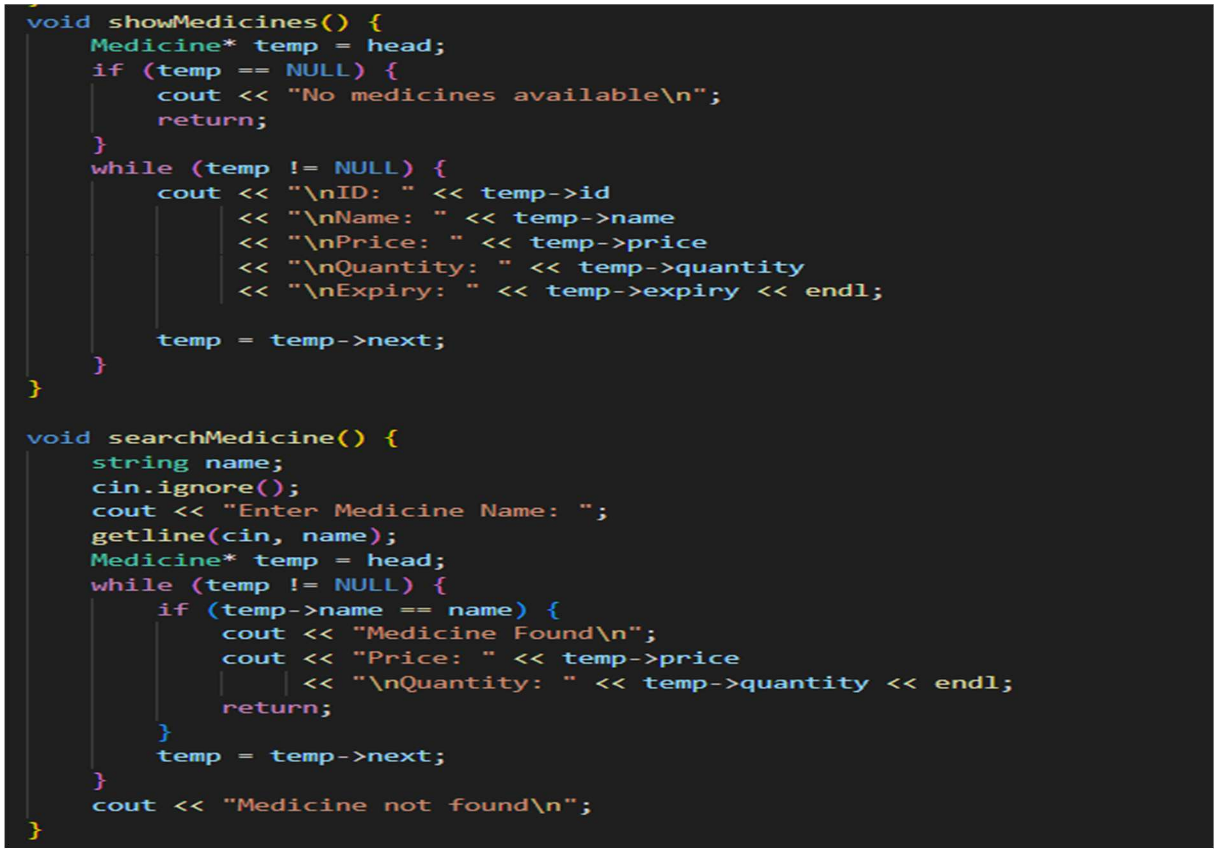
The Pharmacy Management System successfully demonstrates how data structures can be used to build real-world applications. The system provides an efficient way to manage medicine records, track inventory, generate bills, and monitor expiry dates. By using linked list and queue, the project shows how dynamic data structures can be applied to solve practical problems. The system reduces manual errors and improves efficiency in managing pharmacy operations. This project also helps improve understanding of programming concepts and data structures while providing a useful tool for inventory management.

Overall, the Pharmacy Management System achieves its objectives and provides a simple yet effective solution for managing medicine records digitally.

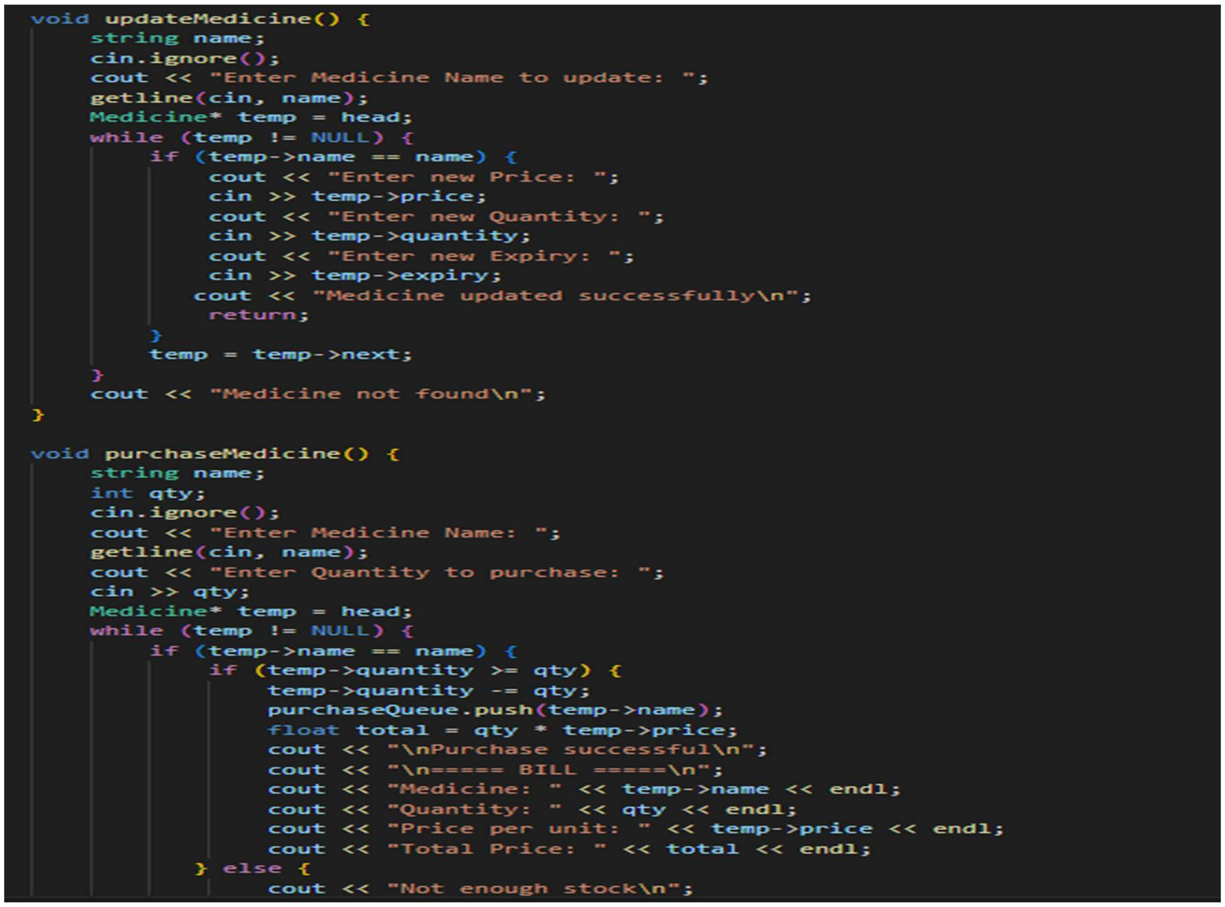
Snapshot of Code



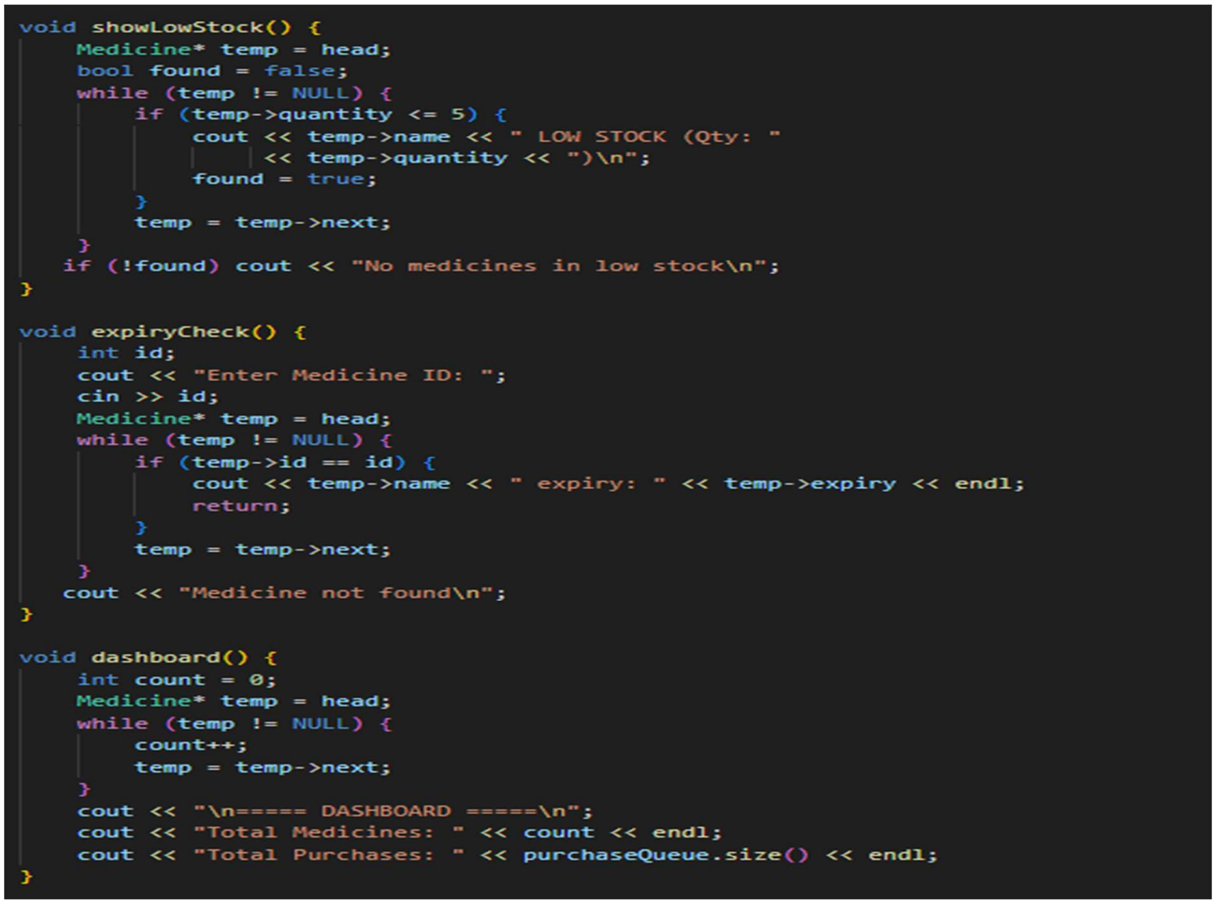
```cpp
void addMedicine() {
    int id;
    string name;
    cout << "Enter Medicine ID: ";
    cin >> id;
    cin.ignore();
    cout << "Enter Medicine Name: ";
    getline(cin, name);
    Medicine* temp = head;
    while (temp != NULL) {
        if (temp->id == id || temp->name == name) {
            cout << "Medicine with same ID or Name already exists\n";
            return;
        }
        temp = temp->next;
    }
    Medicine* newMed = new Medicine();
    newMed->id = id;
    newMed->name = name;
    cout << "Enter Price: ";
    cin >> newMed->price;
    cout << "Enter Quantity: ";
    cin >> newMed->quantity;
    cout << "Enter Expiry (MM/YY): ";
    cin >> newMed->expiry;
    newMed->next = NULL;
    if (head == NULL) head = newMed;
    else {
        temp = head;
        while (temp->next != NULL) temp = temp->next;
        temp->next = newMed;
    }
    cout << "Medicine added successfully\n";
```



```cpp
void showMedicines() {
    Medicine* temp = head;
    if (temp == NULL) {
        cout << "No medicines available\n";
        return;
    }
    while (temp != NULL) {
        cout << "\nID: " << temp->id
             << "\nName: " << temp->name
             << "\nPrice: " << temp->price
             << "\nQuantity: " << temp->quantity
             << "\nExpiry: " << temp->expiry << endl;

        temp = temp->next;
    }
}

void searchMedicine() {
    string name;
    cin.ignore();
    cout << "Enter Medicine Name: ";
    getline(cin, name);
    Medicine* temp = head;
    while (temp != NULL) {
        if (temp->name == name) {
            cout << "Medicine Found\n";
            cout << "Price: " << temp->price
                 << "\nQuantity: " << temp->quantity << endl;
            return;
        }
        temp = temp->next;
    }
    cout << "Medicine not found\n";
}
```



```cpp
void updateMedicine() {
    string name;
    cin.ignore();
    cout << "Enter Medicine Name to update: ";
    getline(cin, name);
    Medicine* temp = head;
    while (temp != NULL) {
        if (temp->name == name) {
            cout << "Enter new Price: ";
            cin >> temp->price;
            cout << "Enter new Quantity: ";
            cin >> temp->quantity;
            cout << "Enter new Expiry: ";
            cin >> temp->expiry;
           cout << "Medicine updated successfully\n";
            return;
        }
        temp = temp->next;
    }
    cout << "Medicine not found\n";
}

void purchaseMedicine() {
    string name;
    int qty;
    cin.ignore();
    cout << "Enter Medicine Name: ";
    getline(cin, name);
    cout << "Enter Quantity to purchase: ";
    cin >> qty;
    Medicine* temp = head;
    while (temp != NULL) {
        if (temp->name == name) {
            if (temp->quantity >= qty) {
                temp->quantity -= qty;
                purchaseQueue.push(temp->name);
                float total = qty * temp->price;
                cout << "\nPurchase successful\n";
                cout << "\n===== BILL =====\n";
                cout << "Medicine: " << temp->name << endl;
                cout << "Quantity: " << qty << endl;
                cout << "Price per unit: " << temp->price << endl;
                cout << "Total Price: " << total << endl;
            } else {
                cout << "Not enough stock\n";
```



```cpp
void showLowStock() {
    Medicine* temp = head;
    bool found = false;
    while (temp != NULL) {
        if (temp->quantity <= 5) {
            cout << temp->name << " LOW STOCK (Qty: "
                 << temp->quantity << ")\n";
            found = true;
        }
        temp = temp->next;
    }
   if (!found) cout << "No medicines in low stock\n";
}

void expiryCheck() {
    int id;
    cout << "Enter Medicine ID: ";
    cin >> id;
    Medicine* temp = head;
    while (temp != NULL) {
        if (temp->id == id) {
            cout << temp->name << " expiry: " << temp->expiry << endl;
            return;
        }
        temp = temp->next;
    }
   cout << "Medicine not found\n";
}

void dashboard() {
    int count = 0;
    Medicine* temp = head;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    cout << "\n===== DASHBOARD =====\n";
    cout << "Total Medicines: " << count << endl;
    cout << "Total Purchases: " << purchaseQueue.size() << endl;
}
```