**UNIVERSITY INSTITUTE OF COMPUTING**

**PROJECT REPORT**

**ON**

**PATIENT MANAGEMENT SYSTEM**

Program Name: BSC

Subject Name/Code: Data Structures(23CAT-201)

**Submitted by: Submitted to:**

**Name:** Franklin Francis **Name:**

**UID:** 23BSC 10079 **Designation:**

**Section:** BSC– 1 “B”

**Table of Contents**

Abstract……………………………………… (1)

Techniques and Data Structures Used………. (2)

System Configuration……….………………. (3)

System Overview …………………..……… (4)

Algorithm…………………………………… (5)

Flowchart……………………………………… (6)

Project source codes……………………………(7)

Project Limitations and Future Scope…… …. (8)

Conclusion……………………………………(9)

(1) ABSTRACT

This C++ patient management application is designed to streamline the management of basic healthcare operations such as patient registration, record management, and patient information retrieval. The system efficiently handles patient data using linked lists, allowing dynamic storage and quick access to patient records.

The program enables the creation of new patient records with details like name, age, gender, and disease. Patients receive a unique token number for easy identification. This application helps hospital staff manage patient information, ensuring organized and efficient record handling.

(2) Technique:

The application is developed using C++ with a focus on linked list data structures to manage patient records dynamically. This structure supports efficient data operations, including adding, deleting, searching, and displaying patient details.

* **Linked List Data Structure**: Patient records are stored dynamically using linked lists, which facilitates easy addition, deletion, and searching operations.
* **Dynamic Token Assignment**: Each patient is assigned a unique token number that increments automatically, ensuring no two patients have the same identification.
* **Memory Management**: A function is included to free memory upon exiting, ensuring efficient resource management.

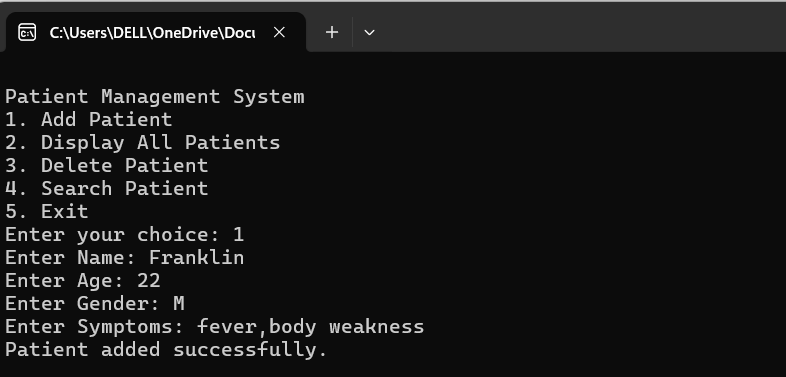
(3) System Configuration:

* **OS:** Windows 10 or Linux
* **Processor:** Intel Core i3 (minimum); Core i5 or higher recommended
* **RAM:** 4 GB (minimum); 8 GB recommended
* **Development Environment:** Any C++ IDE (e.g., Visual Studio, Code::Blocks) or Visual Studio Code with a C++ compiler (GCC or Microsoft C++ Compiler)

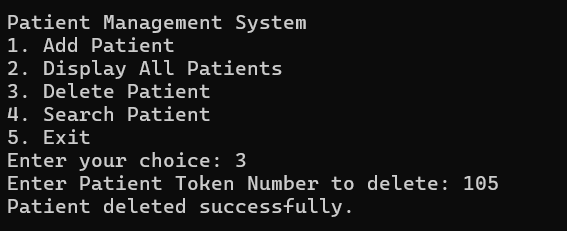
(4) System Overview

Input and Output:

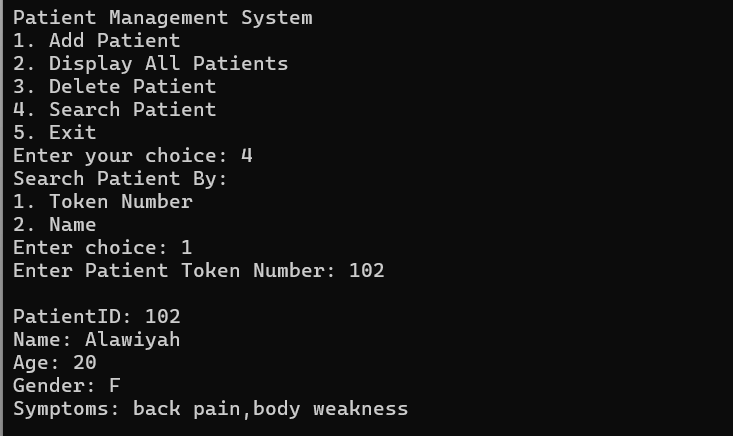
The system provides the following main menu options for hospital staff:

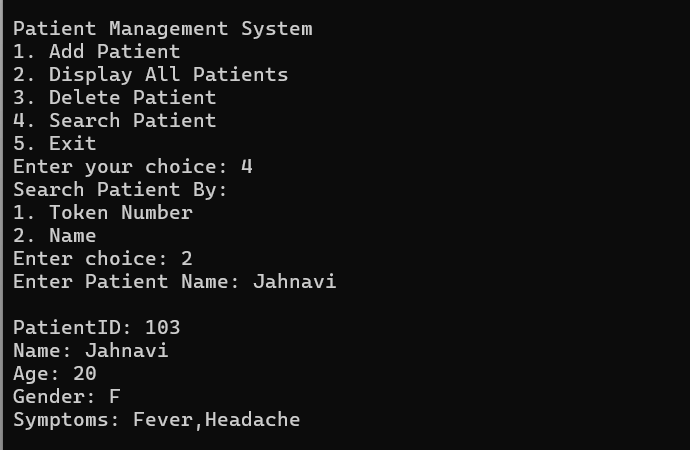
1. **Add Patient**: Enter the patient's details, including name, age, gender, and disease. A unique token number is assigned.
2. **Display All Patients**: Displays all patient records, showing the token number, name, age, gender, and disease.

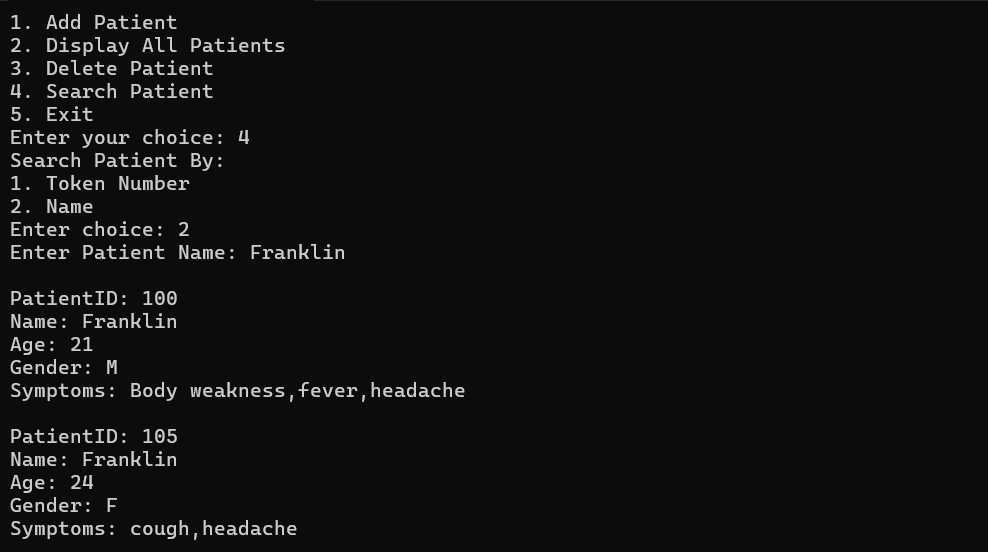
**3)Delete Patient**: Allows the deletion of a patient record by entering the patient’s token number.



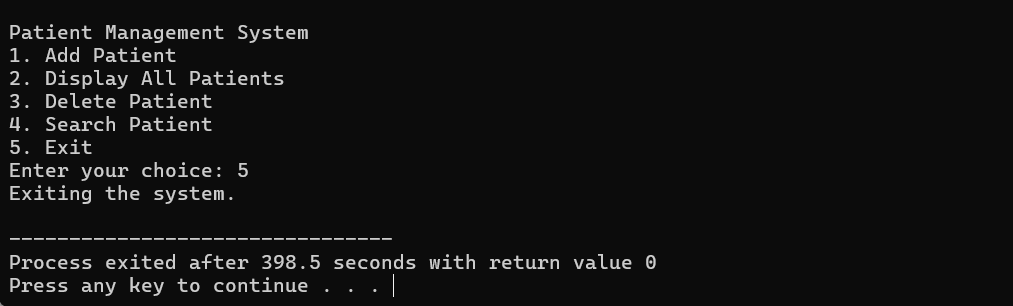
1. **Search Patient**: Search for a patient record by token number or name.
   1. By Patient’s Token Number;



b)**By Patient’s Name(single name available):**

** By Patient’s Name(more than one name available):**

1. **Exit**: Closes the program and frees memory.



(5) Algorithm:

Main Algorithm

1. **Initialize Variables:**
   * head (points to the first patient)
   * tokenCounter (initially 100)
2. **Main Menu Loop:**
   * Display options to the user.
   * Take user input for choice.
   * Execute corresponding function:
     + If **1**, call AddPatient()
     + If **2**, call DisplayPatients()
     + If **3**, call DeletePatient()
     + If **4**, call SearchPatient()
     + If **5**, call FreeMemory() and end program.

**SUB-ALGORITHMS**

1. **AddPatient(name, age, gender, disease)**
   * Create new patient with unique tokenCounter.
   * Set new patient's next pointer to NULL.
   * If list is empty, make head point to new patient.
   * Else, traverse to end of the list and link new patient.
   * Increment tokenCounter.
2. **DisplayPatients()**
   * If list is empty, display “No patients.”
   * Else, traverse list and display each patient’s details.
3. **DeletePatient(token\_number)**
   * If list is empty, display “No patients to delete.”
   * If head matches token\_number, remove head and update it.
   * Else, traverse list until token\_number is found.
   * If found, remove the node; if not, display “Patient not found.”
4. **SearchPatient()**
   * Ask for search type (token number or name).
   * For token number, traverse list and display details if found.
   * For name, traverse list and display details for all matches.
5. **FreeMemory()**
   * Traverse list, deleting each patient node to free memory.
   * Set head to NULL after all nodes are deleted.

(6) FLOW CHART:

**(7) SOURCE CODES:**

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

struct Patient {

int token\_number;

string name;

int age;

string gender;

string Symptoms;

Patient\* next;

};

Patient\* head = NULL;

int tokenCounter = 100;

void addPatient(string name, int age, string gender, string Symptoms) {

Patient\* newPatient = new Patient{tokenCounter++, name, age, gender, Symptoms, nullptr};

if (head == NULL) {

head = newPatient;

} else {

Patient\* temp = head;

while (temp->next) {

temp = temp->next;

}

temp->next = newPatient;

}

cout << "Patient added successfully.\n";

}

void displayPatients() {

if (!head) {

cout << "No patients to display.\n";

return;

}

cout << "\n--------------------------------------------------------------------------------\n";

cout << left << setw(20) << "Token Number"

<< setw(20) << "Name"

<< setw(10) << "Age"

<< setw(10) << "Gender"

<< setw(20) << "Symptoms"

<< "\n";

cout << "--------------------------------------------------------------------------------\n";

Patient\* temp = head;

while (temp) {

cout << left << setw(20) << temp->token\_number

<< setw(20) << temp->name

<< setw(10) << temp->age

<< setw(10) << temp->gender

<< setw(20) << temp->Symptoms

<< "\n";

temp = temp->next;

}

cout << "--------------------------------------------------------------------------------\n";

}

void deletePatient(int token\_number) {

if (!head) {

cout << "No patients to delete.\n";

return;

}

if (head->token\_number == token\_number) {

Patient\* toDelete = head;

head = head->next;

delete toDelete;

cout << "Patient deleted successfully.\n";

return;

}

Patient\* temp = head;

while (temp->next && temp->next->token\_number != token\_number) {

temp = temp->next;

}

if (temp->next && temp->next->token\_number == token\_number) {

Patient\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

cout << "Patient deleted successfully.\n";

} else {

cout << "Patient with Token Number " << token\_number << " not found.\n";

}

}

void searchPatient() {

int searchChoice;

cout << "Search Patient By:\n";

cout << "1. Token Number\n";

cout << "2. Name\n";

cout << "Enter choice: ";

cin >> searchChoice;

if (searchChoice == 1) {

int token\_number;

cout << "Enter Patient Token Number: ";

cin >> token\_number;

Patient\* temp = head;

while (temp) {

if (temp->token\_number == token\_number) {

cout << "\nPatientID: " << temp->token\_number

<< "\nName: " << temp->name

<< "\nAge: " << temp->age

<< "\nGender: " << temp->gender

<< "\nSymptoms: " << temp->Symptoms << "\n";

return;

}

temp = temp->next;

}

cout << "Patient with Token Number " << token\_number << " not found.\n";

}

else if (searchChoice == 2) {

string name;

cout << "Enter Patient Name: ";

cin.ignore();

getline(cin, name);

Patient\* temp = head;

bool found = false;

while (temp) {

if (temp->name == name) {

cout << "\nPatientID: " << temp->token\_number

<< "\nName: " << temp->name

<< "\nAge: " << temp->age

<< "\nGender: " << temp->gender

<< "\nSymptoms: " << temp->Symptoms<< "\n";

found = true;

}

temp = temp->next;

}

if (!found) {

cout << "Patient with name \"" << name << "\" not found.\n";

}

} else {

cout << "Invalid choice. Returning to main menu.\n";

}

}

void freeMemory() {

Patient\* temp = head;

while (temp) {

Patient\* next = temp->next;

delete temp;

temp = next;

}

head = NULL;

}

int main() {

int choice, age;

string name, gender, Symptoms;

for(;;) {

cout << "\nPatient Management System\n";

cout << "1. Add Patient\n";

cout << "2. Display All Patients\n";

cout << "3. Delete Patient\n";

cout << "4. Search Patient\n";

cout << "5. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Age: ";

cin >> age;

cout << "Enter Gender: ";

cin >> gender;

cout << "Enter Symptoms: ";

cin.ignore();

getline(cin,Symptoms);

addPatient(name, age, gender, Symptoms);

break;

case 2:

displayPatients();

break;

case 3:

int token\_number\_to\_delete;

cout << "Enter Patient Token Number to delete: ";

cin >> token\_number\_to\_delete;

deletePatient(token\_number\_to\_delete);

break;

case 4:

searchPatient();

break;

case 5:

cout << "Exiting the system.\n";

freeMemory();

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

**(7) PROJECT LIMITATIONS AND FUTURE SCOPE.**

**A. Lack of Data Persistence**

* The current system does not store patient information persistently; all data is lost when the program is closed.

solution

Adding file handling (e.g., writing to and reading from files) or database integration would be required to make patient records persistent.

**B. Limited Search Capabilities**

* The search functionality is basic and may be inefficient, especially if the list grows large, as it relies on linear traversal for each search.
* This can slow down the program considerably with a larger number of patients.

Solution

Implementing a more efficient data structure, like a binary search tree, or indexing could help.

**C. No Input Validation for Data Types**

* The system assumes correct input for details like age and token number. It lacks robust error handling and validation, which could lead to incorrect or invalid entries.

Solution

Adding checks for input types (e.g., integers for age) and acceptable ranges (e.g., age between 0 and 120) would make the system more user-friendly

**D. Limited Error Handling and Feedback**

* The current code provides minimal feedback on incorrect entries or invalid choices, potentially leading to a poor user experience.

Solution

Adding more descriptive error messages or hints on required input formats would enhance usability.

**(8) CONCLUSION**

Overall, this project serves as a valuable example of how linked lists can be leveraged for data organization and manipulation in real-world applications. With further development, including potential database integration and improved user interface options, this Patient Management System could evolve into a comprehensive, scalable solution for patient information management in larger healthcare settings. This project not only underscores the strengths of data structures but also opens the door to exploring more advanced software design patterns and data management strategies in healthcare applications.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*THANK YOU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*