2023/2024

Mini Project Report

Patient Diagnostic System

Prepared by:

Amini Safa

Groupe: 05

SOMMAIRE

INTRODUCTION

This project involves the development of a patient diagnostic system that simulates the diagnostic process in a healthcare setting. The system uses linked lists to efficiently manage patient records and diagnostics, while also incorporating doctor information to improve the diagnostic process.

In healthcare, effective patient information management is critical to ensuring quality care delivery. Traditional paper-based systems are error-prone, time-consuming, and inefficient, emphasizing the need for digital solutions.

Using linked lists as a fundamental data structure has several advantages, including dynamic memory allocation, efficient insertion and deletion operations, and flexibility in dealing with variable-sized data.

The project, divided into distinct parts, includes the comprehensive management of patient records and diagnostic data:

Part 1: *Patient Management*
Part 2: *Diagnostic Recording*
Part 3: *Doctor Integration*

The development of this system is difficult due to the dynamic nature of patient data management and the need to incorporate changing parameters, such as diagnostic updates and doctor associations. However, the system's design prioritizes accuracy and efficiency in order to streamline the diagnostic process while providing comprehensive patient care.

PROBLEMATIC

How can we create an efficient C program for managing patient records and diagnostic data that uses linked lists to ensure modularity and scalability?

The challenge is to create and implement a patient management system that effectively manages patient records and diagnostic information in a healthcare setting. This system should include features for adding, updating, deleting, searching, and printing patient records and diagnostics while remaining modular and using linked lists for data organization.

To address this challenge, we have:

Key Data to be Manipulated:

Patient Fields:

PatientList: Linked list all diagnostics have been inserted.

PatientID: A unique identifier.

FullName: Patient's full name.

Age: Patient's age.

Gender: Patient's gender.

Height: Patient's height.

Diagnostic Fields:

DiagnosticList: Linked list all diagnostics have been inserted.

PatientID: Unique identifier.

Diagnosis: Like Hypertension, Type 2 Diabetes...

DiagDate: Date of diagnosis.

NextAppointment: Date of the next appointment.

Doctor Fields:

DoctorList: Linked list all doctors have been inserted.

DoctorID: Unique identifier.

FullNameDoc: Doctor's full name.

Specialty: Doctor's specialty.

This is to obtain the desired objectives (results):

Desired Objectives (Results):

Patient Management:

- *Ability to add, update, delete, search, and print patient records.*
- *Efficient organization of patient records within a linked list structure.*
- *Seamless handling of patient data to ensure accuracy and accessibility.*

Diagnostic Management:

•Add, update, delete, search, print records.

•Integrate diagnostic information with patient records.

•Track patient health status and treatment history.

Doctor Management:

•Incorporation of doctor ID, fullNameDoc, specialty fields.

•Associating doctors with specific diagnostics.

•Integration of doctor and diagnostic records for tracking diagnostics.

System Functionality:

•Manages large patient, diagnostic, doctor data effectively.

•Utilizes modular design principles for maintenance and scalability.

•Enhances system performance for responsiveness and reliability.

ANALYSIS

The patient diagnostic system project intends to design an effective healthcare management system, taking into account inputs and outputs:

Inputs:

For the patient records:

- *PatientID.*
- *Age.*
- *Height*
- *FullName.*
- *Gender.*

For the diagnostic records:

- *PatientID.*
- *DiagDate.*
- *Diagnosis.*
- *NextAppointment.*

For the doctor records:

- *DoctorID.*
- *Specialty.*
- *FullNameDoc.*

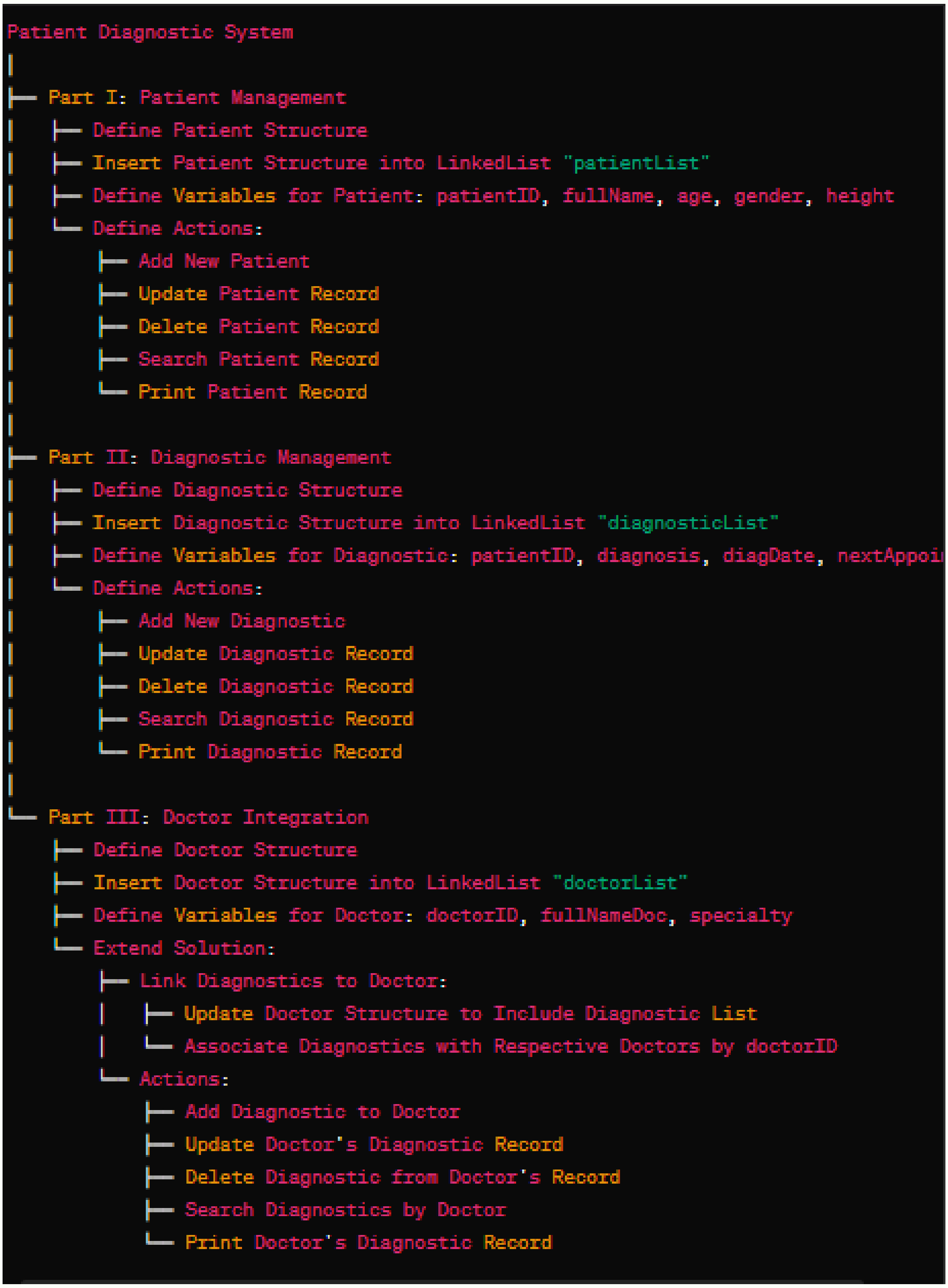
Outputs:

- Output messages confirm the successful addition, update, deletion, or retrieval of patient, diagnostic, or doctor records.
- Printed patient records containing patient, diagnostic, or doctor information.
- Error messages indicate unsuccessful operations or invalid inputs.
- Added doctor records to the doctorList, including a list of diagnoses made by each doctor identified by doctorID.

The theoretical tools used:

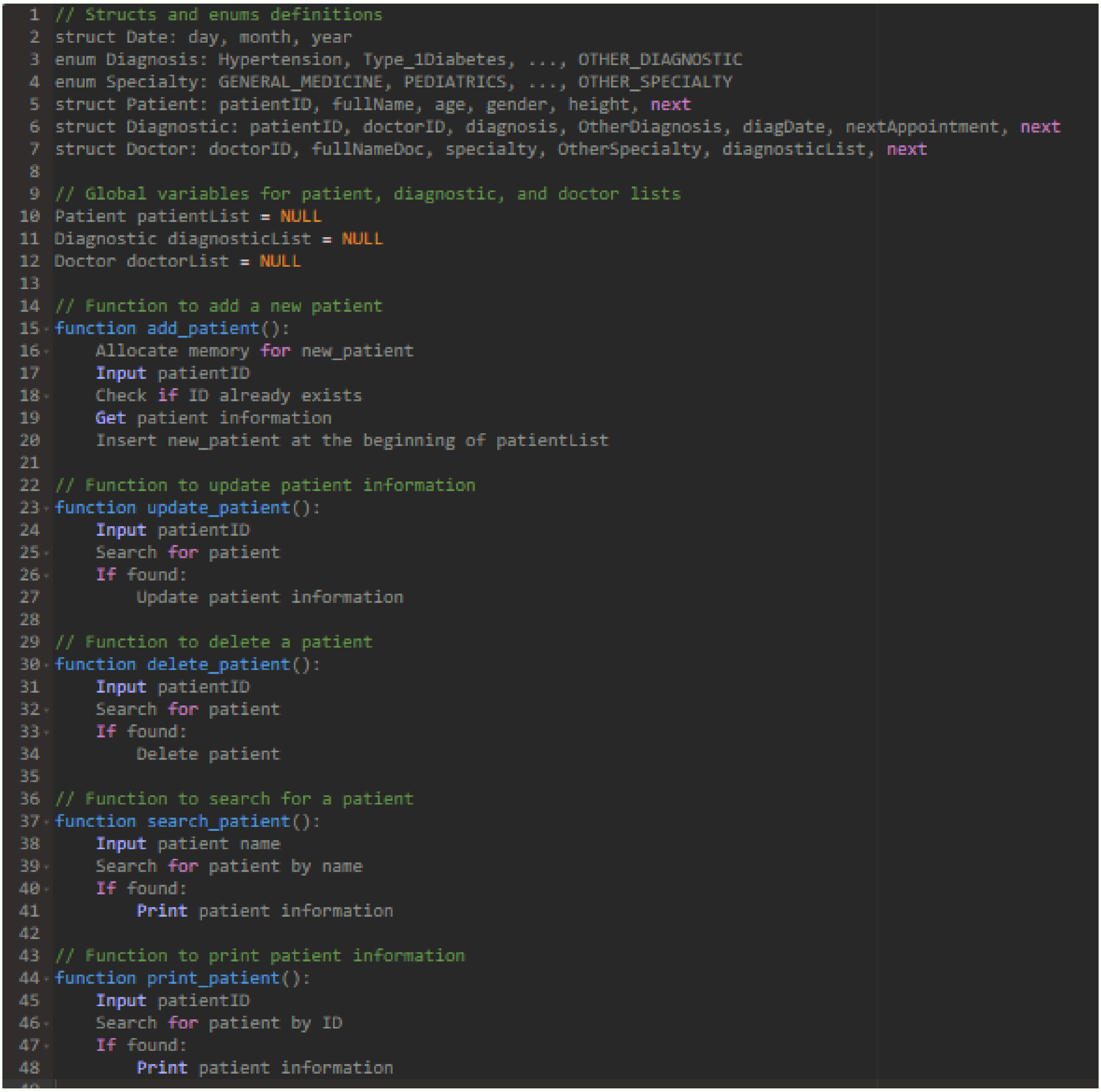
- Linked Lists: Use the linked list data structure to efficiently manage patient, diagnostic, and doctor records.
- Enumeration: Use enumeration to represent different diagnosis options.
- Custom Date Structure: Create a custom date structure to manage dates more efficiently.
- Modularity: The system is designed using modular principles, This promotes code reusability, maintainability, and scalability.

The top-down analysis outlines key steps:

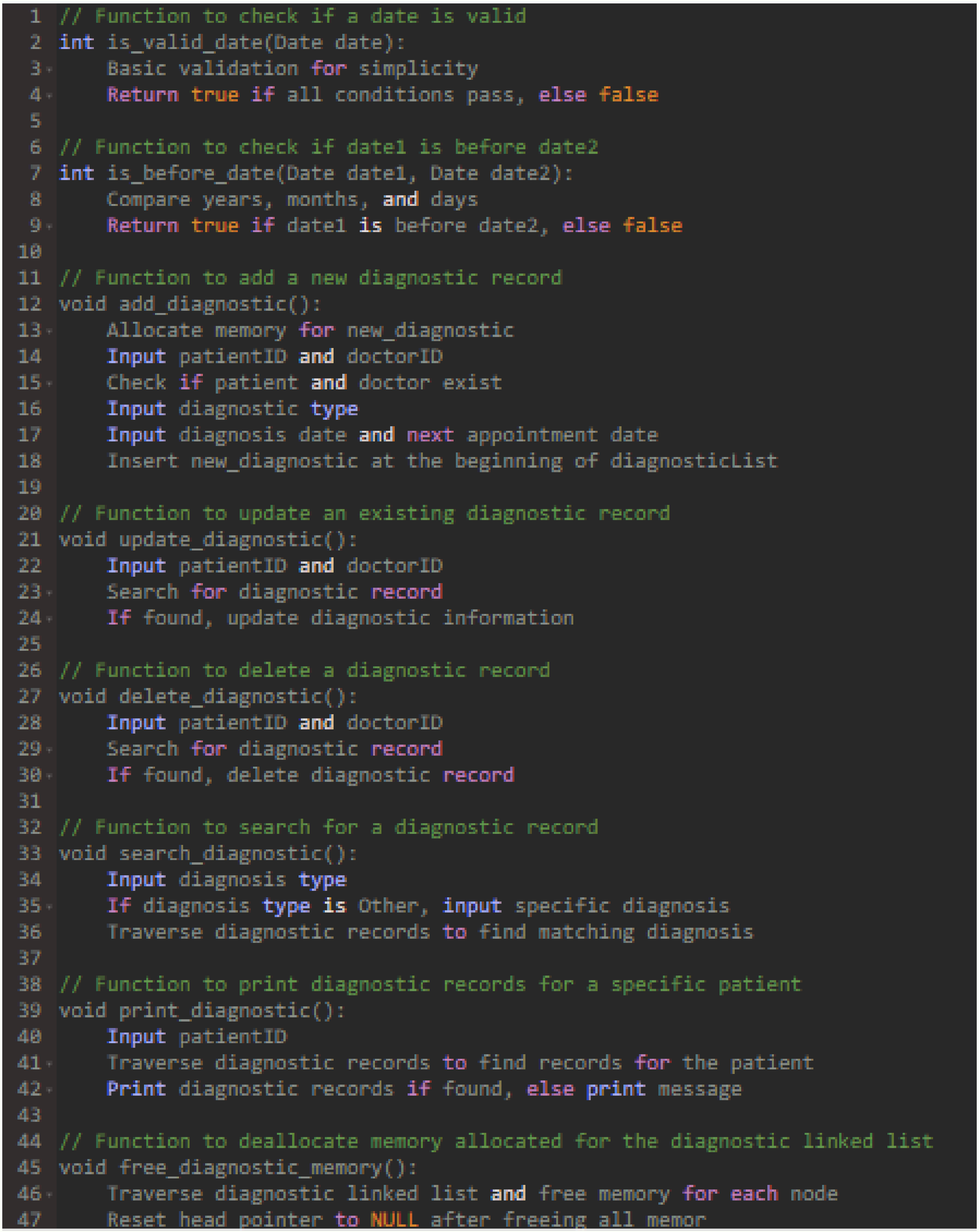


```
Patient Diagnostic System
│
├── Part I: Patient Management
│   ├── Define Patient Structure
│   ├── Insert Patient Structure into LinkedList "patientList"
│   ├── Define Variables for Patient: patientID, fullName, age, gender, height
│   └── Define Actions:
│       ├── Add New Patient
│       ├── Update Patient Record
│       ├── Delete Patient Record
│       ├── Search Patient Record
│       └── Print Patient Record
│
├── Part II: Diagnostic Management
│   ├── Define Diagnostic Structure
│   ├── Insert Diagnostic Structure into LinkedList "diagnosticList"
│   ├── Define Variables for Diagnostic: patientID, diagnosis, diagDate, nextAppoi
│   └── Define Actions:
│       ├── Add New Diagnostic
│       ├── Update Diagnostic Record
│       ├── Delete Diagnostic Record
│       ├── Search Diagnostic Record
│       └── Print Diagnostic Record
│
└── Part III: Doctor Integration
    ├── Define Doctor Structure
    ├── Insert Doctor Structure into LinkedList "doctorList"
    ├── Define Variables for Doctor: doctorID, fullNameDoc, specialty
    └── Extend Solution:
        ├── Link Diagnostics to Doctor:
        │   ├── Update Doctor Structure to Include Diagnostic List
        │   └── Associate Diagnostics with Respective Doctors by doctorID
        └── Actions:
            ├── Add Diagnostic to Doctor
            ├── Update Doctor's Diagnostic Record
            ├── Delete Diagnostic from Doctor's Record
            ├── Search Diagnostics by Doctor
            └── Print Doctor's Diagnostic Record
```

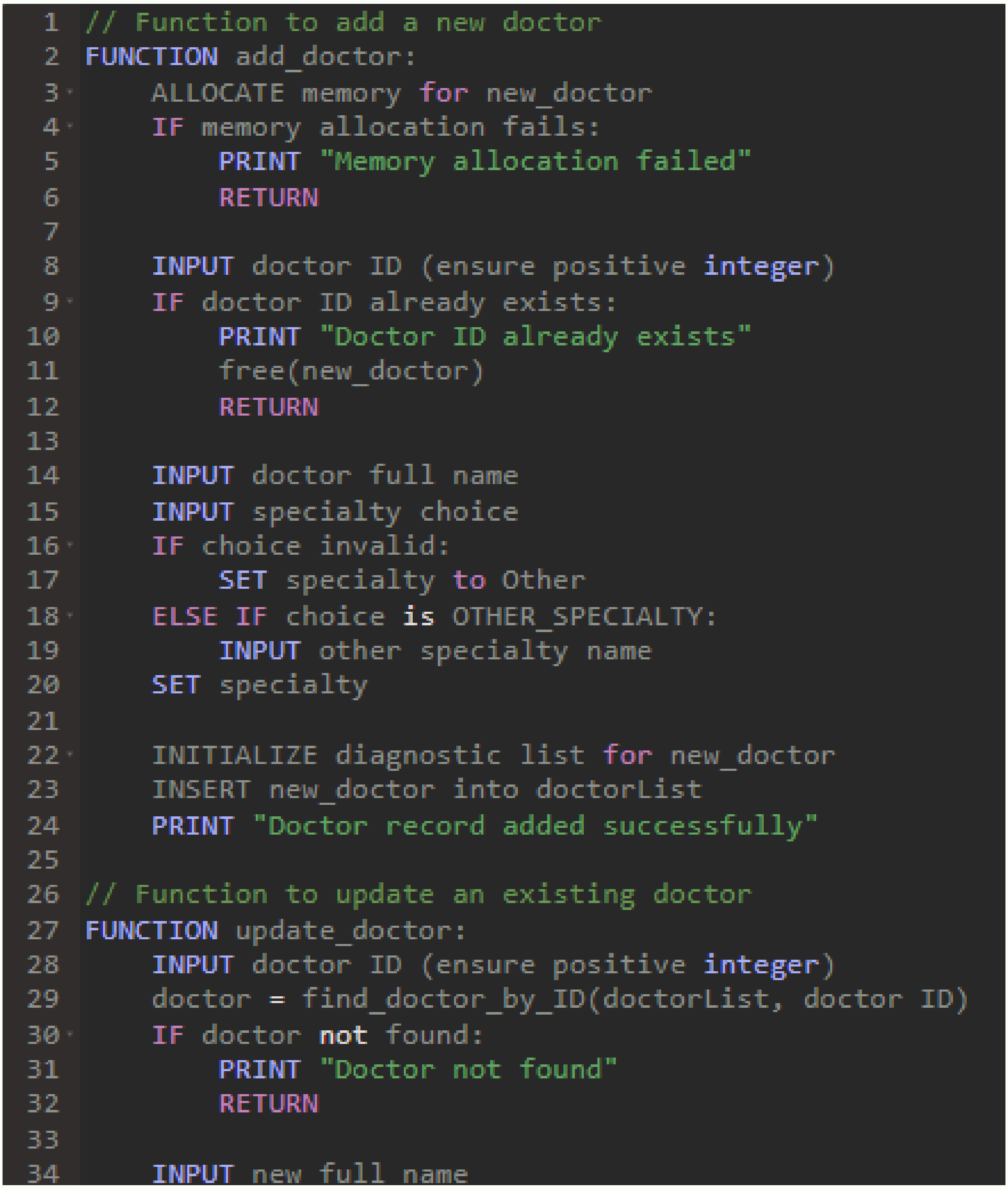
ALGORITHM



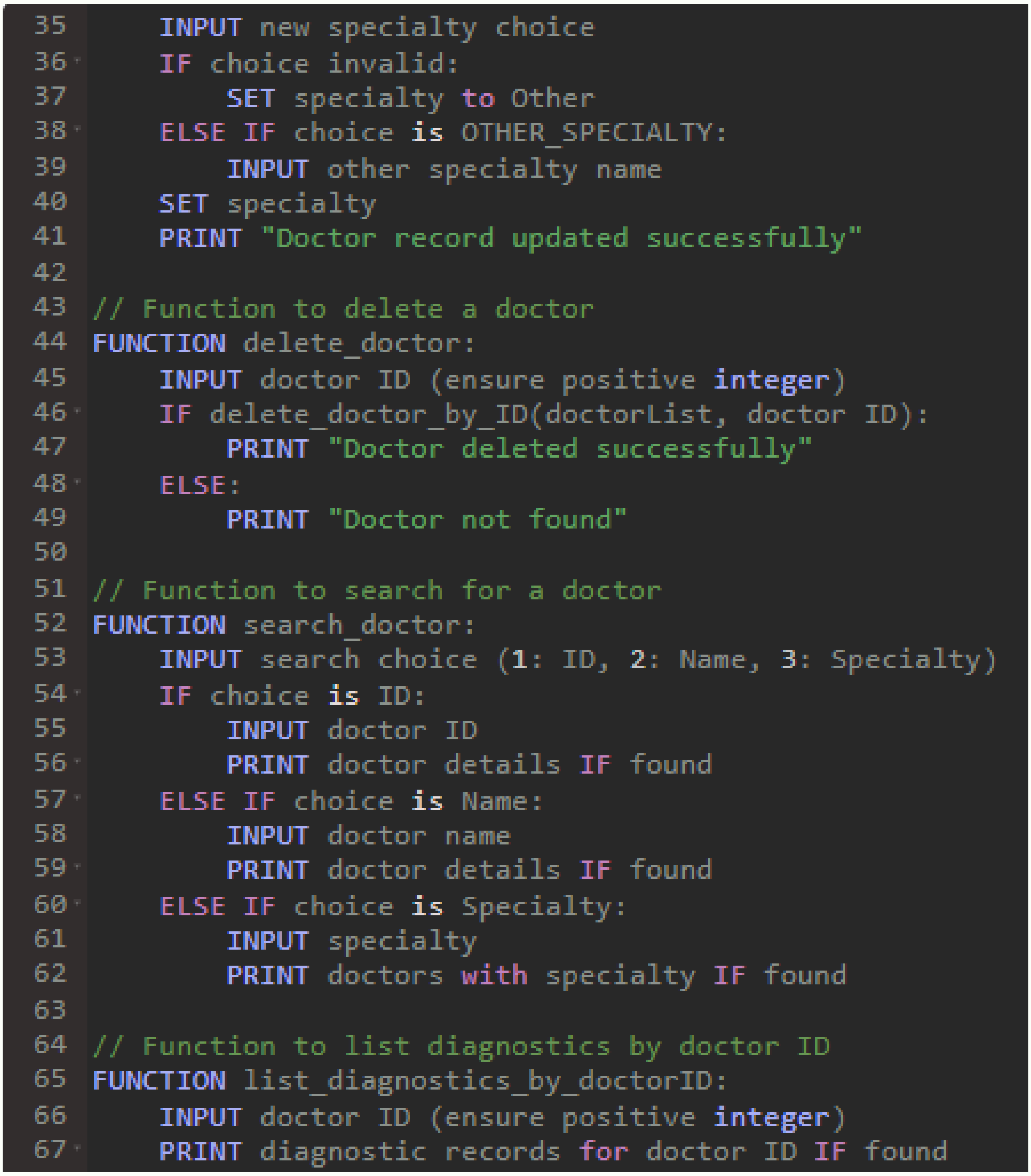
```
// Structs and enums definitions
struct Date: day, month, year
enum Diagnosis: Hypertension, Type_1Diabetes, ..., OTHER_DIAGNOSTIC
enum Specialty: GENERAL_MEDICINE, PEDIATRICS, ..., OTHER_SPECIALTY
struct Patient: patientID, fullName, age, gender, height, next
struct Diagnostic: patientID, doctorID, diagnosis, OtherDiagnosis, diagDate, nextAppointment, next
struct Doctor: doctorID, fullNameDoc, specialty, OtherSpecialty, diagnosticList, next

// Global variables for patient, diagnostic, and doctor lists
Patient patientList = NULL
Diagnostic diagnosticList = NULL
Doctor doctorList = NULL

// Function to add a new patient
function add_patient():
    Allocate memory for new_patient
    Input patientID
    Check if ID already exists
    Get patient information
    Insert new_patient at the beginning of patientList

// Function to update patient information
function update_patient():
    Input patientID
    Search for patient
    If found:
        Update patient information

// Function to delete a patient
function delete_patient():
    Input patientID
    Search for patient
    If found:
        Delete patient

// Function to search for a patient
function search_patient():
    Input patient name
    Search for patient by name
    If found:
        Print patient information

// Function to print patient information
function print_patient():
    Input patientID
    Search for patient by ID
    If found:
        Print patient information
```



```
// Function to check if a date is valid
int is_valid_date(Date date):
    Basic validation for simplicity
    Return true if all conditions pass, else false

// Function to check if date1 is before date2
int is_before_date(Date date1, Date date2):
    Compare years, months, and days
    Return true if date1 is before date2, else false

// Function to add a new diagnostic record
void add_diagnostic():
    Allocate memory for new_diagnostic
    Input patientID and doctorID
    Check if patient and doctor exist
    Input diagnostic type
    Input diagnosis date and next appointment date
    Insert new_diagnostic at the beginning of diagnosticList

// Function to update an existing diagnostic record
void update_diagnostic():
    Input patientID and doctorID
    Search for diagnostic record
    If found, update diagnostic information

// Function to delete a diagnostic record
void delete_diagnostic():
    Input patientID and doctorID
    Search for diagnostic record
    If found, delete diagnostic record

// Function to search for a diagnostic record
void search_diagnostic():
    Input diagnosis type
    If diagnosis type is Other, input specific diagnosis
    Traverse diagnostic records to find matching diagnosis

// Function to print diagnostic records for a specific patient
void print_diagnostic():
    Input patientID
    Traverse diagnostic records to find records for the patient
    Print diagnostic records if found, else print message

// Function to deallocate memory allocated for the diagnostic linked list
void free_diagnostic_memory():
    Traverse diagnostic linked list and free memory for each node
    Reset head pointer to NULL after freeing all memor
```

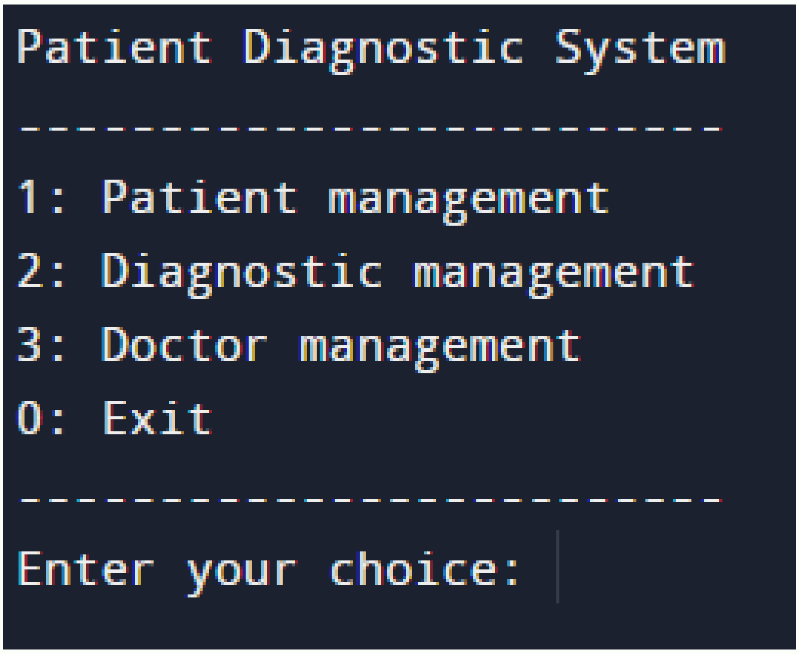


```
// Function to add a new doctor
FUNCTION add_doctor:
    ALLOCATE memory for new_doctor
    IF memory allocation fails:
        PRINT "Memory allocation failed"
        RETURN

    INPUT doctor ID (ensure positive integer)
    IF doctor ID already exists:
        PRINT "Doctor ID already exists"
        free(new_doctor)
        RETURN

    INPUT doctor full name
    INPUT specialty choice
    IF choice invalid:
        SET specialty to Other
    ELSE IF choice is OTHER_SPECIALTY:
        INPUT other specialty name
    SET specialty

    INITIALIZE diagnostic list for new_doctor
    INSERT new_doctor into doctorList
    PRINT "Doctor record added successfully"

// Function to update an existing doctor
FUNCTION update_doctor:
    INPUT doctor ID (ensure positive integer)
    doctor = find_doctor_by_ID(doctorList, doctor ID)
    IF doctor not found:
        PRINT "Doctor not found"
        RETURN

    INPUT new full name
```

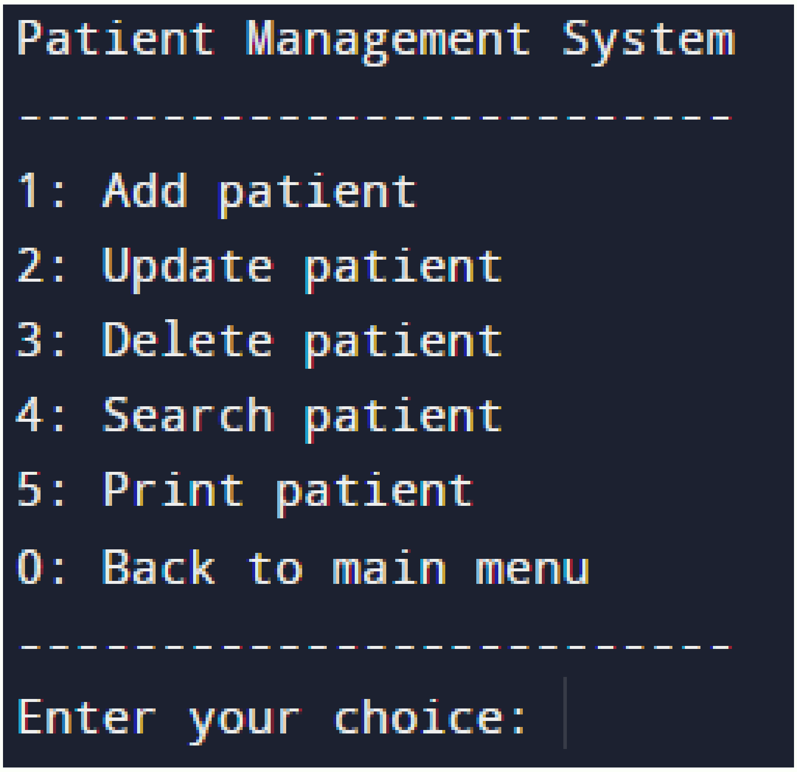


```
    INPUT new specialty choice
    IF choice invalid:
        SET specialty to Other
    ELSE IF choice is OTHER_SPECIALTY:
        INPUT other specialty name
    SET specialty
    PRINT "Doctor record updated successfully"

// Function to delete a doctor
FUNCTION delete_doctor:
    INPUT doctor ID (ensure positive integer)
    IF delete_doctor_by_ID(doctorList, doctor ID):
        PRINT "Doctor deleted successfully"
    ELSE:
        PRINT "Doctor not found"

// Function to search for a doctor
FUNCTION search_doctor:
    INPUT search choice (1: ID, 2: Name, 3: Specialty)
    IF choice is ID:
        INPUT doctor ID
        PRINT doctor details IF found
    ELSE IF choice is Name:
        INPUT doctor name
        PRINT doctor details IF found
    ELSE IF choice is Specialty:
        INPUT specialty
        PRINT doctors with specialty IF found

// Function to list diagnostics by doctor ID
FUNCTION list_diagnostics_by_doctorID:
    INPUT doctor ID (ensure positive integer)
    PRINT diagnostic records for doctor ID IF found
```

PROGRAMMING

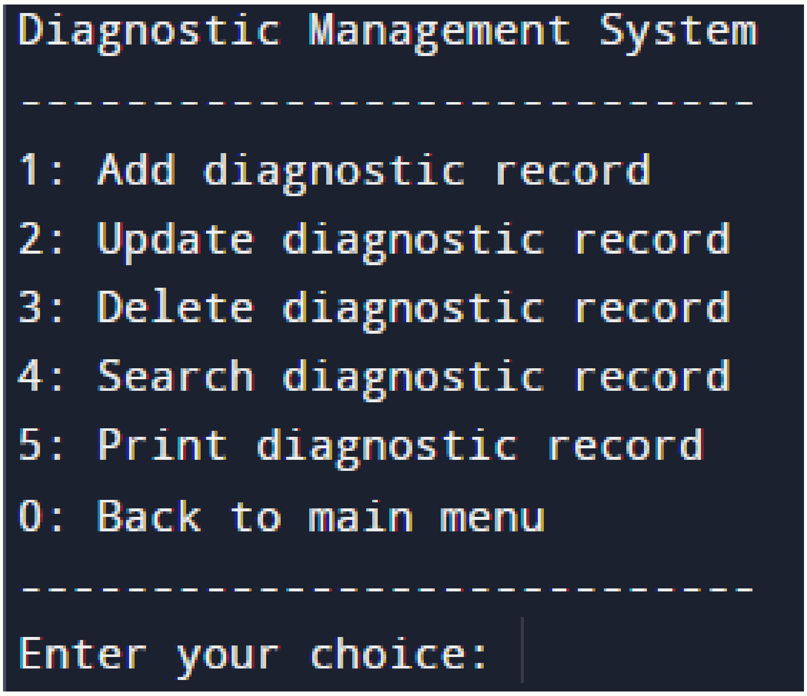


```
Patient Diagnostic System
-------------------------
1: Patient management
2: Diagnostic management
3: Doctor management
0: Exit
-------------------------
Enter your choice:
```

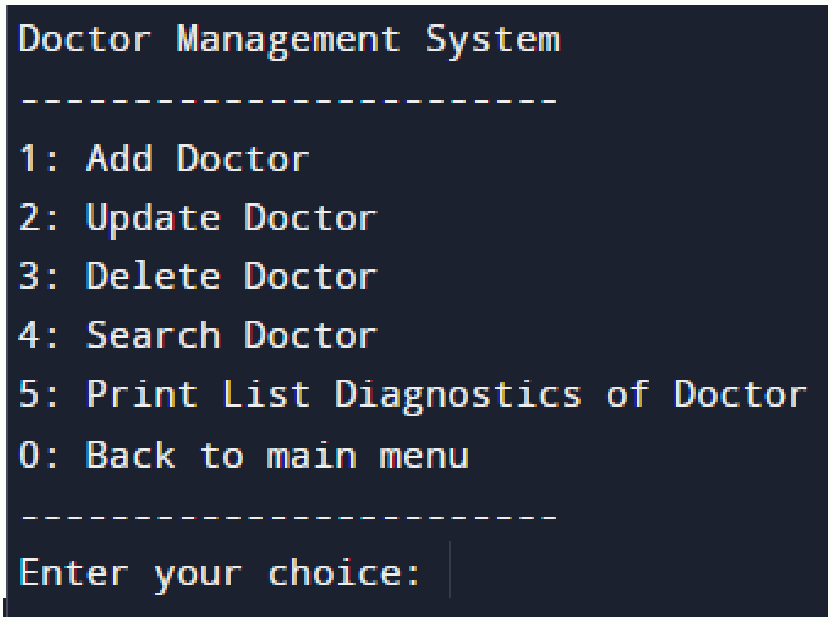


```
Patient Management System
-------------------------
1: Add patient
2: Update patient
3: Delete patient
4: Search patient
5: Print patient
0: Back to main menu
-------------------------
Enter your choice:
```

PROGRAMMING



```
Diagnostic Management System
----------------------------
1: Add diagnostic record
2: Update diagnostic record
3: Delete diagnostic record
4: Search diagnostic record
5: Print diagnostic record
0: Back to main menu
----------------------------
Enter your choice:
```



```
Doctor Management System
------------------------
1: Add Doctor
2: Update Doctor
3: Delete Doctor
4: Search Doctor
5: Print List Diagnostics of Doctor
0: Back to main menu
------------------------
Enter your choice:
```

CONCLUSION

The project aimed to develop a patient diagnostic system using linked lists to manage patient records and simulate the diagnostic process in a healthcare setting. It served as an educational platform to improve skills learned in the Algorithms and Data Structures 2 (ADS2) module, such as problem analysis, algorithm design, modularity, and proficiency in the C programming language. The project involved meticulously defining structures for patients, diagnostics, and doctors, and creating a modular solution for efficient management of each functionality.

However, the project faced challenges in ensuring seamless integration and functionality across various components, particularly in managing linked lists and ensuring data integrity. This necessitated extensive testing and debugging to ensure program reliability and correctness. Future improvements include streamlining the user interface, investigating advanced data structures and algorithms, and adding features like appointment scheduling and prescription management.

04

In summary, the project provided an excellent opportunity to put theoretical concepts into practice, reinforcing fundamental skills in algorithmic problem-solving and programming. The project's completion emphasizes the importance of perseverance and collaborative problem-solving in software development. Leveraging feedback and refining the solution iteratively will be critical for achieving a reliable and efficient patient diagnostic system.