"Hospital Management System (HMS)"’

Introduction in Computer Programming 1

Submitted By

Maria Victoria N. Esteban

January 29, 2025

TABLE OF CONTENTS

Introduction

1.1 Overview

The Hospital Management System (HMS) is a software solution designed to streamline the management of hospital operations. The system is developed using the C programming

language and leverages advanced C concepts like data structures and functions to manage patient records, doctor assignments, and appointment scheduling. The system avoids using file handling to keep it simple and focused on core functionalities.

1.2 Project description

Hospital Management System (HMS) key features include patient management, doctor management, appointment scheduling, and menu-driven interfaces.The system uses structured data models to represent patients, doctors, and appointments. These data models are defined using struct in C. Hospital Management System (HMS) core functionalities can add new patients, display patient details, search for patients, schedule appointments, assign doctor to patient, and update appointment status.

1.3 Purpose and scope of the system

Hospital Management Systems (HMS) are powerful, flexible, and easy to use and are designed and developed to deliver real conceivable benefits to hospitals. The main purpose of developing HMS is to help Hospital companies to be more efficient in registration of their patients and manage appointments, and records of patients. The system will be used as the application that serves hospitals, clinics, or other health institutions. The intention of the system is to increase the number of patients that can be treated and managed properly.If the hospital management system is file based, management of the hospital has to put much effort on securing the files. They can be easily damaged by fire, insects and natural disasters. Also could be misplaced by losing data and information.

System Objectives

The main objectives of a Hospital Management System is to efficiently manage patient records, facilitate doctor assignment and appointment scheduling, and ensure accurate and

```
Hospital Management System
1. Add New Patient
2. Add New Doctor
3. Display Patient Details
4. Search for Patient
5. Schedule Appointment
6. Assign Doctor to Patient
7. Update Appointment Status
8. View All Appointments
9. Exit
```

ystem for better patient care, reduce formation system) report on demand coordination among the different

The Hospital management system features and description are shown below:

Picture 1. Hospital Management System Features

- ❖ Add New Patients
  - ➢ Its function is to add new patients, where its personal details (name, age, gender, diagnosis) is collected.
  - ➢ Store the patient information in an array.
- ❖ Display Patients Details
  - ➢ Its function is to show the patients data that includes personal information by inputting the patient ID.
  - ➢ Iterate over the patient array and display details.
- ❖ Search for Patients

  - ➢ Its function is to look up the patient data by searching its patient ID.
  - ➢ Display the details if a match is found.
- ❖ Schedule Appointment

- ➢ Its function is to set an appointment for check-ups, where date and time is set-up.
- ➢ Validate the input and store the appointment information.

❖ Assign Doctor to Patient

- ➢ Its function is to assign a Doctor to the Patient that seeks medical assistance, by setting it just put the patient ID and Doctor ID.

❖ Update Appointment Status

- ➢ Its function is to update appointment status if there is a need to reschedule and cancel an appointment.

❖ Add New Doctor

- ➢ Its function is to add new doctors, where its status data will be recorded.

❖ View all Appointments

- ➢ Its function is to see all appointments data, it is the summary and status of patient information.

❖ Exit

- ➢ Its function is to leave the system.

❖ Menu-Driven Interface

- ➢ A user-friendly menu to navigate through different options and functionalities.

System Design

In the Hospital Management System (HMS), we use structured data models to represent patients, doctors, and appointments. These data models are defined using struct in C. Below are the descriptions of each data structure:

1. Patient Structure

- ➢ The Patient structure is used to store information about a patient. This includes their unique ID, name, age, gender, and diagnosis details.id: An integer that uniquely identifies each patient.

- **name:** A string to store the patient's name.

- **age**: An integer to store the patient's age.
- **gender**: A string to store the patient's gender.
- **diagnosis**: A string to store the patient's diagnosis details.

2. Doctor Structure

➢ The Doctor structure is used to store information about a doctor. This includes their unique ID, name, and specialization.

- **id:** An integer that uniquely identifies each doctor.
- **name**: A string to store the doctor's name.
- **specialization:** A string to store the doctor's area of specialization.

3. Appointment Structure

➢ The Appointment structure is used to store information about an appointment. This includes the appointment's unique ID, the patient's ID, the doctor's ID, the date and time of the appointment, and the status of the appointment.id: An integer that uniquely identifies each appointment.

- **patient_id:** An integer to store the ID of the patient associated with the appointment.
- **doctor_id**: An integer to store the ID of the doctor assigned to the appointment.
- **date:** A string to store the date of the appointment in the format YYYY-MM-DD.
- **time:** A string to store the time of the appointment in the format HH:MM.
- **status:** A string to store the status of the appointment (e.g., Scheduled, Completed, Cancelled).

Appendix A

System Implementation

The compiler tool used in creating the Hospital management system in C program is GDB compiler that can be used via website. This system consists of three core modules: Patient Management, Appointment Scheduling, and Doctor Management. Below are the key components of each module along with code snippets and explanations

1. Patient Management Module

This module handles adding new patients and displaying patient details.

```
void add_patient() {
    if (patient_count >= MAX_PATIENTS) {
        printf("Patient limit reached. Cannot add more patients.\n");
        return;
    }

    Patient p;
    p.id = patient_count + 1;

    printf("Enter patient name: ");
    scanf(" %[^\n]", p.name);
    printf("Enter patient age: ");
    scanf("%d", &p.age);
    printf("Enter patient gender: ");
    scanf(" %[^\n]", p.gender);
    printf("Enter patient diagnosis: ");
    scanf(" %[^\n]", p.diagnosis);

    patients[patient_count++] = p;
    printf("Patient added successfully with ID %d.\n", p.id);
}
```

Picture 2. void add_patient

```
void display_patients() {
    if (patient_count == 0) {
        printf("No patients available.\n");
        return;
    }

    for (int i = 0; i < patient_count; i++) {
        printf("\nPatient ID: %d\n", patients[i].id);
        printf("Name: %s\n", patients[i].name);
        printf("Age: %d\n", patients[i].age);
        printf("Gender: %s\n", patients[i].gender);
        printf("Diagnosis: %s\n", patients[i].diagnosis);
    }
}
```

Picture 3. void display_patient

Key Functions:

- add_patient(): Adds a new patient to the system.
- display_patients(): Displays the details of all patients.

2. Appointment Scheduling Module

This module handles scheduling appointments and displaying all appointments.

```
void schedule_appointment() {
    if (appointment_count >= MAX_APPOINTMENTS) {
        printf("Appointment limit reached. Cannot schedule more appointments.\n");
        return;
    }

    Appointment a;
    a.id = appointment_count + 1;

    printf("Enter patient ID: ");
    scanf("%d", &a.patient_id);
    if (!validate_patient_id(a.patient_id)) {
        printf("Invalid patient ID.\n");
        return;
    }

    printf("Enter doctor ID: ");
    scanf("%d", &a.doctor_id);
    if (!validate_doctor_id(a.doctor_id)) {
        printf("Invalid doctor ID.\n");
        return;
    }

    printf("Enter appointment date (YYYY-MM-DD): ");
    scanf(" %[^\n]", a.date);
    if (!validate_date(a.date)) {
        printf("Invalid date format.\n");
        return;
    }
```

```
    printf("Enter appointment time (HH:MM): ");
    scanf(" %[^\n]", a.time);
    if (!validate_time(a.time)) {
        printf("Invalid time format.\n");
        return;
    }

    strcpy(a.status, "Scheduled");
    appointments[appointment_count++] = a;
    printf("Appointment scheduled successfully with ID %d.\n", a.id);
}
```

Picture 4. void schedule_appointment

Picture 5. void display_appointments

Key Functions:

- schedule_appointment(): Schedules a new appointment.

- display_appointments(): Displays the details of all appointments.

3. Doctor Management Module

This module handles adding new doctors and assigning doctors to patients.

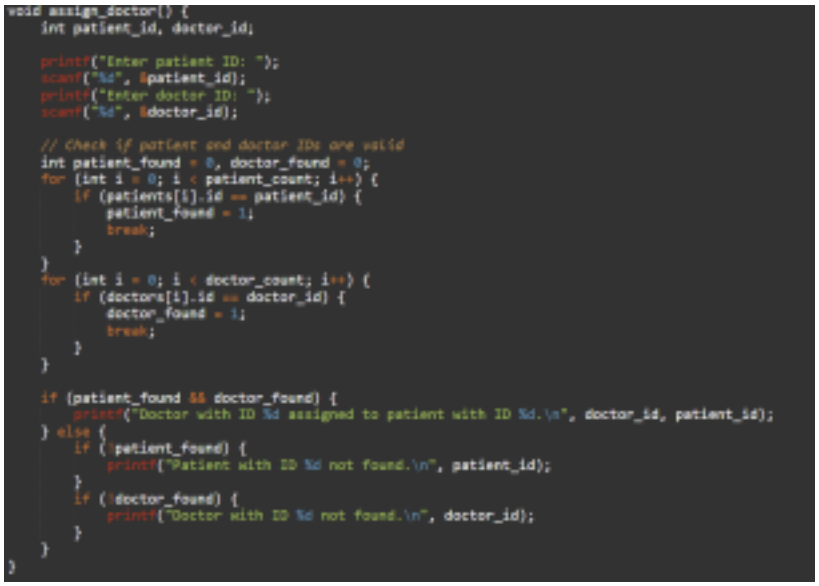
```
void add_doctor() {
    if (doctor_count >= MAX_DOCTORS) {
        printf("Doctor limit reached. Cannot add more doctors.\n");
        return;
    }

    Doctor d;
    d.id = doctor_count + 1;

    printf("Enter doctor name: ");
    scanf(" %[^\n]", d.name);
    printf("Enter doctor specialization: ");
    scanf(" %[^\n]", d.specialization);

    doctors[doctor_count++] = d;
    printf("Doctor added successfully with ID %d.\n", d.id);
}
```

Picture 6. void add_doctor



```
void assign_doctor() {
    int patient_id, doctor_id;

    printf("Enter patient ID: ");
    scanf("%d", &patient_id);
    printf("Enter doctor ID: ");
    scanf("%d", &doctor_id);

    // Check if patient and doctor IDs are valid
    int patient_found = 0, doctor_found = 0;
    for (int i = 0; i < patient_count; i++) {
        if (patients[i].id == patient_id) {
            patient_found = 1;
            break;
        }
    }
    for (int i = 0; i < doctor_count; i++) {
        if (doctors[i].id == doctor_id) {
            doctor_found = 1;
            break;
        }
    }

    if (patient_found && doctor_found) {
        printf("Doctor with ID %d assigned to patient with ID %d.\n", doctor_id, patient_id);
    } else {
        if (!patient_found) {
            printf("Patient with ID %d not found.\n", patient_id);
        }
        if (!doctor_found) {
            printf("Doctor with ID %d not found.\n", doctor_id);
        }
    }
}
```

Picture 7. void assign_doctor

Key Functions:

- add_doctor(): Adds a new doctor to the system.
- assign_doctor(): Assigns a doctor to a patient.

This code now includes the display_appointments function, allowing users to view all scheduled appointments. The menu has been updated to include this option under number 8.

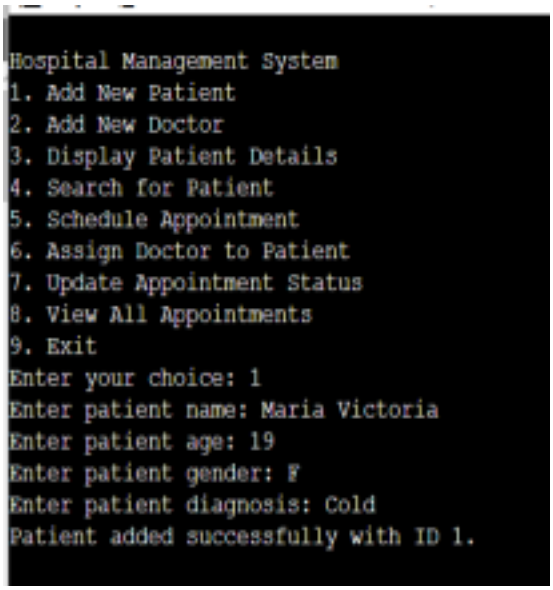
Appendix B

Testing

This section will display a series of screenshots demonstrating the testing and functionality of the Hospital Management System (HMS) software.

Screenshots	Explanation (Demonstration)

<table>
<tr>
<td>



</td>
<td>

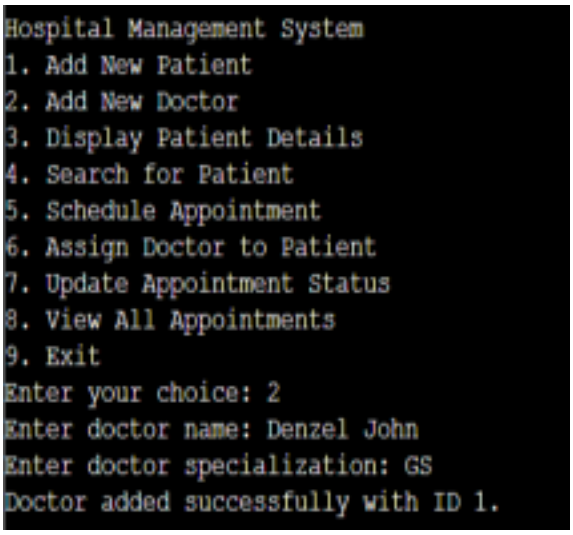
Steps:

1. Run the program
2. Select option 1 from the menu to add a new patient
3. Enter the following details when prompted:
    - Name: Maria Victoria
    - Age: 19
    - Gender: Female
    - Diagnosis: Cold
4. Verify the output to ensure the patient is added successfully with a unique ID.

Expected Result:

- The patient should be added to the system with a unique ID (e.g., 1).
- The system should display a confirmation message indicating the successful addition of the patient.

</td>
</tr>
<tr>
<td>



</td>
<td>

Steps:

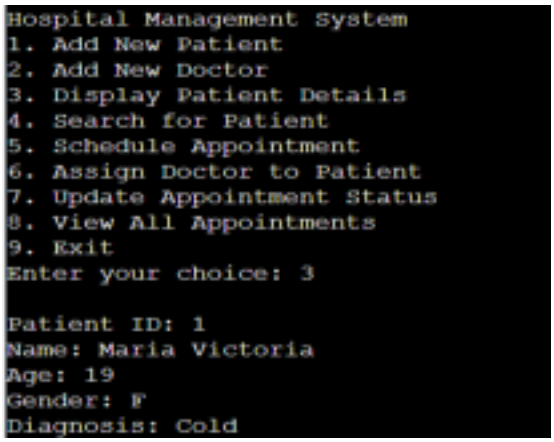
1. Run the program
2. Select option 2 from the menu to add a new doctor
3. Enter the following details when prompted:
    - Name: Dr. Denzel John
    - Specialization: GS
4. Verify the output to ensure the doctor is added successfully with a unique ID.

Expected Result:

- The doctor should be added to the system with a unique ID (e.g., 1).
- The system should display a confirmation message indicating the successful addition of the doctor.

</td>
</tr>
</table>

<table>
<tr>
<td>



</td>
<td>

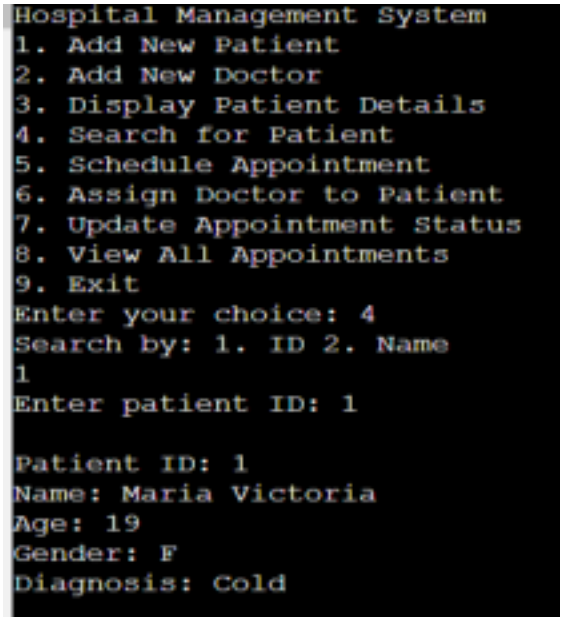
Steps:

1. Run the program
2. Select option 3 from the menu to display patient details.
3. Enter the following when prompted and details will be displayed.
4. Verify the output to ensure the details of the patient.

Expected Result:

- The details of the patient will be displayed.

</td>
</tr>
<tr>
<td>



</td>
<td>

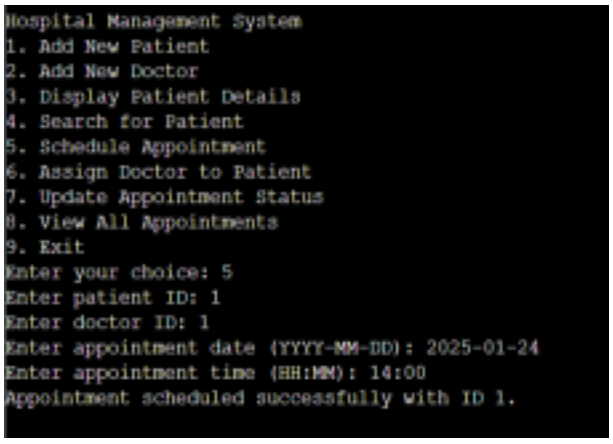
Steps:

1. Run the program
2. Select option 4 from the menu to search for a patient.
3. Enter the following details when prompted:
    - Search by: 1. ID
    - Search by: 2. Name
    - Enter patient ID/Name: 1
4. Verify the output to ensure the details of the patient.

Expected Result:

- The information of the patient will be displayed.

</td>
</tr>
<tr>
<td>



</td>
<td>

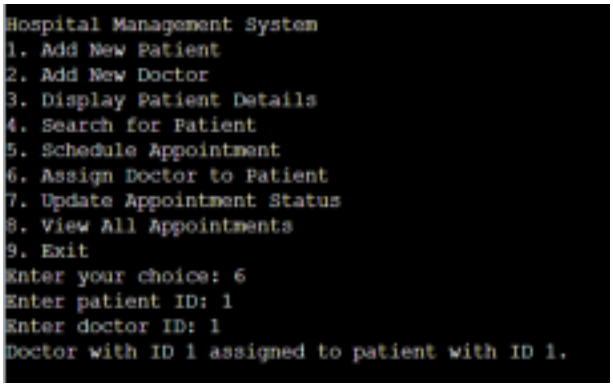
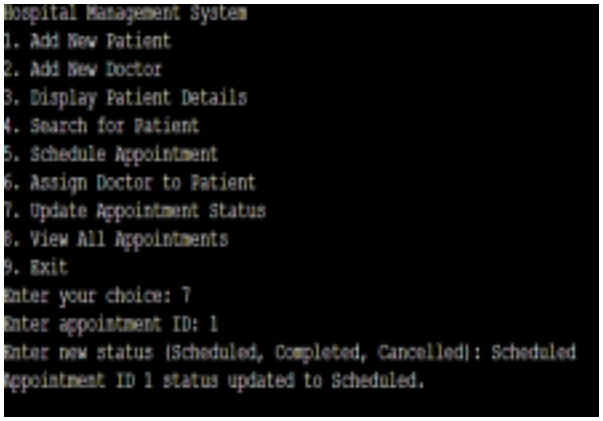
Steps:

1. Run the program
2. Select option 5 from the menu to schedule an appointment.
3. Enter the following details when prompted:
    - Patient ID: 1
    - Doctor ID: 1
    - Appointment Date: 2025-01-24
    - Appointment Time: 14:00
5. Verify the output to ensure the appointment is scheduled successfully with a unique ID.

Expected Result:

- The appointment should be scheduled

</td>
</tr>
</table>

	with a unique ID (e.g., 1). ● The system should display a confirmation message indicating the successful scheduling of the appointment.
	**Steps:** 1. Run the program 2. Select option 6 from the menu to search for an assigned doctor to the patient. 3. Enter the following details when prompted: ● Enter patient ID: 1 ● Enter doctor ID: 1 4. Verify the output to ensure the assigned doctor to the patient is added successfully. **Expected Result:** ● The assigned doctor to the patient should display a confirmation message indicating the successful assigning of doctor to the patient.
	**Steps:** 1. Run the program 2. Select option 7 from the menu to search for update appointment status. 3. Enter the following details when prompted: ● Enter appointment ID: 1 ● Enter new status(scheduled, cancelled, completed): scheduled 4. Verify the output to ensure the details of the appointment are updated **Expected Result:** ● The system should display a confirmation message indicating the successful status of appointment of the patient.

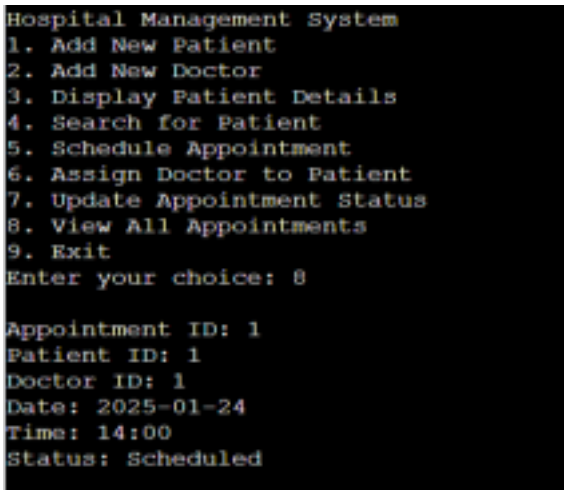
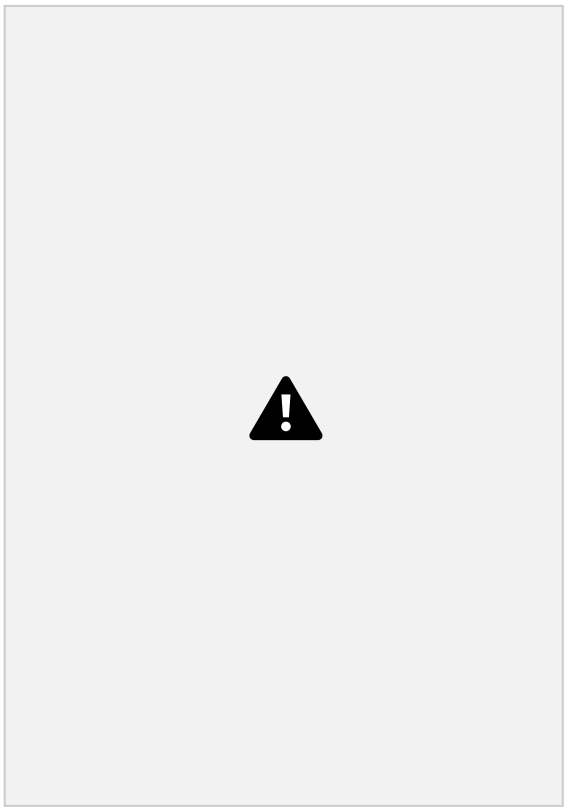
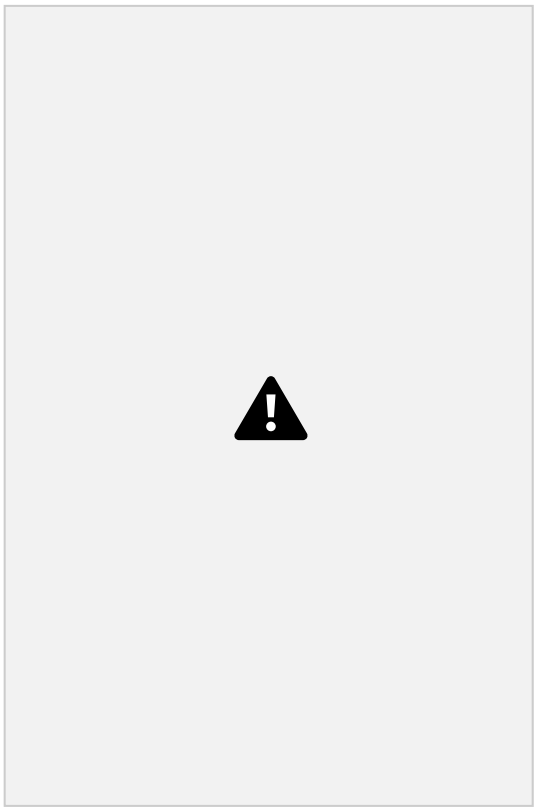
	**Steps:** 1. Run the program 2. Select option 8 from the menu to view all appointments. 3. EEnter the following when prompted and details will be displayed. 4. Verify the output to ensure the details of the patient. **Expected Result:** ● The information of the patient will be displayed.

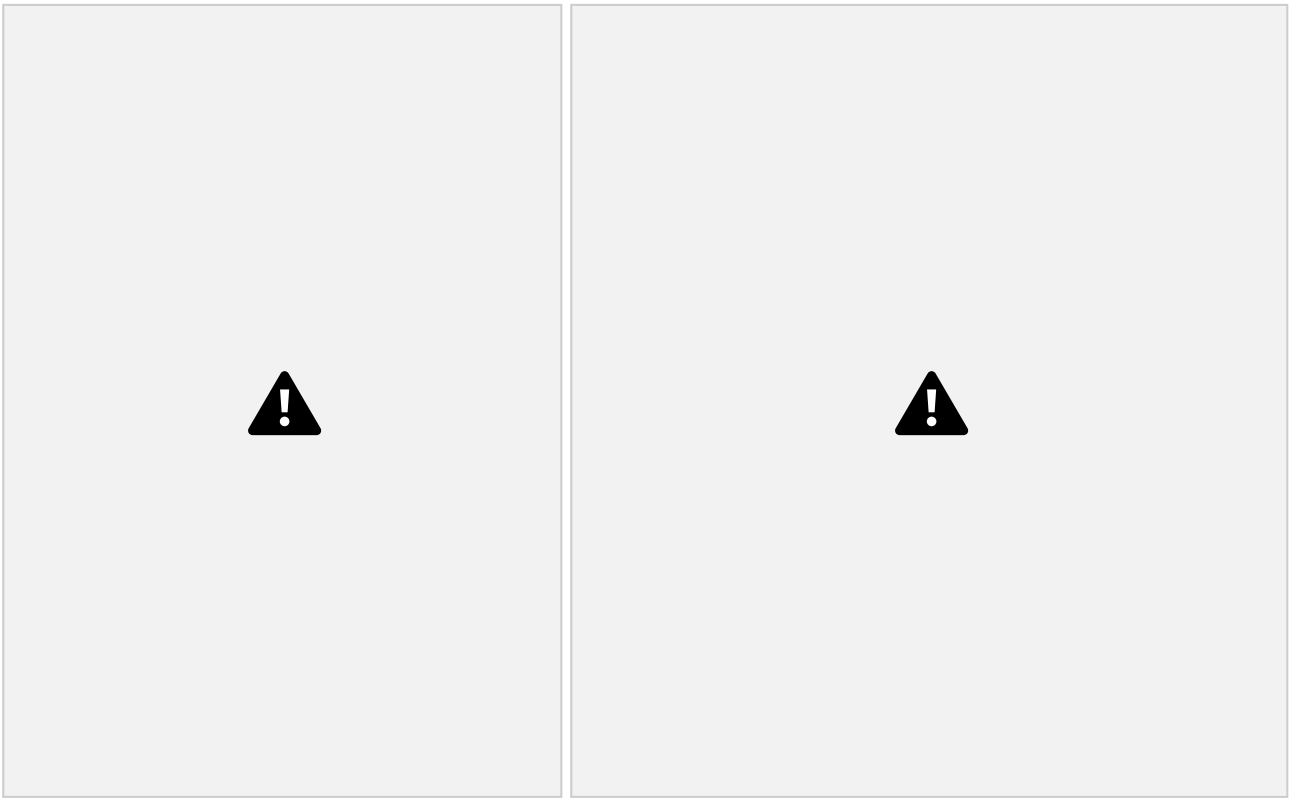
Table 1. Screenshots and Demonstration of HMS

Results

After testing the functionality of the software, the results demonstrate its capability to perform successfully. The picture below illustrates the complete session, showcasing each step:







Picture 8. Screenshot of demonstration of steps

Errors Handling

Effective error handling is crucial for creating robust and user-friendly applications. To achieve this, it is important to use meaningful error messages that provide clear and informative details to help diagnose issues. Logging errors to a file or monitoring system can facilitate debugging and tracking, ensuring that issues can be identified and resolved promptly. Applications should be designed to fail gracefully, meaning they can handle errors without crashing by providing fallback mechanisms or user-friendly error messages. Validating inputs is essential to prevent errors caused by invalid or unexpected data, ensuring the application processes only correct and expected information. Additionally, using custom exceptions to handle specific error scenarios in a more structured way can further enhance the robustness of the application. By following these practices, developers can handle errors effectively and create applications that are both reliable and user-friendly.

Conclusion

The Hospital Management System (HMS) project achieved several significant milestones. The system was successfully developed, incorporating core functionalities such as adding patients and doctors, scheduling appointments, and managing appointment statuses. A user-friendly command-line interface was created, enabling efficient user interaction. Input validation was implemented to ensure that only valid data is processed, thereby enhancing the system's robustness. Additionally, effective error handling mechanisms were put in place to manage unexpected scenarios and provide meaningful feedback to users. These achievements collectively contribute to a reliable and user-friendly hospital management solution. Additionally, managing a large number of patients, doctors, and appointments presented significant challenges in terms of data storage and retrieval. Ensuring that all user inputs were valid and handled correctly required careful design and testing. Another major challenge was designing the system to handle a growing number of records without performance degradation. The development of the Hospital Management System was a valuable learning experience, providing insight into the complexities of managing healthcare data. The project highlighted the importance of thorough planning, effective error handling, and user-friendly design. Moving forward, the suggested improvements and future enhancements will help to create a more robust, scalable, and user-friendly system. This project has laid a solid foundation for further development and potential real-world application in healthcare management.

Recommendation

To future researchers to address these challenges, several improvements and future enhancements are suggested. Integrating the system with a database will allow it to handle larger datasets efficiently and provide persistent storage. Developing a graphical user interface (GUI) will enhance user interaction and make the system more accessible to users with limited technical knowledge. Implementing advanced search and filtering options will enable users to quickly find specific records. Adding functionality for automated notifications will help remind patients of upcoming appointments or notify doctors of schedule changes. Finally, implementing security measures such as user authentication and data encryption will protect sensitive information and enhance the overall security of the system.

Source code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define the maximum number of patients, doctors, and appointments
#define MAX_PATIENTS 100
#define MAX_DOCTORS 50
#define MAX_APPOINTMENTS 200

// Patient structure
typedef struct {
 int id;
 char name[100];
 int age;
 char gender[10];
 char diagnosis[200];
} Patient;

// Doctor structure
typedef struct {
 int id;
 char name[100];
 char specialization[100];
} Doctor;

// Appointment structure
typedef struct {
 int id;
 int patient_id;
 int doctor_id;
 char date[20]; // Format: YYYY-MM-DD
 char time[10]; // Format: HH:MM
 char status[20]; // e.g., Scheduled, Completed, Cancelled
} Appointment;

// Global arrays to store patients, doctors, and appointments
```

```
Patient patients[MAX_PATIENTS];
Doctor doctors[MAX_DOCTORS];
```



```
Appointment appointments[MAX_APPOINTMENTS];

// Counters to keep track of the number of patients, doctors, and appointments
int patient_count = 0;
int doctor_count = 0;
int appointment_count = 0;

// Function prototypes
void show_menu();
void add_patient();
void display_patients();
void search_patient();
void schedule_appointment();
void assign_doctor();
void update_appointment_status();
void add_doctor();
void display_appointments();
int validate_patient_id(int id);
int validate_doctor_id(int id);
int validate_date(const char* date);
int validate_time(const char* time);

int main() {
 int choice;

 while (1) {
 show_menu();
 scanf("%d", &choice);

 switch (choice) {
 case 1:
 add_patient();
 break;
 case 2:
 add_doctor();
 break;
```

```
 case 3:
 display_patients();
 break;
 case 4:
```



```
 search_patient();
 break;
 case 5:
 schedule_appointment();
 break;
 case 6:
 assign_doctor();
 break;
 case 7:
 update_appointment_status();
 break;
 case 8:
 display_appointments();
 break;
 case 9:
 printf("Exiting...\n");
 exit(0);
 default:
 printf("Invalid choice! Please try again.\n");
 }
 }

 return 0;
}

void show_menu(){
 printf("\nHospital Management System\n");
 printf("1. Add New Patient\n");
 printf("2. Add New Doctor\n");
 printf("3. Display Patient Details\n");
 printf("4. Search for Patient\n");
 printf("5. Schedule Appointment\n");
 printf("6. Assign Doctor to Patient\n");
 printf("7. Update Appointment Status\n");
```

```
 printf("8.View All Appointments\n");
 printf("9. Exit\n");
 printf("Enter your choice: ");
}

void add_patient(){
```



```
 if (patient_count >= MAX_PATIENTS) {
 printf("Patient limit reached. Cannot add more patients.\n");
 return;
 }

 Patient p;
 p.id = patient_count + 1;

 printf("Enter patient name: ");
 scanf(" %[^\n]", p.name);
 printf("Enter patient age: ");
 scanf("%d", &p.age);
 printf("Enter patient gender: ");
 scanf(" %[^\n]", p.gender);
 printf("Enter patient diagnosis: ");
 scanf(" %[^\n]", p.diagnosis);

 patients[patient_count++] = p;
 printf("Patient added successfully with ID %d.\n", p.id);
}

void add_doctor() {
 if (doctor_count >= MAX_DOCTORS) {
 printf("Doctor limit reached. Cannot add more doctors.\n");
 return;
 }

 Doctor d;
 d.id = doctor_count + 1;

 printf("Enter doctor name: ");
 scanf(" %[^\n]", d.name);
```

```
 printf("Enter doctor specialization: ");
 scanf(" %[^\n]", d.specialization);

 doctors[doctor_count++] = d;
 printf("Doctor added successfully with ID %d.\n", d.id);
}

void display_patients() {
```



```
 if (patient_count == 0) {
 printf("No patients available.\n");
 return;
 }

 for (int i = 0; i < patient_count; i++) {
 printf("\nPatient ID: %d\n", patients[i].id);
 printf("Name: %s\n", patients[i].name);
 printf("Age: %d\n", patients[i].age);
 printf("Gender: %s\n", patients[i].gender);
 printf("Diagnosis: %s\n", patients[i].diagnosis);
 }
}

void search_patient() {
 int choice;
 printf("Search by: 1. ID 2. Name\n");
 scanf("%d", &choice);

 if (choice == 1) {
 int id;
 printf("Enter patient ID: ");
 scanf("%d", &id);

 for (int i = 0; i < patient_count; i++) {
 if (patients[i].id == id) {
 printf("\nPatient ID: %d\n", patients[i].id);
 printf("Name: %s\n", patients[i].name);
 printf("Age: %d\n", patients[i].age);
 printf("Gender: %s\n", patients[i].gender);
```

```
 printf("Diagnosis: %s\n", patients[i].diagnosis);
 return;
 }
 }
 printf("Patient with ID %d not found.\n", id);

 } else if (choice == 2) {
 char name[100];
 printf("Enter patient name: ");
 scanf(" %[^\n]", name);
```



```
 for (int i = 0; i < patient_count; i++) {
 if (strcmp(patients[i].name, name) == 0) {
 printf("\nPatient ID: %d\n", patients[i].id);
 printf("Name: %s\n", patients[i].name);
 printf("Age: %d\n", patients[i].age);
 printf("Gender: %s\n", patients[i].gender);
 printf("Diagnosis: %s\n", patients[i].diagnosis);
 return;
 }
 }
 printf("Patient with name %s not found.\n", name);

 } else {
 printf("Invalid choice.\n");
 }
}

void schedule_appointment() {
 if (appointment_count >= MAX_APPOINTMENTS) {
 printf("Appointment limit reached. Cannot schedule more appointments.\n");
return;
 }

 Appointment a;
 a.id = appointment_count + 1;

 printf("Enter patient ID: ");
 scanf("%d", &a.patient_id);
```

```
 if (!validate_patient_id(a.patient_id)) {
 printf("Invalid patient ID.\n");
 return;
 }

 printf("Enter doctor ID: ");
 scanf("%d", &a.doctor_id);
 if (!validate_doctor_id(a.doctor_id)) {
 printf("Invalid doctor ID.\n");
 return;
 }
```



```
 printf("Enter appointment date (YYYY-MM-DD): ");
 scanf(" %[^\n]", a.date);
 if (!validate_date(a.date)) {
 printf("Invalid date format.\n");
 return;
 }

 printf("Enter appointment time (HH:MM): ");
 scanf(" %[^\n]", a.time);
 if (!validate_time(a.time)) {
 printf("Invalid time format.\n");
 return;
 }

 strcpy(a.status, "Scheduled");
 appointments[appointment_count++] = a;
 printf("Appointment scheduled successfully with ID %d.\n", a.id);
}

void assign_doctor() {
 int patient_id, doctor_id;

 printf("Enter patient ID: ");
 scanf("%d", &patient_id);
 printf("Enter doctor ID: ");
 scanf("%d", &doctor_id);
```

```
// Check if patient and doctor IDs are valid
int patient_found = 0, doctor_found = 0;
for (int i = 0; i < patient_count; i++) {
if (patients[i].id == patient_id) {
patient_found = 1;
break;
}
}
for (int i = 0; i < doctor_count; i++) {
if (doctors[i].id == doctor_id) {
doctor_found = 1;
break;
```



```
}
}

if (patient_found && doctor_found) {
printf("Doctor with ID %d assigned to patient with ID %d.\n", doctor_id, patient_id);  }
else {
if (!patient_found) {
printf("Patient with ID %d not found.\n", patient_id);
}
if (!doctor_found) {
printf("Doctor with ID %d not found.\n", doctor_id);
}
}
}

void update_appointment_status() {
int appointment_id;
char new_status[20];

printf("Enter appointment ID: ");
scanf("%d", &appointment_id);
printf("Enter new status (Scheduled, Completed, Cancelled): ");
scanf(" %[^\n]", new_status);

for (int i = 0; i < appointment_count; i++) {
```

```
 if (appointments[i].id == appointment_id) {
 strcpy(appointments[i].status, new_status);
 printf("Appointment ID %d status updated to %s.\n", appointment_id, new_status);
return;
 }
 }
 printf("Appointment with ID %d not found.\n", appointment_id);
}

void display_appointments() {
 if (appointment_count == 0) {
 printf("No appointments available.\n");
 return;
 }
```



```
 for (int i = 0; i < appointment_count; i++) {
 printf("\nAppointment ID: %d\n", appointments[i].id);
 printf("Patient ID: %d\n", appointments[i].patient_id);
 printf("Doctor ID: %d\n", appointments[i].doctor_id);
 printf("Date: %s\n", appointments[i].date);
 printf("Time: %s\n", appointments[i].time);
 printf("Status: %s\n", appointments[i].status);
 }
}

int validate_patient_id(int id) {
 for (int i = 0; i < patient_count; i++) {
 if (patients[i].id == id) {
 return 1; // Valid patient ID
 }
 }
 return 0; // Invalid patient ID
}

int validate_doctor_id(int id) {
 for (int i = 0; i < doctor_count; i++) {
 if (doctors[i].id == id) {
 return 1; // Valid doctor ID
```

```
 }
 }
 return 0; // Invalid doctor ID
}

int validate_date(const char* date) {
 // Basic date validation (YYYY-MM-DD)
 if (strlen(date) != 10) {
 return 0;
 }
 if (date[4] != '-' || date[7] != '-') {
 return 0;
 }
 // Additional checks can be added here
 return 1;
}
```



```
int validate_time(const char* time) {
 // Basic time validation (HH:MM)
 if (strlen(time) != 5) {
 return 0;
 }
 if (time[2] != ':') {
 return 0;
 }
 // Additional checks can be added here
 return 1;
}
```

References

Bisht, E., Rathi, A., Chaudhary, M., & Hasti, A. (2021). Hospital Management System. In MATA SUNDRI COLLEGE FOR WOMEN & Department Of Computer Science, Delhi University, *B.Sc. (H) Computer Science* [Thesis]. https://mscw.ac.in/NAAC/Criteria1/Samples-of-ProjectWork_Fieldwork/Computer_Scien

ce/software_Engineering/Software%20Engineering/Hospital%20Managment%20Project%20SE-converted.pdf

Nym. (2022, August 5). *Hospital Management System Project Report PDF*. Itsourcecode.com. https://itsourcecode.com/fyp/hospital-management-system-project-report-documentations-pdf/

Viar. (n.d.). *GitHub - Viar09/Hospital-management-system-in-C-program: Hospital management system in C language.* GitHub. https://github.com/Viar09/Hospital-management-system-in-C-program