UPES

UNIVERSITY OF TOMORROW

Healthcare Record Automation System

(Using C Language)

Name: Diya Singhal
SAP ID: 590022673
Course: B.Tech CSE
Batch: 29

Submitted to: Mr. Joginder Singh
Semester: 1
Institute: SOCS UPES
Academic Year: 2025-26

CERTIFICATE

This is to certify that the project titled: “Healthcare Record Automation System (Using C Language)”

has been successfully completed and submitted by **Diya Singhal of B.Tech CSE – 1st Year**
SAP ID - 590022673 as part of the C Programming Lab / Course Project for the academic year 2025–2026.

This project has been carried out under the guidance of: **Mr. Joginder Singh** Faculty, Department of Computer Science & Engineering. The work presented in this report is genuine and completed by the student as a part of the curriculum requirement.

SIGNATURE
Mr. Joginder Singh
Faculty Guide

School of Computer Science
University of Petroleum and Energy Studies, Dehradun

ACKNOWLEDGEMENT

I wish to express my deep sense of gratitude to **Mr. Joginder Singh**, Faculty, Department of Computer Science & Engineering, for his continuous guidance, valuable suggestions, and encouragement throughout the development of this project titled **“Healthcare Record Automation System (Using C Language)”**. His support has been instrumental in enhancing my understanding of C programming and its practical implementation.

I am also grateful to the **Department of Computer Science & Engineering** and the **University of Petroleum and Energy Studies** for providing the required academic environment and resources essential for completing this project successfully.

I extend my sincere thanks to my peers and family members for their cooperation, motivation, and constant support during the course of this work.

Diya Singhal
B.Tech CSE – 1st Year
SAP ID: **590022673**

ABSTRACT

The Hospital Management System is a console-based application developed using the C programming language with effective implementation of file handling techniques. The objective of this project is to automate and streamline the management of hospital records including patient information, doctor details, and billing data in an efficient and organized manner. The system demonstrates the **practical implementation of structured programming concepts** through real-world problem solving.

Key features of the system include automated ID generation, separate modules for patient and doctor management, individual and master file storage, billing computation based on treatment, medicine, and room charges, search and filter operations, and a statistical dashboard for data analysis. Special emphasis is placed on **modular design, data integrity, and efficient file-based data management**.

The project highlights the practical use of C language concepts such as structures, file handling, conditional logic, and modular programming. Overall, the system reduces manual effort, improves accuracy, and serves as a strong foundation for understanding real-time application development, with scope for future enhancements such as database integration and graphical user interfaces.

List Of Abbreviations

Sr. No.	**Abbreviation**	**Full Form**
1	OPD	Out Patient Department
2	IPD	In Patient Department
3	ICU	Intensive Care Unit
4	ID	Identification Number
5	GUI	Graphical User Interface
6	DBMS	DataBase Management System
7	CLI	Command Line Interface
8	HMS	Hospital Management System
9	API	Application Programming Interface

TABLE OF CONTENTS

PROBLEM DESCRIPTION

Managing patient and doctor information in many small healthcare settings is still largely done manually using registers, loose sheets, or basic spreadsheets. This manual approach often results in misplaced records, difficulty in searching patient details, delays in updating information, and a lack of organized data classification. Such challenges affect the accuracy and availability of essential information, ultimately impacting decision-making and overall operational efficiency.

Healthcare facilities that rely on handwritten records or basic registers also face issues such as inconsistent updates, limited accessibility, and the absence of a centralized system that supports quick addition, retrieval, or modification of data. As the volume of records grows, searching for a specific patient or doctor's details becomes increasingly difficult. Tasks such as classifying patients based on income or categorizing doctors become time-consuming and inefficient without a structured process.

These limitations highlight the need for a simple, computer-based system that can store, manage, and organize healthcare data in a clear and systematic manner. By minimizing manual effort, reducing errors, and enabling smooth access to information, such a system can significantly improve the reliability and efficiency of healthcare record management. To address these challenges, this project proposes an automated *Healthcare Record Automation System*, developed using the C programming language, to maintain separate as well as combined files for patients and doctors while ensuring fast retrieval, easy updates, and structured data handling.

PROBLEM DEFINITION

In many small hospitals and clinics, patient and doctor records are still maintained manually using registers or basic spreadsheets. This manual handling often leads to misplaced files, difficulty in searching information, delayed updates, and poor data organization. As the number of records grows, retrieving or modifying information becomes time-consuming and inefficient. Lack of proper classification further complicates tasks such as identifying patient categories or sorting doctor details. These issues reduce accuracy, increase human error, and slow down overall workflow. Therefore, there is a need for a simple, automated system that can store, manage, and update healthcare records in a structured and reliable manner.

OBJECTIVES OF THE PROJECT

1. To develop an automated system that efficiently stores, retrieves, and manages patient and doctor records using the C programming language.
2. To reduce manual errors and paperwork by replacing handwritten or spreadsheet-based record handling with a structured digital approach.
3. To enable quick access and easy updating of healthcare data, including adding new records, searching existing ones, and modifying information.
4. To organize and classify records—such as categorizing patients based on income and maintaining separate as well as combined files for doctors and patients.

SCOPE OF THE PROJECT

The scope of this project includes designing and developing a basic yet functional automated system for managing healthcare records using the C programming language. It covers the creation, storage, retrieval, updating, and deletion of both patient and doctor information. The system also supports classification of patients based on income levels and maintains separate as well as combined files for organized and structured data access. This project is intended for small clinics, healthcare centers, or academic learning environments where a simple file-based system can improve record-handling efficiency.

Additionally, the project focuses on providing a user-friendly, menu-driven interface that allows even non-technical staff to operate the system with ease. It aims to demonstrate how fundamental programming and file-handling concepts can be applied to solve real-world problems. However, the scope does not include advanced functionalities such as database connectivity, security encryption, networking, or multi-user access.

METHODOLOGY

The development of the *Healthcare Record Automation System (using C language)* was carried out through a structured and systematic process. Every step—from problem analysis to final testing—was carefully executed to ensure clarity, accuracy, and proper functioning of the system. The complete methodology is as follows:

1. **Problem Identification:**
   The challenges in manual healthcare record management were studied, including lost records, difficulty in retrieval, slow updates, and lack of classification.

2. **Requirement Gathering:**
   The need for operations like add, display, search, update, delete, and classify was identified. Requirements for maintaining separate files for doctors and patients were finalized.

3. **Literature Review (Basic Research):**
   Reference materials and basic examples of file handling in C were studied to understand how data can be saved, updated, and retrieved from text files.

4. **Feasibility Analysis:**
   Checked whether the project could be implemented using only C language without any external databases. Ensured that all required features could be developed with simple text files.

5. **Planning the Program Structure:**
   A menu-driven approach was chosen for easy navigation. Decisions were made about the number of functions, their responsibilities, input formats, and file structure.

6. **Designing Flowcharts:**
   Flowcharts for operations like adding data, searching records, updating files, and deleting entries were drawn to visualize program flow.

7. **Writing Pseudocode:**
   High-level pseudocode was created for each module to simplify the coding process and avoid logical mistakes.

8. **Defining Data Structures:**
   Structured formats (using struct) were planned for storing patient and doctor details in a consistent format.

9. **File Structure Design:**
   Decided how data will be saved in text files, how records will be separated, and how combined data will be generated.

10. **Module-by-Module Coding:**
    Each feature—add, display, search, update, delete, classify—was developed in separate functions to maintain clarity.

11. **User Input Handling:**
    Inputs were validated to avoid errors like blank entries, wrong IDs, or incorrect formats.

12. **File Handling Implementation:**
    Used file operations such as fopen, fprintf, fscanf, fseek, and temporary files for update/delete processes.

13. **Integration of All Modules:**
    All functions were merged into a single program with a unified main menu and sub-menus.

14. **Debugging and Error Fixing:**
    Logical errors, infinite loops, incorrect file reads, and missing newline issues were identified and corrected.

15. **Testing With Sample Data:**
    The system was tested with multiple dummy patient and doctor records to ensure that all operations worked correctly.

16. **Boundary & Error Case Testing:**
    Cases like empty files, invalid IDs, partial data, or no matches were tested to check system stability.

17. **Output Verification:**
    Ensured that the outputs generated—display lists, classifications, and combined files—were accurate and well formatted.

18. **Final Optimization:**
    Unnecessary lines of code were removed, indentation was improved, and the program structure was polished for clarity.

19. **Documentation Creation:**
    All findings, results, screenshots, design diagrams, and logic explanations were added to the final project report.

20. **Review & Finalization:**
    The completed project was reviewed to ensure it met the objectives and requirements defined at the beginning.

ALGORITHM

1. **Start**
2. **Initialise the system**

   2.1 Set the initial ID and Doctor ID counters

   2.2 Read the patient master file to determine the next available Patient ID

   2.3 Read the doctor master file to determine the next available Doctor ID
3. **Display the Main Menu with the following options:**
   - Patient Module
   - Doctor Module
   - Billing System
   - Search and Filters
   - Statistics Dashboard
   - Exit
4. **Read the user's main menu choice.**
5. **If the user selects Patient Module, then:**

   5.1 Display the Patient Menu

   5.2 Allow the user to add, display, search, update, delete, classify patients, assign doctors, or manage treatment history

   5.3 Perform selected patient operation using file handling

   5.4 Return to the Patient Menu until the user chooses to go back
6. **If the user selects Doctor Module, then:**

   6.1 Display the Doctor Menu

   6.2 Allow the user to add, display, search, update, delete, or classify doctors

   6.3 Perform selected doctor operation using file handling

   6.4 Return to the Doctor Menu until the user chooses to go back
7. **If the user selects Billing System, then:**

   7.1 Read the Patient ID

   7.2 Accept room, medicine, and treatment charges

   7.3 Apply discount and calculate the total bill amount

   7.4 Generate a bill file and store billing details
8. **If the user selects Search and Filters, then:**

   8.1 Provide options to search patients by disease, age, or income

   8.2 Provide options to search doctors by specialization

   8.3 Display the matching results
9. **If the user selects Statistics Dashboard, then:**

   9.1 Count the total number of patients

   9.2 Count the total number of doctors

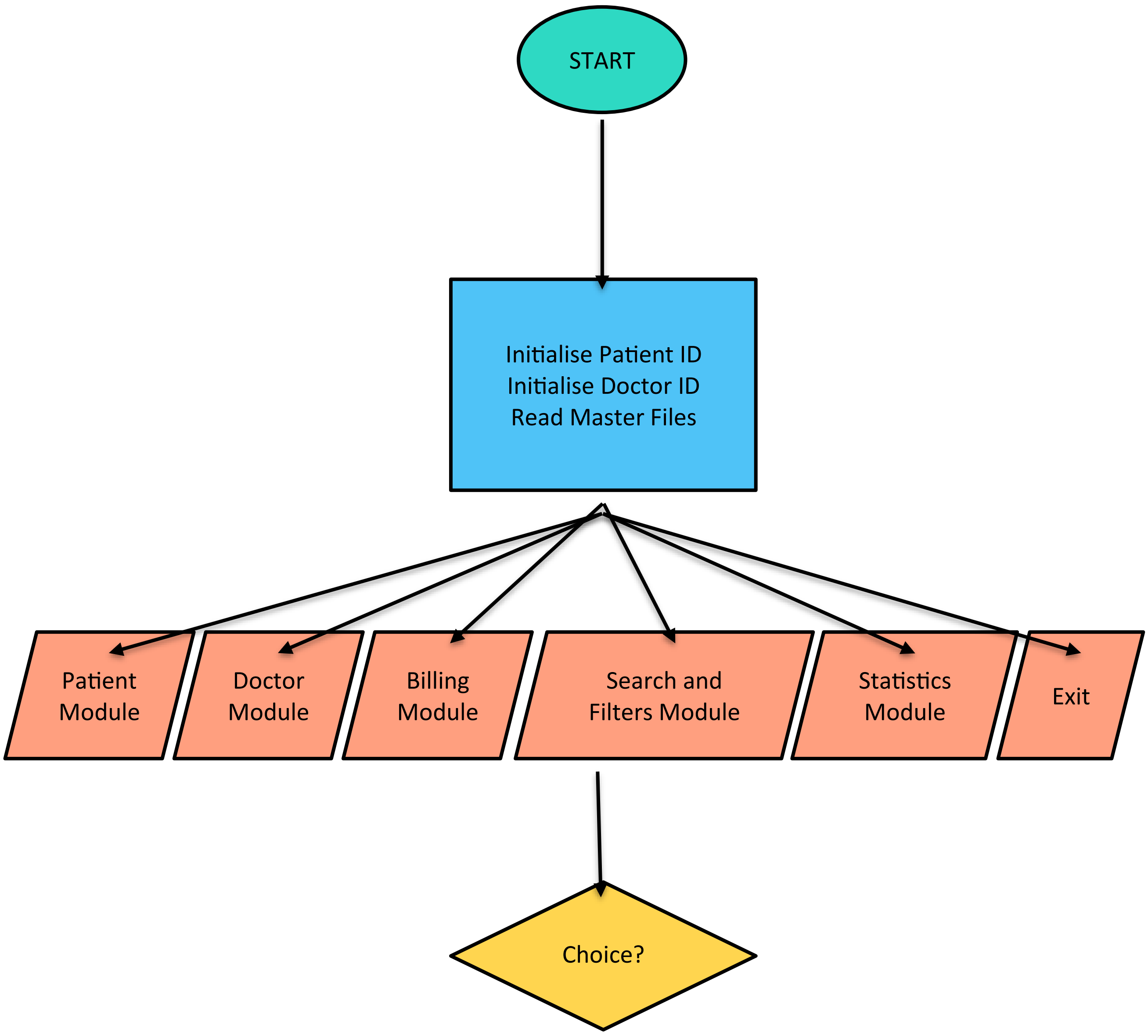
   9.3 Identify the most common disease

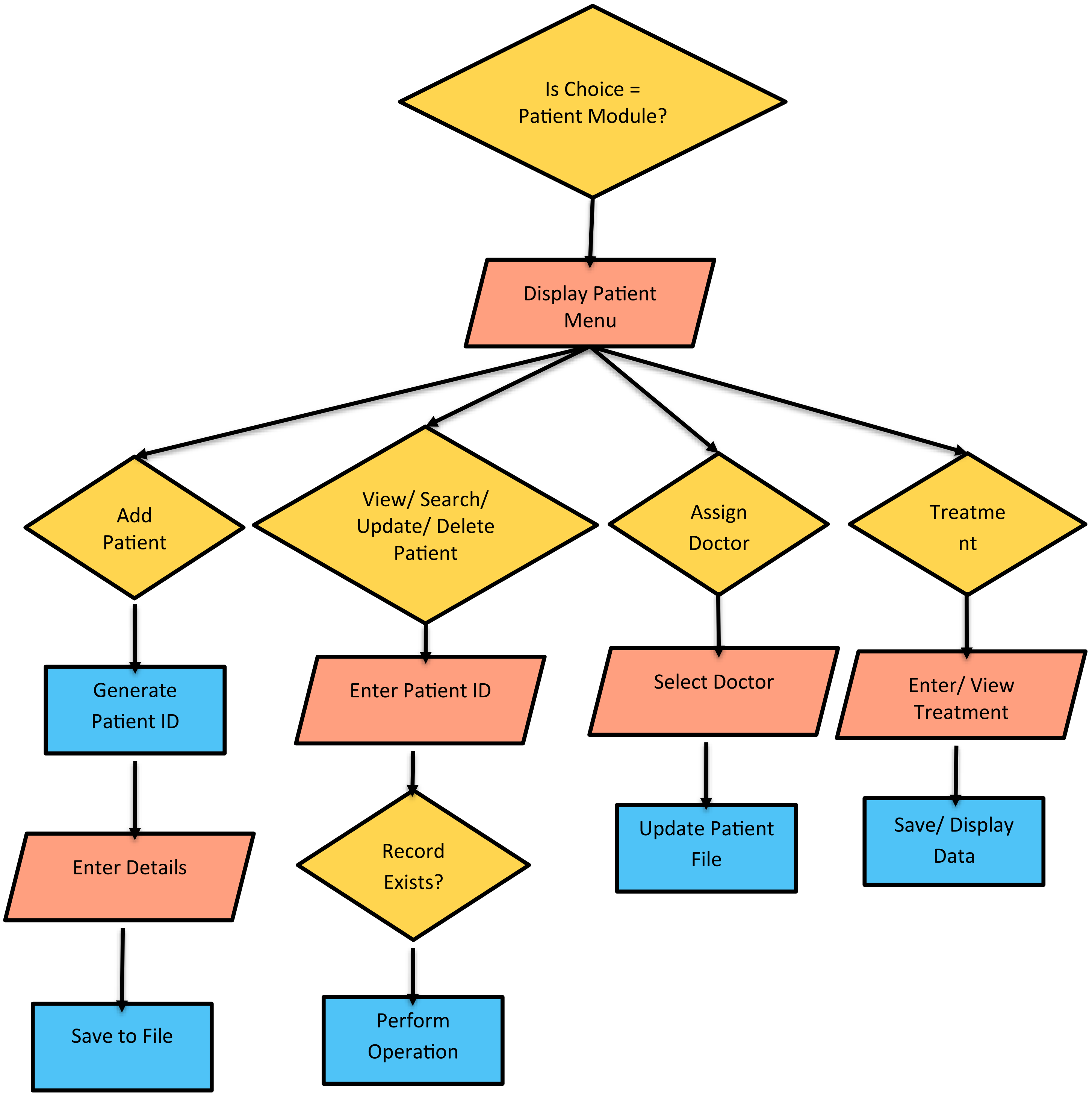
   9.4 Display patient distribution by income category
10. **If the user selects Exit, then:**
11. **Repeat Steps 3 to 10 until Exit option is chosen.**
12. **Stop**

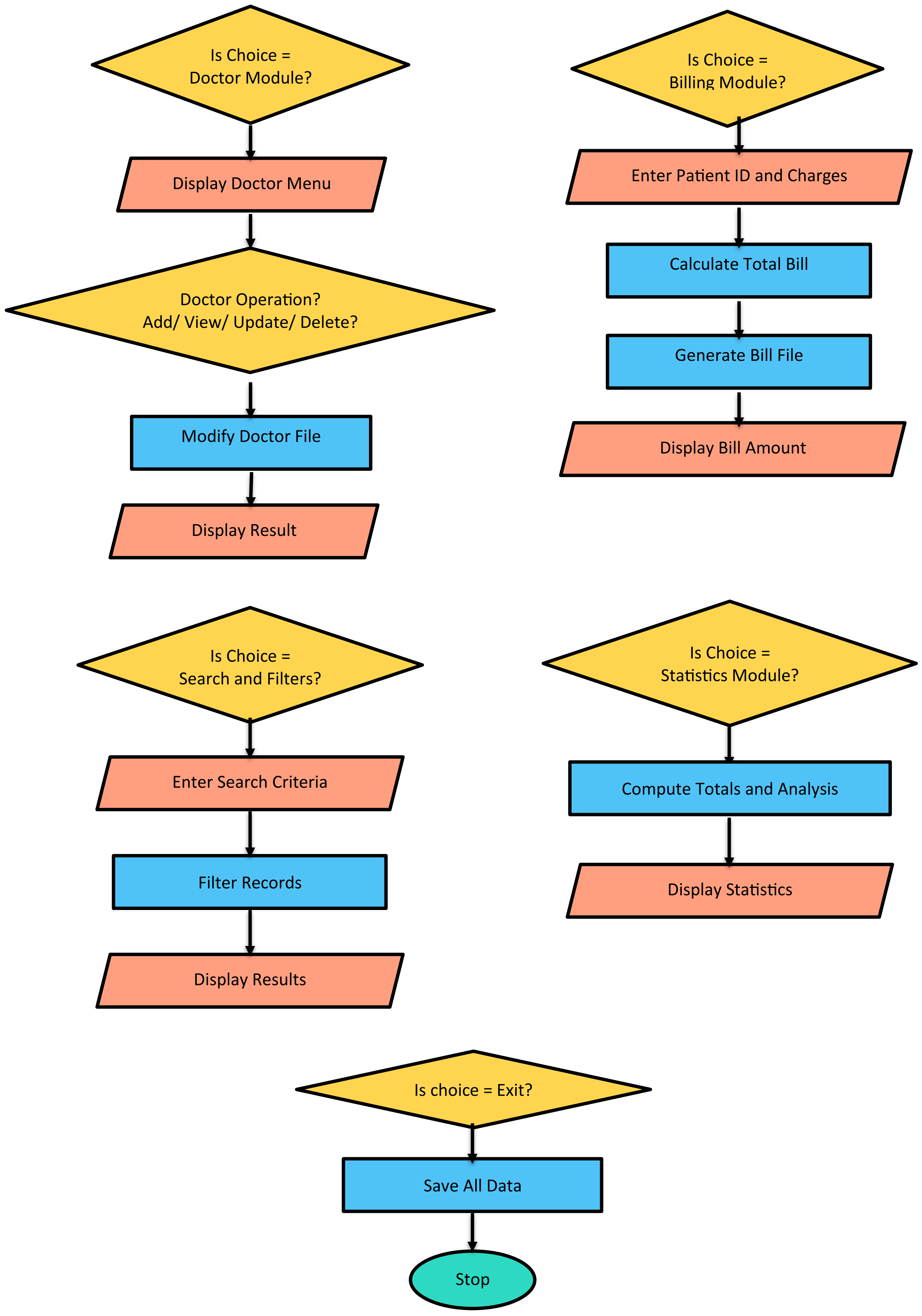
FLOWCHART

SYMBOLS USED:

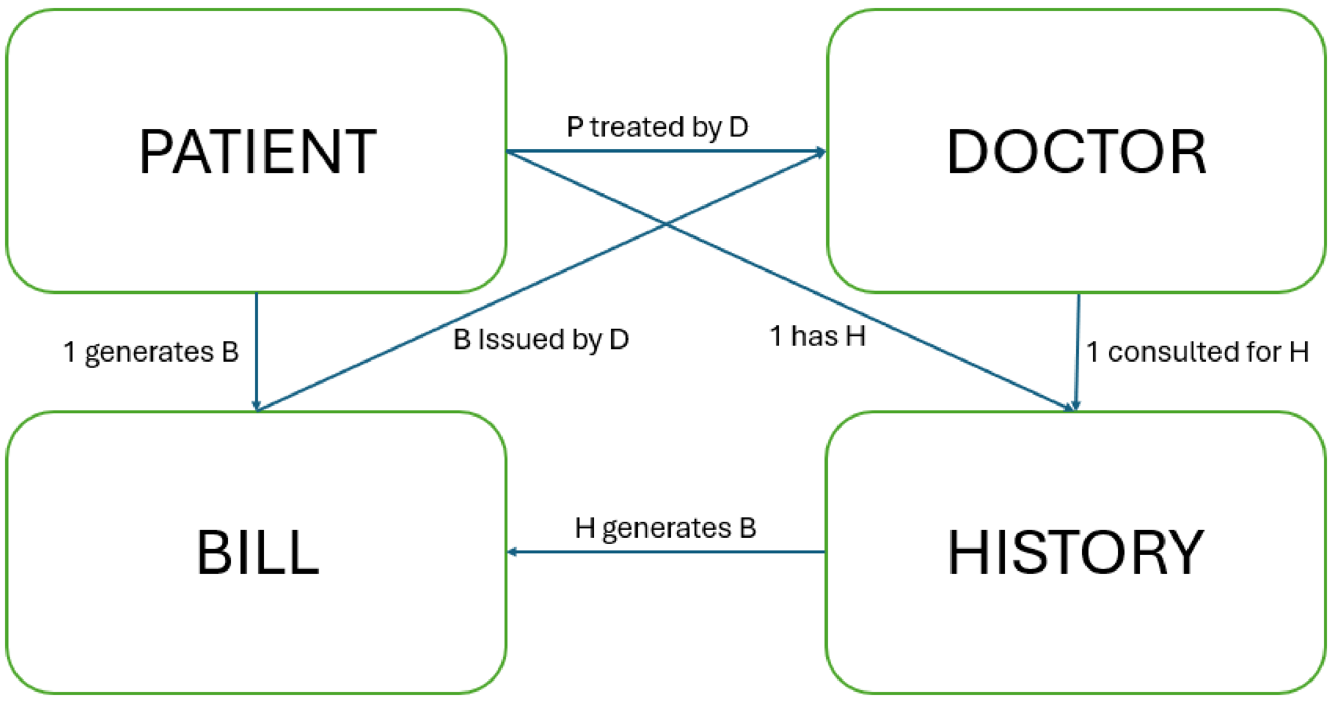
1. Oval: Start/ Stop ⬭
2. Rectangle: Processing ▭
3. Diamond: Decision ◆
4. Parallelogram: Input/ Output ▱
5. Arrow: Flow Direction →




Is Choice =
Patient Module?
Display Patient
Menu
Add
Patient
View/ Search/
Update/ Delete
Patient
Assign
Doctor
Treatme
nt
Generate
Patient ID
Enter Patient ID
Select Doctor
Enter/ View
Treatment
Enter Details
Record
Exists?
Update Patient
File
Save/ Display
Data
Save to File
Perform
Operation


Is Choice = Doctor Module?
Display Doctor Menu
Doctor Operation? Add/ View/ Update/ Delete?
Modify Doctor File
Display Result
Is Choice = Billing Module?
Enter Patient ID and Charges
Calculate Total Bill
Generate Bill File
Display Bill Amount
Is Choice = Search and Filters?
Enter Search Criteria
Filter Records
Display Results
Is Choice = Statistics Module?
Compute Totals and Analysis
Display Statistics
Is choice = Exit?
Save All Data
Stop

ENTITY - RELATIONSHIP DIAGRAM



Here,

P denotes multiple number of patients

D denotes multiple number of Doctors

B denotes multiple number of Bills

H denotes multiple number of Histories

PSEUDO CODE

```
START

// ---------- System Initialization ----------
SET nextPatientID ← 1
SET nextDoctorID ← 1

IF Patient Master File exists THEN
    READ each record
    FIND maximum patient ID
    SET nextPatientID ← maximum ID + 1
END IF

IF Doctor Master File exists THEN
    READ each record
    FIND maximum doctor ID
    SET nextDoctorID ← maximum ID + 1
END IF


// ---------- Main Menu ----------
REPEAT

    DISPLAY Main Menu
        1. Patient Module
        2. Doctor Module
        3. Billing System
        4. Search and Filters
        5. Statistics Dashboard
        6. Exit

    READ main_choice


    // ---------- Patient Module ----------
    IF main_choice = 1 THEN

        REPEAT
            DISPLAY Patient Menu
                1. Add New Patient
                2. Display Patients
                3. Search Patient by ID
                4. Update Patient
                5. Delete Patient
                6. Classify Patients by Income
                7. Assign Doctor to Patient
                8. Treatment History
                9. Back to Main Menu

            READ patient_choice

            IF patient_choice = 1 THEN
                Generate Patient ID using nextPatientID
                Read patient name, age, disease
                Read patient type (OPD / IPD / Emergency)
                Read admission date and room details
                Read annual family income
                IF income < 200000 THEN
                    SET income_category ← Low
                ELSE IF income < 1000000 THEN
                    SET income_category ← Middle
                ELSE
                    SET income_category ← High
                END IF

                Create patient file
                Store patient details in file
                Add summary record to patient master file
                DISPLAY confirmation message

            ELSE IF patient_choice = 2 THEN
                READ patient master file
                DISPLAY all patient records

            ELSE IF patient_choice = 3 THEN
                READ patient ID
                IF patient file exists THEN
                    DISPLAY patient details
                ELSE
                    DISPLAY "Patient not found"
                END IF

            ELSE IF patient_choice = 4 THEN
                READ patient ID
                IF patient exists THEN
                    READ updated patient details
                    UPDATE patient file
                    UPDATE patient master record
                ELSE
                    DISPLAY "Patient not found"
                END IF

            ELSE IF patient_choice = 5 THEN
                READ patient ID
                DELETE patient file
                DELETE patient history file (if exists)
                REMOVE record from master file
                DISPLAY deletion message

            ELSE IF patient_choice = 6 THEN
                FOR each patient in master file
                    CLASSIFY into Low, Middle or High income group
                    DISPLAY classification
                END FOR

            ELSE IF patient_choice = 7 THEN
                READ patient ID
                DISPLAY list of doctors
                READ doctor ID
                UPDATE patient file with doctor ID
                UPDATE patient master file

            ELSE IF patient_choice = 8 THEN
                DISPLAY Treatment Options
                    1. Add Treatment Entry
                    2. View Treatment History
```

```
            READ treatment_choice

            IF treatment_choice = 1 THEN
                READ treatment description
                GET current date and time
                SAVE entry in patient history file
            ELSE
                DISPLAY patient history file
            END IF

        ELSE IF patient_choice = 9 THEN
            EXIT Patient Menu

        ELSE
            DISPLAY "Invalid option"
        END IF

    UNTIL patient_choice = 9


// ---------- Doctor Module ----------
ELSE IF main_choice = 2 THEN

    REPEAT
        DISPLAY Doctor Menu
            1. Add Doctor
            2. Display Doctors
            3. Search Doctor by ID
            4. Update Doctor
            5. Delete Doctor
            6. Classify Doctors by Specialization
            7. Back to Main Menu

        READ doctor_choice

        IF doctor_choice = 1 THEN
            Generate Doctor ID using nextDoctorID
            Read doctor name, specialization, experience
            Create doctor file
            Add record to doctor master file

        ELSE IF doctor_choice = 2 THEN
            DISPLAY all doctors from master file

        ELSE IF doctor_choice = 3 THEN
            READ doctor ID
            DISPLAY doctor file if exists

        ELSE IF doctor_choice = 4 THEN
            READ doctor ID
            UPDATE doctor details and master record

        ELSE IF doctor_choice = 5 THEN
            READ doctor ID
            DELETE doctor file
            REMOVE entry from master file

        ELSE IF doctor_choice = 6 THEN
            DISPLAY doctors grouped by specialization
        ELSE IF doctor_choice = 7 THEN
            EXIT Doctor Menu

        ELSE
            DISPLAY "Invalid option"
        END IF

    UNTIL doctor_choice = 7


// ---------- Billing System ----------
ELSE IF main_choice = 3 THEN

    READ patient ID
    VERIFY patient exists

    READ room charges
    READ medicine charges
    READ treatment charges
    READ discount percentage

    CALCULATE subtotal
    APPLY discount
    CALCULATE final bill amount

    GENERATE bill file with date and time
    STORE bill summary in bills master file
    DISPLAY total bill amount


// ---------- Search and Filters ----------
ELSE IF main_choice = 4 THEN

    DISPLAY Search Menu
        1. Search Patients by Disease
        2. Search Doctors by Specialization
        3. Search Patients by Age Range
        4. Search Patients by Income Category

    READ search_choice

    PERFORM selected search
    DISPLAY matching results


// ---------- Statistics Dashboard ----------
ELSE IF main_choice = 5 THEN

    COUNT total number of patients
    COUNT total number of doctors

    FIND most common disease
    COUNT patients in each income category

    DISPLAY all statistics


// ---------- Exit ----------
ELSE IF main_choice = 6 THEN
    DISPLAY "Saving and exiting system"
```

```
        EXIT main loop

    ELSE
        DISPLAY "Invalid choice"
    END IF

UNTIL main_choice = 6


STOP
```

TOOLS AND TECHNOLOGIES USED

- **Procedural Programming Approach**
  The project follows a structured and procedural programming methodology using functions for modular design.
- **Menu-Driven Interface Design**
  A user-friendly, menu-driven structure is implemented to navigate between patient, doctor, billing, and search modules.
- **Data Persistence using File System**
  Records are stored permanently in the system using file-based storage, ensuring data availability even after program termination.
- **Unique ID Generation Logic**
  Automatic generation of unique patient and doctor IDs using system-initialization logic.
- **Temporary Files Handling**
  Temporary files are used during update and delete operations to ensure data integrity.
- **String Manipulation Techniques**
  Used for parsing, formatting, and processing records stored in text files.
- **Date and Time Functions**
  System date and time are captured for treatment history and billing records using time-related functions.
- **Input Validation using Standard Functions**
  Functions like atoi() and atof() are used to safely convert user input into numeric values.
- **Operating System File System**
  Relies on the underlying OS file system for creating, deleting, renaming, and managing record files.
- **Text-Based Database Simulation**
  Acts as a simple database system by organizing structured information in text files.
- **Cross-Platform Compatibility**
  The program can run on any operating system that supports a standard C compiler.
- **No External Libraries Used**
  Keeps the project lightweight and fully dependent on built-in C libraries.

DATA STRUCTURES USED

1. Overview

This project does not rely heavily on in-memory data structures like struct, linked lists, or dynamic arrays. Instead, it implements a file-based data organization approach, where each text file acts as a persistent data structure.

This design is especially suitable for:

- Beginner-friendly C Projects
- Persistent storage without databases
- Simple record retrieval and modification

2. Logical Data Structures

Although explicit C struct definitions are not used, the organization of data inside files is equivalent to structured data. Each entity follows a fixed and well-defined format.

The system uses three main logical entities:

I. Patient
II. Doctor
III. Bill

Additionally, master files and history files act as indexing and relationship management structures.

3. Patient Data Structure (File-Based)

Logical Structure: Patient

Each patient is stored in an individual text file:

patient_<patientID>.txt

Fields Stored:

- Patient ID (Auto-generated, unique)
- Patient Name
- Age
- Disease
- Patient Type (OPD/ IPD/ Emergency)
- Income Category
- Admission Date
- Room/ Ward Type
- Assigned Doctor ID

Description:

- Each file represents one complete patient record
- Accessed using the Patient ID

- Supports CRUD operations

Advantages:

- Easy to locate a patient by ID
- Reduces chance of data corruption
- Independent records allow safe deletion

4. Patient Master File (Index Structure)

File:

patients_master.txt

Purpose:

- Acts like an index or registry
- Maintains a list of all the patients
- Stores summary information (ID, Name, Disease, etc.)

Characteristics:

- Pipe (|) separated fields
- Read sequentially during listing and statistics
- Helps in fast enumeration of records

Role as a data structure:

- Similar to an array of patient references
- Works like a primary key index

5. Doctor Data Structure (File-Based)

Logical Structure: Doctor

Each doctor is stored in:

- Doctor_<doctorID>.txt

Fields Stored:

- Doctor ID (auto-generated)
- Doctor Name
- Specialisation
- Years of Experience

Description:

- One file = one doctor record
- Enables independent update and deletion
- Used during patient assignment and searching

6. Doctor Master File

File:

- doctors_master.txt

Purpose:

- stores a summarized list of all doctors
- used for display, search and classification

Function:

- acts as a lookup table
- provides specialisation-based filtering

7. Billing Data Structure

Logical Structure: Bill

Billing records are stored as:

bill_<patientID>_<timestamp>.txt

Fields Stored:

- Patient ID
- Room Charges
- Treatment Charges
- Medicine Charges
- Discount Applied
- Total Amount
- Date and Time

Characteristics:

- One bill per visit
- Timestamp avoids filename collision
- Ensures financial record traceability

8. Bill Master File

File:

bills_master.txt

Purpose:

- Stores a summary of all bills generated
- Used for reference and audit

Data Structure Role:

- Sequential record structure
- Supports financial consistency

9. Treatment History Structure

File:

treatment_history_<patientID>.txt

Function:

- Stores chronological history of treatments
- Each new treatment is appended

Benefits:

- Maintains medical timeline
- Prevents overwriting of previous records
- Acts like a log data structure

10.ID Generation mechanism

Variables:

- nextPatientID
- nextDoctorID

Role:

- ensure uniqueness of records
- act like auto-increment counters
- stored and updated during execution

11.Enumerations (Logical Classification Structures)

Although enums may not be explicitly declared, the system logically uses **categorical classification**, such as:

- Patient Type (OPD / IPD / Emergency)
- Income Category (Low / Middle / High)
- Discount Category

These function as:

✓ Enumerated values

✓ Controlled state variables

12.Temporary In-Memory Structures

The program uses:

- Character arrays (char[])
- Integer variables (int)
- Floating-point variables (float)

Usage:

- Input buffering
- File read/write operations
- Calculations and comparisons

13.Data Structure Relationships

ENTITY	RELATIONSHIP
Patient → Doctor	Many-to-One
Patient → Bill	One-to-Many
Patient → History	One-to-Many

14.Advantages of File-Based Data Structures

- No external database required
- Easy debugging
- Persistent storage
- Platform independent
- Suitable for academic evaluation
- Demonstrates understanding of data handling in C

15.Limitations

- Sequential file access is slower for large data
- No concurrent access control
- Manual indexing required

IMPLEMENTATION (MODULE DESCRIPTION)

This Hospital Management System is implemented in C language using structures, enums, file handling, and menu-driven programming. The program is divided into well-defined modules to improve clarity, scalability, and ease of maintenance.

1) **PATIENT MODULE**

   The Patient Module is responsible for managing all patient-related records. Each patient is assigned a unique auto-generated Patient ID. Patient data is stored in two formats:

   - A master patient file containing basic summary information.
   - An individual patient file storing complete details.

   Functions Implemented

   - Add New Patient
   - Display All Patients
   - Search Patient by ID
   - Update Patient Details
   - Delete Patient Record
   - Classify Patients by Income
   - Assign Doctor to Patient
   - Maintain Treatment History

   Working Description

   - When the user selects Add New Patient, the system automatically generates a Patient ID.
   - The user enters patient details such as name, age, disease, patient type (OPD/IPD/Emergency), income category, and admission date.
   - The system displays all available doctors and allows the user to assign a Doctor ID.
   - The patient record is stored in:
     - patients_master.txt
     - patient_<ID>.txt
   - Search, update, and delete operations are performed using Patient ID.
   - Deleted records are removed from both the master file and individual patient file.

2) **DOCTOR MODULE**

   The Doctor Module manages doctor information and ensures doctors can be assigned to patients efficiently. Each doctor is stored using a unique Doctor ID.

   Functions Implemented

   - Add Doctor
   - Display Doctors
   - Search Doctor by ID

- Update Doctor Information
- Delete Doctor Record
- Classify Doctors by Specialization

Working Description

- On adding a doctor, the system auto-generates a Doctor ID.
- The user enters name, specialization, and years of experience.
- Doctor records are saved in:
    - doctors_master.txt
    - doctor_<ID>.txt
- Classification allows filtering doctors based on specialization.
- Deleted doctors are removed from both master and individual files.

3) **BILING SYSTEM**

The Billing System calculates and generates patient bills based on treatment data stored in files.

Functions Implemented

- Generate Bill by Patient ID
- Calculate Charges
- Apply Discounts
- Generate Bill Receipt

Working Description

- The user enters a valid Patient ID
- The system reads the patient's data from file.
- Charges are calculated based on:
    - Room type (ICU/ Private/ General)
    - Treatment Cost
    - Medicine Charges
- Discounts are applied based on income category using enums.
- The final bill is saved in:
  bill_<PatientID>.txt
- The billing receipt is also appended to bills_master.txt

4) **SEARCH AND FILTER MODULE**

This module allows fast searching and filtering of records without altering data.

Functions Implemented

- Search Patients by Disease
- Search Patients by Age Range
- Search by Income Category
- Search Doctors by Specialization

Working Description

- The system reads from master files.
- Matching records are displayed instantly.
- This module improves accessibility and reduces manual searching.

5) STATISTICS DASHBOARD MODULE

The Statistics Module provides analytical insights about hospital data.

Functions Implemented

- Count Total Patients
- Count total Doctors
- Identify Most Common Disease
- Count Patients by Income Category

Working Description

- Uses master files (patients_master.txt, doctors_master.txt).
- Reads all records and generates statistical summaries.
- Helps in understanding patient distribution and hospital workload.

6) FILE HANDLING MECHANISM

The project uses text-based persistent storage

Files Used

- patients_master.txt
- patient_<ID>.txt
- doctors_master.txt
- doctor_<ID>.txt
- bills_master.txt
- bill_<PatientID>.txt

Purpose

- Ensures data persistence
- Allows separate storage for each other
- Easy manual inspection and debugging

7) ID GENERATION LOGIC

- Patient IDs and Doctor IDs are auto-incremented
- IDs are initialized by counting existing records in master files during program startup.
- This ensures:
  - No duplicate IDs
  - Consistent record linking

Why This Implementation Is Effective

- Modular design
- Simple logic with strong structure
- Efficient file handling
- Easy debugging and testing
- Suitable for academic evaluation

SOURCE CODE

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

#define MAXLINE 512
#define MASTER_PATIENTS "patients_master.txt"
#define MASTER_DOCTORS "doctors_master.txt"
#define BILLS_MASTER "bills_master.txt"

int nextPatientID = 1;
int nextDoctorID = 1;

void init_system()
{
    FILE *f;
    char line[MAXLINE];
    f = fopen(MASTER_PATIENTS, "r");
    if(f)
    {
        while(fgets(line, sizeof(line), f))
        {
            int id;
            if(sscanf(line, "%d|", &id) == 1)
            {
                if(id >= nextPatientID) nextPatientID = id + 1;
            }
        }
        fclose(f);
    }
    f = fopen(MASTER_DOCTORS, "r");
    if(f)
    {
        while(fgets(line, sizeof(line), f))
        {
            int id;
            if(sscanf(line, "%d|", &id) == 1)
            {
                if(id >= nextDoctorID) nextDoctorID = id + 1;
            }
        }
        fclose(f);
    }
}

void pause_for_enter()
{
    printf("\nPress Enter to continue...");
    getchar();
}

void add_doctor()
{
    char name[128], spec[128], expS[16];
    int exp;
    int id = nextDoctorID++;
    printf("Enter doctor's name: ");
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = 0;
    printf("Enter specialisation: ");
    fgets(spec, sizeof(spec), stdin);
    spec[strcspn(spec, "\n")] = 0;
    printf("Enter experience (years): ");
    fgets(expS, sizeof(expS), stdin);
    exp = atoi(expS);
    char fname[128];
    sprintf(fname, "doctor_%d.txt", id);
    FILE *f = fopen(fname, "w");
    if(!f) { printf("Error creating doctor file.\n"); return; }
    fprintf(f, "ID:%d\nName:%s\nSpecialisation:%s\nExperience:%d\n", id, name, spec,
exp);
    fclose(f);
    FILE *m = fopen(MASTER_DOCTORS, "a");
    if(m)
    {
        fprintf(m, "%d|%s|%s|%d\n", id, name, spec, exp);
        fclose(m);
    }
    printf("Doctor added with ID %d\n", id);
    pause_for_enter();
}

void display_doctors()
{
    FILE *m = fopen(MASTER_DOCTORS, "r");
    char line[MAXLINE];
    if(!m)
    {
        printf("No doctors found.\n");
        pause_for_enter();
        return;
    }
    printf("ID | Name | Specialisation | Experience\n");
    printf("--------------------------------------------------\n");
    while(fgets(line, sizeof(line), m))
    {
        int id, exp;
        char name[128], spec[128];
        if(sscanf(line, "%d|%127[^|]|%127[^|]|%d", &id, name, spec, &exp) == 4)
        {
            printf("%d | %s | %s | %d\n", id, name, spec, exp);
        }
    }
    fclose(m);
    pause_for_enter();
}

void search_doctor_by_id()
{
    char s[16];
    printf("Enter Doctor ID: ");
    fgets(s, sizeof(s), stdin);
    int id = atoi(s);
    char fname[128];
    sprintf(fname, "doctor_%d.txt", id);
    FILE *f = fopen(fname, "r");
    if(!f)
    {
        printf("Doctor not found.\n");
        pause_for_enter();
        return;
    }
    char line[MAXLINE];
    while(fgets(line, sizeof(line), f))
    {
        printf("%s", line);
    }
    fclose(f);
    pause_for_enter();
}

void update_doctor()
{
    char s[16];
    printf("Enter Doctor ID to update: ");
    fgets(s, sizeof(s), stdin);
    int id = atoi(s);
    char fname[128];
    sprintf(fname, "doctor_%d.txt", id);
    FILE *f = fopen(fname, "r");
    if(!f)
    {
        printf("Doctor not found.\n");
        pause_for_enter();
        return;
    }
    fclose(f);
    char name[128], spec[128], expS[16];
    int exp;
    printf("Enter new name: ");
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = 0;
    printf("Enter new specialisation: ");
    fgets(spec, sizeof(spec), stdin);
    spec[strcspn(spec, "\n")] = 0;
    printf("Enter new experience: ");
    fgets(expS, sizeof(expS), stdin);
    exp = atoi(expS);
    f = fopen(fname, "w");
    if(!f) { printf("Error updating doctor file.\n"); return; }
    fprintf(f, "ID:%d\nName:%s\nSpecialisation:%s\nExperience:%d\n", id, name, spec,
exp);
    fclose(f);
    FILE *m = fopen(MASTER_DOCTORS, "r");
    FILE *tmp = fopen("doctors_tmp.txt", "w");
    char line[MAXLINE];
    if(m && tmp)
    {
        while(fgets(line, sizeof(line), m))
        {
            int rid, rexp;
            char rname[128], rspec[128];
            if(sscanf(line, "%d|%127[^|]|%127[^|]|%d", &rid, rname, rspec, &rexp) ==
4)
            {
                if(rid == id)
                {
                    fprintf(tmp, "%d|%s|%s|%d\n", id, name, spec, exp);
                }
                else
                {
                    fprintf(tmp, "%s", line);
                }
            }
        }
        fclose(m);
        fclose(tmp);
        remove(MASTER_DOCTORS);
        rename("doctors_tmp.txt", MASTER_DOCTORS);
    }
    printf("Doctor updated.\n");
    pause_for_enter();
}

void delete_doctor()
{
    char s[16];
    printf("Enter Doctor ID to delete: ");
    fgets(s, sizeof(s), stdin);
    int id = atoi(s);
    char fname[128];
    sprintf(fname, "doctor_%d.txt", id);
    if(remove(fname) != 0)
    {
        printf("Doctor file not found.\n");
        pause_for_enter();
        return;
    }
    FILE *m = fopen(MASTER_DOCTORS, "r");
    FILE *tmp = fopen("doctors_tmp.txt", "w");
    char line[MAXLINE];
    if(m && tmp)
    {
        while(fgets(line, sizeof(line), m))
        {
            int rid;
            if(sscanf(line, "%d|", &rid) == 1)
            {
                if(rid != id) fprintf(tmp, "%s", line);
            }
        }
        fclose(m);
        fclose(tmp);
        remove(MASTER_DOCTORS);
        rename("doctors_tmp.txt", MASTER_DOCTORS);
    }
    printf("Doctor deleted.\n");
    pause_for_enter();
}

void classify_doctors()
{
    FILE *m = fopen(MASTER_DOCTORS, "r");
    char line[MAXLINE];
    if(!m) { printf("No doctors.\n"); pause_for_enter(); return; }
    printf("Classified by specialisation:\n");
    while(fgets(line, sizeof(line), m))
    {
        int id, exp;
        char name[128], spec[128];
        if(sscanf(line, "%d|%127[^|]|%127[^|]|%d", &id, name, spec, &exp) == 4)
        {
            printf("%s : %s (ID:%d)\n", spec, name, id);
        }
    }
    fclose(m);
    pause_for_enter();
}

void add_patient()
{
    char name[128], ageS[8], disease[128], type[32], incomeS[32], room[64],
admDate[64];
    int age, incomeCat = 0;
    int id = nextPatientID++;
    printf("Enter patient name: ");
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = 0;
    printf("Enter age: ");
    fgets(ageS, sizeof(ageS), stdin);
    age = atoi(ageS);
    printf("Enter disease/condition: ");
    fgets(disease, sizeof(disease), stdin);
    disease[strcspn(disease, "\n")] = 0;
    printf("Enter patient type (OPD/IPD/Emergency): ");
    fgets(type, sizeof(type), stdin);
    type[strcspn(type, "\n")] = 0;
    printf("Enter admission date (YYYY-MM-DD) or NA: ");
    fgets(admDate, sizeof(admDate), stdin);
    admDate[strcspn(admDate, "\n")] = 0;
    printf("Enter room/ward (ICU/Private/General) or NA: ");
    fgets(room, sizeof(room), stdin);
    room[strcspn(room, "\n")] = 0;
    printf("Enter annual family income (number): ");
    fgets(incomeS, sizeof(incomeS), stdin);
    int income = atoi(incomeS);
    if(income < 200000) incomeCat = 1;
    else if(income < 1000000) incomeCat = 2;
    else incomeCat = 3;
    char fname[128];
    sprintf(fname, "patient_%d.txt", id);
    FILE *f = fopen(fname, "w");
    if(!f) { printf("Error creating patient file.\n"); return; }
    fprintf(f,
"ID:%d\nName:%s\nAge:%d\nDisease:%s\nType:%s\nAdmissionDate:%s\nRoom:%s\nIncome:%d\nIn
comeCategory:%d\nDoctorID:%d\n", id, name, age, disease, type, admDate, room, income,
incomeCat, 0);
    fclose(f);
    FILE *m = fopen(MASTER_PATIENTS, "a");
    if(m)
    {
        fprintf(m, "%d|%s|%d|%s|%d|%d|%s\n", id, name, age, disease, income, 0, type);
        fclose(m);
    }
    printf("Patient added with ID %d\n", id);
    pause_for_enter();
}
```

```
void display_patients()
{
    FILE *m = fopen(MASTER_PATIENTS, "r");
    char line[MAXLINE];
    if(!m) { printf("No patients found.\n"); pause_for_enter(); return; }
    printf("ID | Name | Age | Disease | Income | DoctorID | Type\n");
    printf("----------------------------------------------------------\n");
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], disease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
disease, &income, &docid, type) >= 6)
        {
            printf("%d | %s | %d | %s | %d | %d | %s\n", id, name, age, disease,
income, docid, type);
        }
    }
    fclose(m);
    pause_for_enter();
}

void search_patient_by_id()
{
    char s[16];
    printf("Enter Patient ID: ");
    fgets(s, sizeof(s), stdin);
    int id = atoi(s);
    char fname[128];
    sprintf(fname, "patient_%d.txt", id);
    FILE *f = fopen(fname, "r");
    if(!f) { printf("Patient not found.\n"); pause_for_enter(); return; }
    char line[MAXLINE];
    while(fgets(line, sizeof(line), f))
    {
        printf("%s", line);
    }
    fclose(f);
    pause_for_enter();
}

void update_patient()
{
    char s[16];
    printf("Enter Patient ID to update: ");
    fgets(s, sizeof(s), stdin);
    int id = atoi(s);
    char fname[128];
    sprintf(fname, "patient_%d.txt", id);
    FILE *f = fopen(fname, "r");
    if(!f) { printf("Patient not found.\n"); pause_for_enter(); return; }
    fclose(f);
    char name[128], ageS[8], disease[128], type[32], incomeS[32], room[64],
admDate[64];
    int age, income, incomeCat;
    printf("Enter new name: ");
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = 0;
    printf("Enter new age: ");
    fgets(ageS, sizeof(ageS), stdin);
    age = atoi(ageS);
    printf("Enter new disease: ");
    fgets(disease, sizeof(disease), stdin);
    disease[strcspn(disease, "\n")] = 0;
    printf("Enter new type (OPD/IPD/Emergency): ");
    fgets(type, sizeof(type), stdin);
    type[strcspn(type, "\n")] = 0;
    printf("Enter new admission date or NA: ");
    fgets(admDate, sizeof(admDate), stdin);
    admDate[strcspn(admDate, "\n")] = 0;
    printf("Enter new room/ward or NA: ");
    fgets(room, sizeof(room), stdin);
    room[strcspn(room, "\n")] = 0;
    printf("Enter new annual family income: ");
    fgets(incomeS, sizeof(incomeS), stdin);
    income = atoi(incomeS);
    if(income < 200000) incomeCat = 1;
    else if(income < 1000000) incomeCat = 2;
    else incomeCat = 3;
    f = fopen(fname, "w");
    if(!f) { printf("Error updating patient file.\n"); return; }
    fprintf(f,
"ID:%d\nName:%s\nAge:%d\nDisease:%s\nType:%s\nAdmissionDate:%s\nRoom:%s\nIncome:%d\nIn
comeCategory:%d\nDoctorID:%d\n", id, name, age, disease, type, admDate, room, income,
incomeCat, 0);
    fclose(f);
    FILE *m = fopen(MASTER_PATIENTS, "r");
    FILE *tmp = fopen("patients_tmp.txt", "w");
    char line[MAXLINE];
    if(m && tmp)
    {
        while(fgets(line, sizeof(line), m))
        {
            int rid, rage, rincome, rdoc;
            char rname[128], rdisease[128], rtype[64];
            if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &rid, rname,
&rage, rdisease, &rincome, &rdoc, rtype) >= 6)
            {
                if(rid == id)
                {
                    fprintf(tmp, "%d|%s|%d|%s|%d|%d|%s\n", id, name, age, disease,
income, 0, type);
                }
                else
                {
                    fprintf(tmp, "%s", line);
                }
            }
        }
        fclose(m);
        fclose(tmp);
        remove(MASTER_PATIENTS);
        rename("patients_tmp.txt", MASTER_PATIENTS);
    }
    printf("Patient updated.\n");
    pause_for_enter();
}

void delete_patient()
{
    char s[16];
    printf("Enter Patient ID to delete: ");
    fgets(s, sizeof(s), stdin);
    int id = atoi(s);
    char fname[128];
    sprintf(fname, "patient_%d.txt", id);
    if(remove(fname) != 0)
    {
        printf("Patient file not found.\n");
        pause_for_enter();
        return;
    }
    char hist[128];
    sprintf(hist, "patient_%d_history.txt", id);
    remove(hist);
    FILE *m = fopen(MASTER_PATIENTS, "r");
    FILE *tmp = fopen("patients_tmp.txt", "w");
    char line[MAXLINE];
    if(m && tmp)
    {
        while(fgets(line, sizeof(line), m))
        {
            int rid;
            if(sscanf(line, "%d|", &rid) == 1)
            {
                if(rid != id) fprintf(tmp, "%s", line);
            }
        }
        fclose(m);
        fclose(tmp);
        remove(MASTER_PATIENTS);
        rename("patients_tmp.txt", MASTER_PATIENTS);
    }
    printf("Patient deleted.\n");
    pause_for_enter();
}

void classify_patients_by_income()
{
    FILE *m = fopen(MASTER_PATIENTS, "r");
    char line[MAXLINE];
    if(!m) { printf("No patients.\n"); pause_for_enter(); return; }
    printf("Income Category 1 (Low):\n");
    rewind(m);
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], disease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
disease, &income, &docid, type) >= 6)
        {
            if(income < 200000) printf("%d | %s | %d | %s\n", id, name, age, disease);
        }
    }
    printf("\nIncome Category 2 (Middle):\n");
    rewind(m);
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], disease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
disease, &income, &docid, type) >= 6)
        {
            if(income >= 200000 && income < 1000000) printf("%d | %s | %d | %s\n", id,
name, age, disease);
        }
    }
    printf("\nIncome Category 3 (High):\n");
    rewind(m);
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], disease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
disease, &income, &docid, type) >= 6)
        {
            if(income >= 1000000) printf("%d | %s | %d | %s\n", id, name, age,
disease);
        }
    }
    fclose(m);
    pause_for_enter();
}

void assign_doctor_to_patient()
{
    char sp[16];
    printf("Enter Patient ID to assign doctor: ");
    fgets(sp, sizeof(sp), stdin);
    int pid = atoi(sp);
    char pfname[128];
    sprintf(pfname, "patient_%d.txt", pid);
    FILE *pf = fopen(pfname, "r");
    if(!pf) { printf("Patient not found.\n"); pause_for_enter(); return; }
    fclose(pf);
    display_doctors();
    char sd[16];
    printf("Enter Doctor ID to assign: ");
    fgets(sd, sizeof(sd), stdin);
    int did = atoi(sd);
    char dfname[128];
    sprintf(dfname, "doctor_%d.txt", did);
    FILE *df = fopen(dfname, "r");
    if(!df) { printf("Doctor not found.\n"); pause_for_enter(); return; }
    fclose(df);
    char line[MAXLINE];
    FILE *f = fopen(pfname, "r");
    FILE *tmp = fopen("patient_update_tmp.txt", "w");
    if(f && tmp)
    {
        while(fgets(line, sizeof(line), f))
        {
            if(strncmp(line, "DoctorID:", 9) == 0)
            {
                fprintf(tmp, "DoctorID:%d\n", did);
            }
            else fprintf(tmp, "%s", line);
        }
        fclose(f);
        fclose(tmp);
        remove(pfname);
        rename("patient_update_tmp.txt", pfname);
    }
    FILE *m = fopen(MASTER_PATIENTS, "r");
    FILE *mt = fopen("patients_tmp.txt", "w");
    if(m && mt)
    {
        while(fgets(line, sizeof(line), m))
        {
            int id, age, income, docid;
            char name[128], disease[128], type[64];
            if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age, 
disease, &income, &docid, type) >= 6)
            {
                if(id == pid) fprintf(mt, "%d|%s|%d|%s|%d|%d|%s\n", id, name, age, 
disease, income, did, type);
                else fprintf(mt, "%s", line);
            }
        }
        fclose(m);
        fclose(mt);
        remove(MASTER_PATIENTS);
        rename("patients_tmp.txt", MASTER_PATIENTS);
    }
    printf("Doctor %d assigned to patient %d\n", did, pid);
    pause_for_enter();
}

void add_treatment_history()
{
    char sp[16];
    printf("Enter Patient ID to add treatment entry: ");
    fgets(sp, sizeof(sp), stdin);
    int pid = atoi(sp);
    char pfname[128];
    sprintf(pfname, "patient_%d.txt", pid);
    FILE *pf = fopen(pfname, "r");
    if(!pf) { printf("Patient not found.\n"); pause_for_enter(); return; }
    fclose(pf);
    char entry[256];
    printf("Enter treatment entry text: ");
    fgets(entry, sizeof(entry), stdin);
    entry[strcspn(entry, "\n")] = 0;
    char histname[128];
    sprintf(histname, "patient_%d_history.txt", pid);
    FILE *h = fopen(histname, "a");
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);
    fprintf(h, "%04d-%02d-%02d %02d:%02d:%02d | %s\n", tm->tm_year+1900, tm->tm_mon+1, 
tm->tm_mday, tm->tm_hour, tm->tm_min, tm->tm_sec, entry);
    fclose(h);
    printf("Treatment entry added.\n");
    pause_for_enter();
}

void view_treatment_history()
{
    char sp[16];
    printf("Enter Patient ID to view history: ");
    fgets(sp, sizeof(sp), stdin);
    int pid = atoi(sp);
    char histname[128];
    sprintf(histname, "patient_%d_history.txt", pid);
    FILE *h = fopen(histname, "r");
    char line[MAXLINE];
    if(!h) { printf("No history found.\n"); pause_for_enter(); return; }
    while(fgets(line, sizeof(line), h))
    {
        printf("%s", line);
    }
    fclose(h);
    pause_for_enter();
}

void calculate_bill()
{
    char sp[16];
    printf("Enter Patient ID for billing: ");
    fgets(sp, sizeof(sp), stdin);
    int pid = atoi(sp);
    char pfname[128];
    sprintf(pfname, "patient_%d.txt", pid);
    FILE *pf = fopen(pfname, "r");
    if(!pf) { printf("Patient not found.\n"); pause_for_enter(); return; }
    fclose(pf);
    char roomsS[32], medS[32], treatS[32], discS[16];
    double roomC, medC, treatC, discount;
    printf("Enter room charges: ");
    fgets(roomsS, sizeof(roomsS), stdin);
    roomC = atof(roomsS);
    printf("Enter medicines charges: ");
    fgets(medS, sizeof(medS), stdin);
    medC = atof(medS);
    printf("Enter treatment/other charges: ");
    fgets(treatS, sizeof(treatS), stdin);
    treatC = atof(treatS);
    printf("Enter discount percentage (0-100): ");
    fgets(discS, sizeof(discS), stdin);
    discount = atof(discS);
    double subtotal = roomC + medC + treatC;
    double total = subtotal * (1.0 - discount/100.0);
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);
    char billname[128];
    sprintf(billname, "bill_%d_%04d%02d%02d%02d%02d%02d.txt", pid, tm->tm_year+1900,
tm->tm_mon+1, tm->tm_mday, tm->tm_hour, tm->tm_min, tm->tm_sec);
    FILE *b = fopen(billname, "w");
    if(!b) { printf("Error creating bill file.\n"); return; }
    fprintf(b,
"PatientID:%d\nRoomCharges:%.2f\nMedicines:%.2f\nTreatmentCharges:%.2f\nDiscount:%.2f\
nTotal:%.2f\nDate:%04d-%02d-%02d %02d:%02d:%02d\n", pid, roomC, medC, treatC,
discount, total, tm->tm_year+1900, tm->tm_mon+1, tm->tm_mday, tm->tm_hour, tm->tm_min,
tm->tm_sec);
    fclose(b);
    FILE *bm = fopen(BILLS_MASTER, "a");
    if(bm)
    {
        fprintf(bm, "%s|%d|%.2f\n", billname, pid, total);
        fclose(bm);
    }
    printf("Bill generated: %s\nTotal: %.2f\n", billname, total);
    pause_for_enter();
}

void search_patients_by_disease()
{
    char disease[128];
    printf("Enter disease to search: ");
    fgets(disease, sizeof(disease), stdin);
    disease[strcspn(disease, "\n")] = 0;
    FILE *m = fopen(MASTER_PATIENTS, "r");
    char line[MAXLINE];
    int found = 0;
    if(!m) { printf("No patients.\n"); pause_for_enter(); return; }
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], rdisease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
rdisease, &income, &docid, type) >= 6)
        {
            if(strcasecmp(rdisease, disease) == 0)
            {
                printf("%d | %s | %d | %s\n", id, name, age, rdisease);
                found = 1;
            }
        }
    }
    if(!found) printf("No patients with disease %s found.\n", disease);
    fclose(m);
    pause_for_enter();
}

void search_doctors_by_specialisation()
{
    char spec[128];
    printf("Enter specialisation to search: ");
    fgets(spec, sizeof(spec), stdin);
    spec[strcspn(spec, "\n")] = 0;
    FILE *m = fopen(MASTER_DOCTORS, "r");
    char line[MAXLINE];
    int found = 0;
    if(!m) { printf("No doctors.\n"); pause_for_enter(); return; }
    while(fgets(line, sizeof(line), m))
    {
        int id, exp;
        char name[128], rspec[128];
        if(sscanf(line, "%d|%127[^|]|%127[^|]|%d", &id, name, rspec, &exp) == 4)
        {
            if(strcasecmp(rspec, spec) == 0)
            {
                printf("%d | %s | %s | %d\n", id, name, rspec, exp);
                found = 1;
            }
        }
    }
    if(!found) printf("No doctors with specialisation %s found.\n", spec);
    fclose(m);
    pause_for_enter();
}
```

```
void search_by_age_range()
{
    char a1s[16], a2s[16];
    printf("Enter min age: ");
    fgets(a1s, sizeof(a1s), stdin);
    printf("Enter max age: ");
    fgets(a2s, sizeof(a2s), stdin);
    int a1 = atoi(a1s);
    int a2 = atoi(a2s);
    FILE *m = fopen(MASTER_PATIENTS, "r");
    char line[MAXLINE];
    int found = 0;
    if(!m) { printf("No patients.\n"); pause_for_enter(); return; }
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], disease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age, 
disease, &income, &docid, type) >= 6)
        {
            if(age >= a1 && age <= a2)
            {
                printf("%d | %s | %d | %s\n", id, name, age, disease);
                found = 1;
            }
        }
    }
    if(!found) printf("No patients in age range.\n");
    fclose(m);
    pause_for_enter();
}

void search_by_income_category()
{
    char s[8];
    printf("Enter income category (1=Low,2=Middle,3=High): ");
    fgets(s, sizeof(s), stdin);
    int cat = atoi(s);
    FILE *m = fopen(MASTER_PATIENTS, "r");
    char line[MAXLINE];
    int found = 0;
    if(!m) { printf("No patients.\n"); pause_for_enter(); return; }
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], disease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age, 
disease, &income, &docid, type) >= 6)
        {
            int ic = (income < 200000) ? 1 : ((income < 1000000) ? 2 : 3);
            if(ic == cat)
            {
                printf("%d | %s | %d | %s | %d\n", id, name, age, disease, income);
                found = 1;
            }
        }
    }
    if(!found) printf("No patients in that income category.\n");
    fclose(m);
    pause_for_enter();
}

void statistics_dashboard()
{
    FILE *mp = fopen(MASTER_PATIENTS, "r");
    FILE *md = fopen(MASTER_DOCTORS, "r");
    char line[MAXLINE];
    int pc = 0, dc = 0;
    if(mp)
    {
        while(fgets(line, sizeof(line), mp)) pc++;
        fclose(mp);
    }
    if(md)
    {
        while(fgets(line, sizeof(line), md)) dc++;
        fclose(md);
    }
    printf("Total Patients: %d\nTotal Doctors: %d\n", pc, dc);
    FILE *m = fopen(MASTER_PATIENTS, "r");
    if(!m) { pause_for_enter(); return; }
    char disease[128];
    struct DiseaseCount { char name[128]; int count; } items[256];
    int itemc = 0;
    while(fgets(line, sizeof(line), m))
    {
        int id, age, income, docid;
        char name[128], rdisease[128], type[64];
        if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
rdisease, &income, &docid, type) >= 6)
        {
            int found = 0;
            for(int i=0;i<itemc;i++)
            {
                if(strcasecmp(items[i].name, rdisease) == 0)
                {
                    items[i].count++;
                    found = 1;
                    break;
                }
            }
            if(!found)
            {
                strcpy(items[itemc].name, rdisease);
                items[itemc].count = 1;
                itemc++;
            }
        }
    }
    fclose(m);
    int maxc = 0, maxi = -1;
    for(int i=0;i<itemc;i++)
    {
        if(items[i].count > maxc) { maxc = items[i].count; maxi = i; }
    }
    if(maxi >= 0) printf("Most common disease: %s (%d patients)\n", items[maxi].name,
items[maxi].count);
    else printf("Most common disease: N/A\n");
    int cat1=0, cat2=0, cat3=0;
    m = fopen(MASTER_PATIENTS, "r");
    if(m)
    {
        while(fgets(line, sizeof(line), m))
        {
            int id, age, income, docid;
            char name[128], rdisease[128], type[64];
            if(sscanf(line, "%d|%127[^|]|%d|%127[^|]|%d|%d|%63[^\n]", &id, name, &age,
rdisease, &income, &docid, type) >= 6)
            {
                if(income < 200000) cat1++;
                else if(income < 1000000) cat2++;
                else cat3++;
            }
        }
        fclose(m);
    }
    printf("Income categories: Low=%d Middle=%d High=%d\n", cat1, cat2, cat3);
    pause_for_enter();
}

void patient_module()
{
    while(1)
    {
        printf("\nPATIENT MENU\n1. Add New Patient\n2. Display Patients\n3. Search by
ID\n4. Update\n5. Delete\n6. Classify by Income\n7. Assign Doctor\n8. Treatment
History\n9. Back to Main Menu\nChoice: ");
        char s[8];
        fgets(s, sizeof(s), stdin);
        int c = atoi(s);
        if(c == 1) add_patient();
        else if(c == 2) display_patients();
        else if(c == 3) search_patient_by_id();
        else if(c == 4) update_patient();
        else if(c == 5) delete_patient();
        else if(c == 6) classify_patients_by_income();
        else if(c == 7) assign_doctor_to_patient();
        else if(c == 8)
        {
            printf("1. Add Entry\n2. View History\nChoice: ");
            char ss[8];
            fgets(ss, sizeof(ss), stdin);
            int ch = atoi(ss);
            if(ch == 1) add_treatment_history();
            else view_treatment_history();
        }
        else if(c == 9) break;
        else printf("Invalid choice.\n");
    }
}

void doctor_module()
{
    while(1)
    {
        printf("\nDOCTOR MENU\n1. Add Doctor\n2. Display Doctors\n3. Search by ID\n4.
Update\n5. Delete\n6. Classify by Specialisation\n7. Back to Main Menu\nChoice: ");
        char s[8];
        fgets(s, sizeof(s), stdin);
        int c = atoi(s);
        if(c == 1) add_doctor();
        else if(c == 2) display_doctors();
        else if(c == 3) search_doctor_by_id();
        else if(c == 4) update_doctor();
        else if(c == 5) delete_doctor();
        else if(c == 6) classify_doctors();
        else if(c == 7) break;
        else printf("Invalid choice.\n");
    }
}

void billing_module()
{
    while(1)
    {
        printf("\nBILLING MENU\n1. Generate Bill\n2. Back to Main Menu\nChoice: ");
        char s[8];
        fgets(s, sizeof(s), stdin);
        int c = atoi(s);
        if(c == 1) calculate_bill();
        else if(c == 2) break;
        else printf("Invalid choice.\n");
    }
}

void search_filter_module()
{
    while(1)
    {
        printf("\nSEARCH/FILTER MENU\n1. Search Patients by Disease\n2. Search Doctors
by Specialisation\n3. Search by Age Range\n4. Search by Income Category\n5. Back to
Main Menu\nChoice: ");
        char s[8];
        fgets(s, sizeof(s), stdin);
        int c = atoi(s);
        if(c == 1) search_patients_by_disease();
        else if(c == 2) search_doctors_by_specialisation();
        else if(c == 3) search_by_age_range();
        else if(c == 4) search_by_income_category();
        else if(c == 5) break;
        else printf("Invalid choice.\n");
    }
}

int main()
{
    init_system();
    while(1)
    {
        printf("\nMAIN MENU\n1. Patient Module\n2. Doctor Module\n3. Billing
System\n4. Search / Filters\n5. Statistics Dashboard\n6. Exit\nChoice: ");
        char s[8];
        fgets(s, sizeof(s), stdin);
        int c = atoi(s);
        if(c == 1) patient_module();
        else if(c == 2) doctor_module();
        else if(c == 3) billing_module();
        else if(c == 4) search_filter_module();
        else if(c == 5) statistics_dashboard();
        else if(c == 6)
        {
            printf("Saving and exiting...\n");
            break;
        }
        else printf("Invalid choice.\n");
    }
    return 0;
}
```

TESTING AND DEBUGGING

Testing and debugging were carried out to ensure the correctness, reliability, and stability of the Hospital Management System. Various test cases were executed to verify the functionality of each module and to identify and fix possible errors during development.

1. File Opening Tests

All required files such as patient master file, doctor master file, and billing files were tested for successful opening in different modes (read, write, append). Proper handling was implemented when files were not found to prevent program crashes.

Test Case 1A: Run program with no files present

Before running:

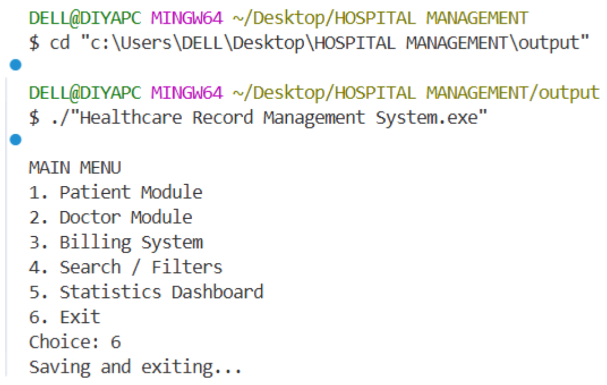
- Make sure these files do not exist in the folder:
  - patient_master.txt
  - doctors_master.txt
  - bills_master.txt

Input Sequence:

```
Run program
MAIN MENU → 6 (Exit)
```

Expected output:

- Program starts normally
- No crash
- IDs initialise as 1



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT
$ cd "c:\Users\DELL\Desktop\HOSPITAL MANAGEMENT\output"

DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 6
Saving and exiting...
```

Test Case 1B: Automatic File creation

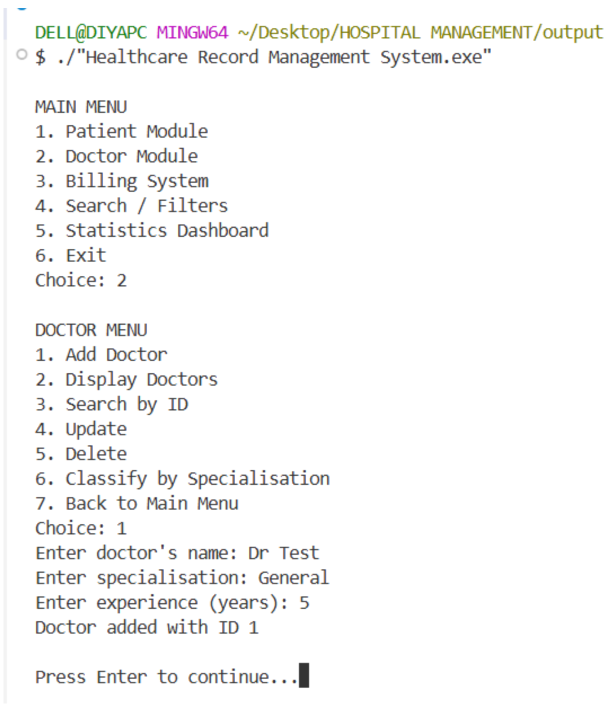
Input sequence:

```
MAIN MENU → 2 (Doctor Module)
→ 1 (Add Doctor)

Doctor Name: Dr Test
Specialisation: General
Experience: 5
[Press Enter]
```

Expected result:

- File created:
  - doctor_1.txt
  - doctors_master.txt (append mode)

- Confirms write + append mode



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 2

DOCTOR MENU
1. Add Doctor
2. Display Doctors
3. Search by ID
4. Update
5. Delete
6. Classify by Specialisation
7. Back to Main Menu
Choice: 1
Enter doctor's name: Dr Test
Enter specialisation: General
Enter experience (years): 5
Doctor added with ID 1

Press Enter to continue...
```

2. **Boundary Testing for Invalid IDs**

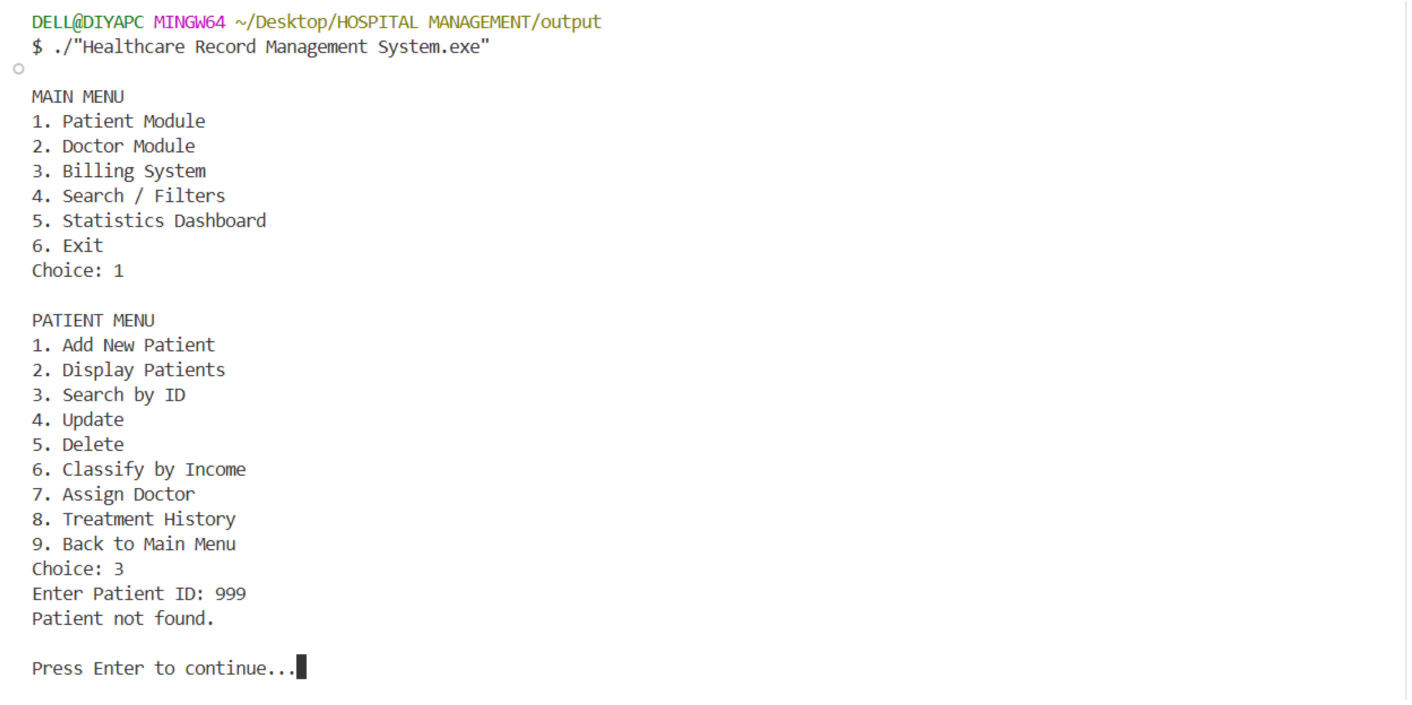
   The system was tested using invalid patient IDs and doctor IDs that did not exist in the records. The program correctly displayed error messages and prevented operations on non-existing records.

Test Case 2A: Invalid Patient ID Search

Input sequence:

```
MAIN MENU → 1 (Patient Module)
→ 3 (Search by ID)

Patient ID: 999
```

Expected output:



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 1

PATIENT MENU
1. Add New Patient
2. Display Patients
3. Search by ID
4. Update
5. Delete
6. Classify by Income
7. Assign Doctor
8. Treatment History
9. Back to Main Menu
Choice: 3
Enter Patient ID: 999
Patient not found.

Press Enter to continue...
```

Test Case 2B: Invalid Doctor Update

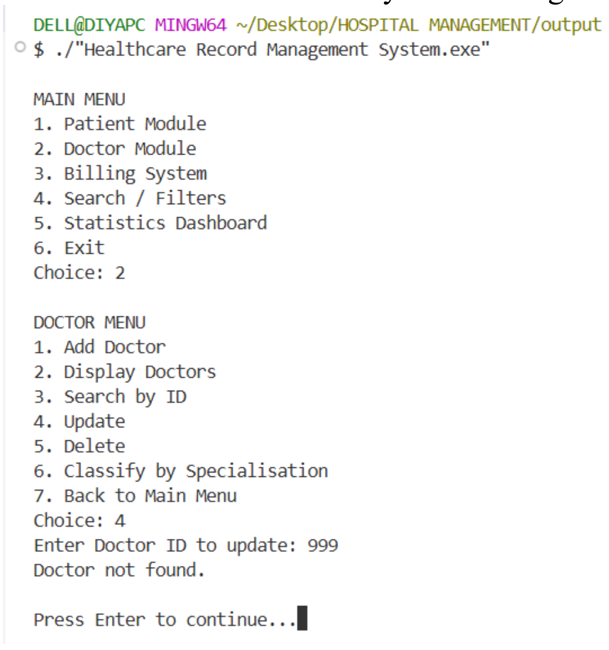
Input sequence:

```
MAIN MENU → 2 (Doctor Module)
```

```
→ 4 (Update Doctor)

Doctor ID: 999
```

Expected output:

- Confirms boundary ID checking



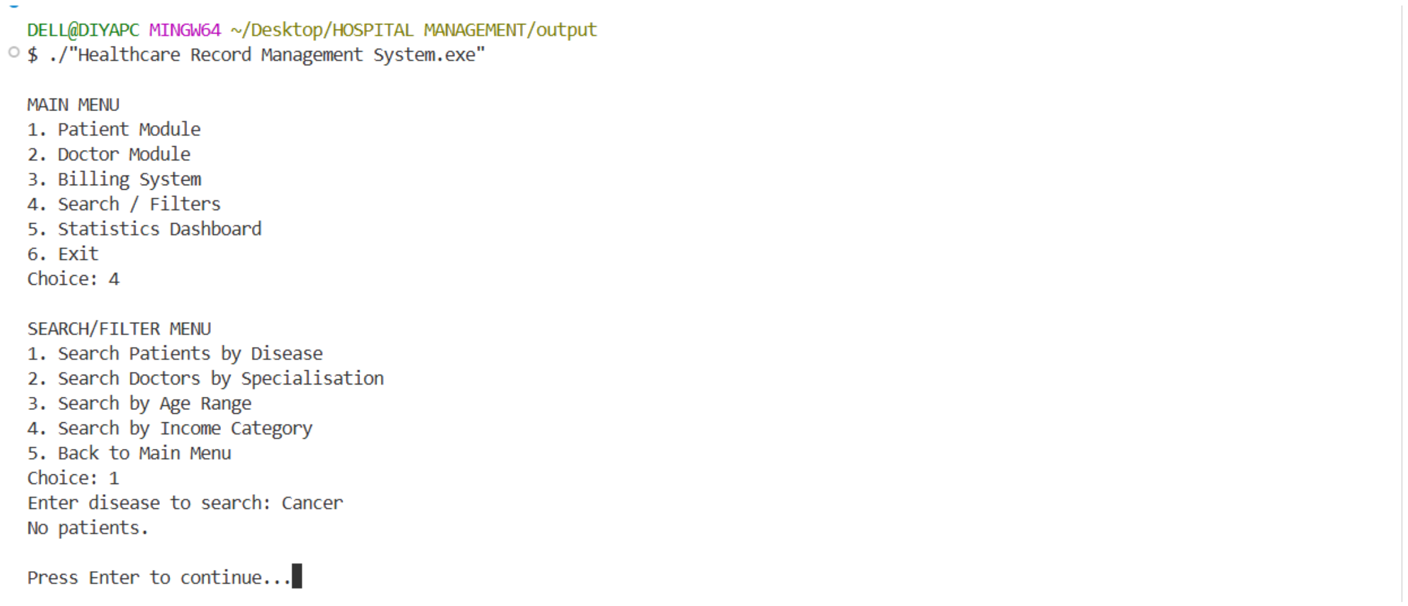
```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 2

DOCTOR MENU
1. Add Doctor
2. Display Doctors
3. Search by ID
4. Update
5. Delete
6. Classify by Specialisation
7. Back to Main Menu
Choice: 4
Enter Doctor ID to update: 999
Doctor not found.

Press Enter to continue...
```

3. **Searching Non-Existing Records**
   Search operations were performed using names, diseases, and IDs that were not present in the system. The program correctly indicated when no matching records were found, ensuring robustness of the search functionality.

   Test Case 3A: Search Patient by Disease (Non-Existing)
   Input sequence:

```
MAIN MENU → 4 (Search / Filters)
→ 1 (Search Patients by Disease)

Disease: Cancer
```

Expected output:



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 4

SEARCH/FILTER MENU
1. Search Patients by Disease
2. Search Doctors by Specialisation
3. Search by Age Range
4. Search by Income Category
5. Back to Main Menu
Choice: 1
Enter disease to search: Cancer
No patients.

Press Enter to continue...
```

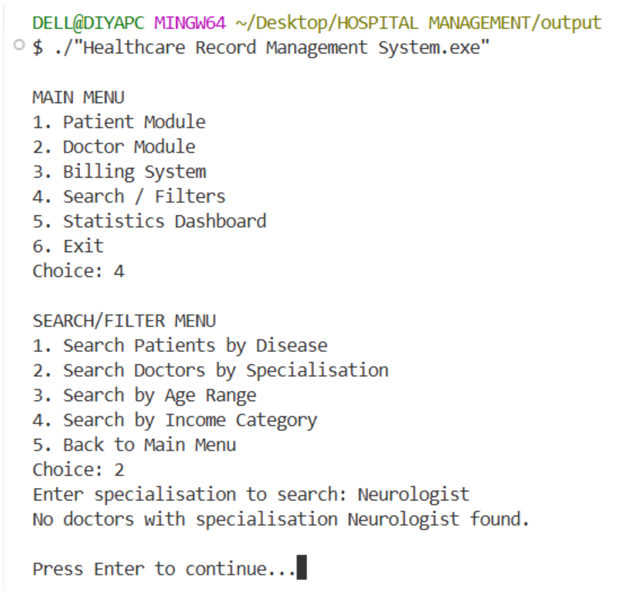
Test Case 3B: Search Doctor by Specialisation (Non-Existing)
Input sequence:

```
MAIN MENU → 4
→ 2 (Search Doctors by Specialisation)
```

```
Specialisation: Neurologist
```

Expected output:

- Confirms robust search with zero-result handling



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 4

SEARCH/FILTER MENU
1. Search Patients by Disease
2. Search Doctors by Specialisation
3. Search by Age Range
4. Search by Income Category
5. Back to Main Menu
Choice: 2
Enter specialisation to search: Neurologist
No doctors with specialisation Neurologist found.

Press Enter to continue...
```

4. Updating and Re-Saving Records

Update operations were tested by modifying patient and doctor details. After updating, records were re-saved and verified by re-displaying them, ensuring that changes were permanently stored in the files.

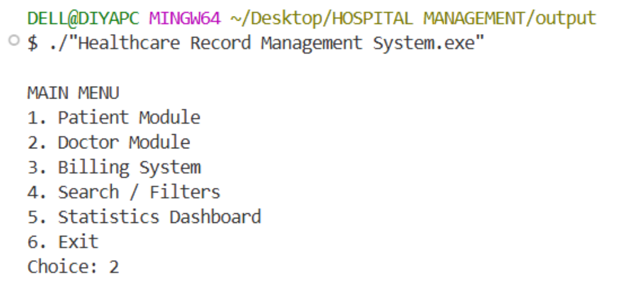
Test Case 4A: Add → Update → Verify Doctor

Input sequence:

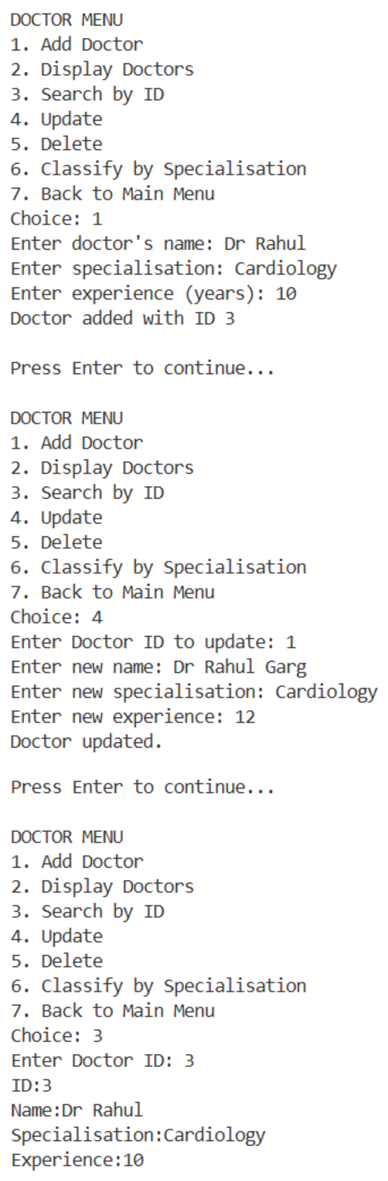
```
MAIN MENU → 2
→ 1

Name: Dr Rahul
Specialisation: Cardiology
Experience: 10

MAIN MENU → 2
→ 4

Doctor ID: 1
New Name: Dr Rahul Sharma
New Specialisation: Cardiology
New Experience: 12

MAIN MENU → 2
→ 3 (Search by ID)

Doctor ID: 1
```

Expected output:

- Confirms data re-saved correctly to files



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 2
```



```
DOCTOR MENU
1. Add Doctor
2. Display Doctors
3. Search by ID
4. Update
5. Delete
6. Classify by Specialisation
7. Back to Main Menu
Choice: 1
Enter doctor's name: Dr Rahul
Enter specialisation: Cardiology
Enter experience (years): 10
Doctor added with ID 3

Press Enter to continue...

DOCTOR MENU
1. Add Doctor
2. Display Doctors
3. Search by ID
4. Update
5. Delete
6. Classify by Specialisation
7. Back to Main Menu
Choice: 4
Enter Doctor ID to update: 1
Enter new name: Dr Rahul Garg
Enter new specialisation: Cardiology
Enter new experience: 12
Doctor updated.

Press Enter to continue...

DOCTOR MENU
1. Add Doctor
2. Display Doctors
3. Search by ID
4. Update
5. Delete
6. Classify by Specialisation
7. Back to Main Menu
Choice: 3
Enter Doctor ID: 3
ID:3
Name:Dr Rahul
Specialisation:Cardiology
Experience:10
```

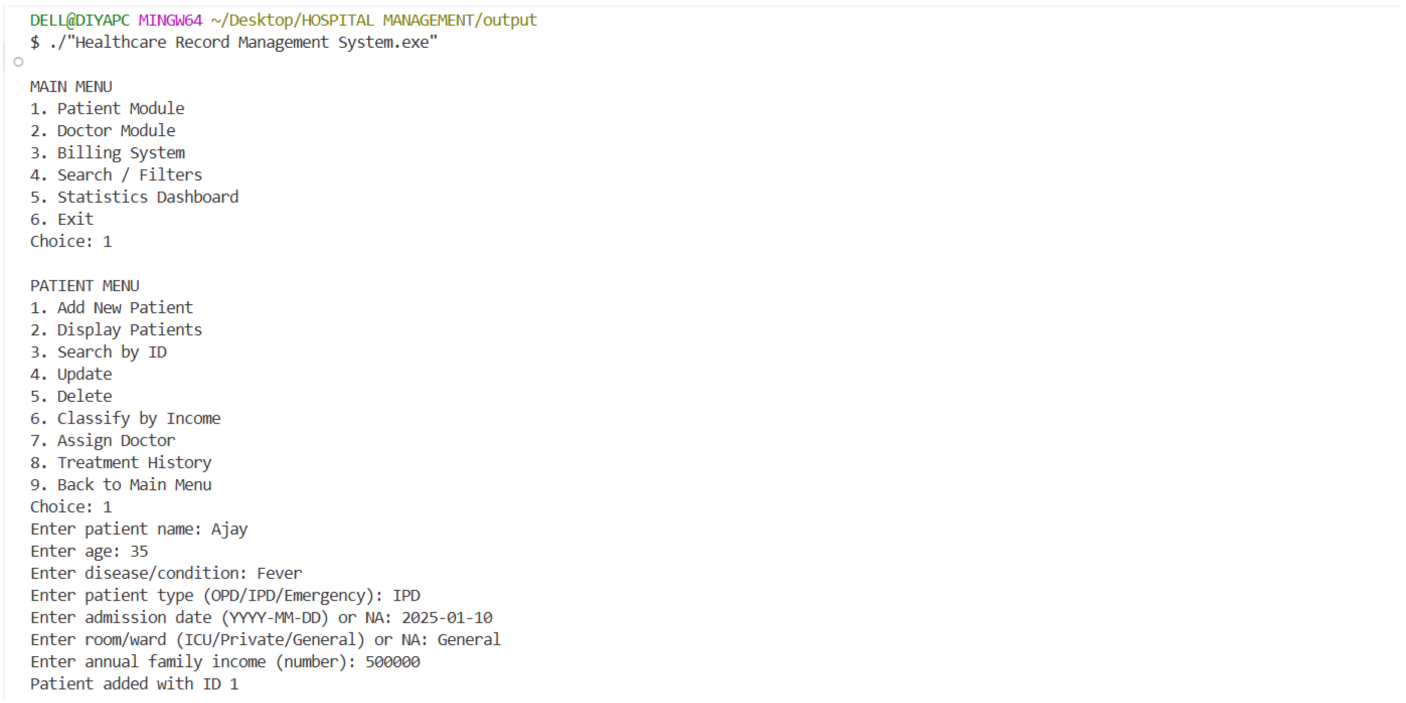
Billing Value Verification

The billing module was tested with different combinations of room charges, treatment fees, and medicine costs. The final bill amount was manually calculated and compared with the system output to confirm billing accuracy.

Test Case 5A: Controlled Billing Calculation
Precondition: Add one patient

```
MAIN MENU → 1
→ 1

Name: Ajay
Age: 35
Disease: Fever
Type: IPD
Admission Date: 2025-01-10
Room/Ward: General
```

```
Income: 500000
```



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 1

PATIENT MENU
1. Add New Patient
2. Display Patients
3. Search by ID
4. Update
5. Delete
6. Classify by Income
7. Assign Doctor
8. Treatment History
9. Back to Main Menu
Choice: 1
Enter patient name: Ajay
Enter age: 35
Enter disease/condition: Fever
Enter patient type (OPD/IPD/Emergency): IPD
Enter admission date (YYYY-MM-DD) or NA: 2025-01-10
Enter room/ward (ICU/Private/General) or NA: General
Enter annual family income (number): 500000
Patient added with ID 1
```

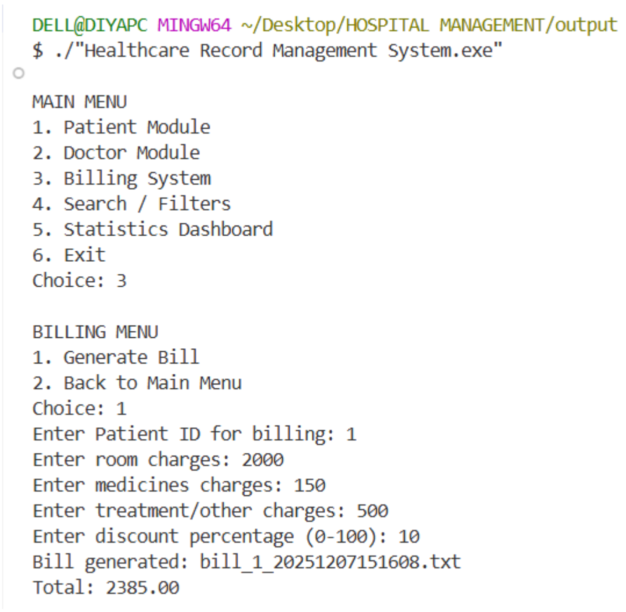
Input sequence:

```
MAIN MENU → 3 (Billing)
→ 1 (Generate Bill)

Patient ID: 1
Room Charges: 2000
Medicine Charges: 1500
Treatment Charges: 500
Discount Percentage: 10
```

Manual Calculation:

```
Subtotal = 2000 + 150 + 500 = 2650
Discount 10% = 265
Final Total = 2385
```

Expected output:



```
DELL@DIYAPC MINGW64 ~/Desktop/HOSPITAL MANAGEMENT/output
$ ./"Healthcare Record Management System.exe"

MAIN MENU
1. Patient Module
2. Doctor Module
3. Billing System
4. Search / Filters
5. Statistics Dashboard
6. Exit
Choice: 3

BILLING MENU
1. Generate Bill
2. Back to Main Menu
Choice: 1
Enter Patient ID for billing: 1
Enter room charges: 2000
Enter medicines charges: 150
Enter treatment/other charges: 500
Enter discount percentage (0-100): 10
Bill generated: bill_1_20251207151608.txt
Total: 2385.00
```

Summary Table:

Test Case	Input Given	Expected Result
File open	Program start	No crash
Invalid IDs	ID = 999	Error message
Non-existent search	Disease = Cancer	No record found
Update-save	Modify doctor	Data persists
Billing	Charges + discount	Correct total

CONCLUSION

The Hospital Management System project was successfully developed using the C programming language with effective use of file handling techniques. The system provided a structured and reliable way to manage hospital operations such as patient registration, doctor management, billing, searching, classification, and statistical analysis. Each patient and doctor record was stored in separate files along with master files, ensuring organized data storage and easy retrieval.

The implementation of automated ID generation, billing calculations, and data classification improved accuracy and reduced manual effort. The search and filter modules enabled quick access to records, while the statistics dashboard provided meaningful insights into hospital data. Rigorous testing and debugging ensured that the system performed accurately under different scenarios and handled invalid inputs gracefully.

Overall, the project successfully achieved its objectives and demonstrated that C language, combined with file handling, can be effectively used to develop real-world management systems. The project also strengthened practical understanding of structured programming, data management, and modular system design.

FUTURE SCOPE

The system can be upgraded from a console-based application to a **Graphical User Interface (GUI)** using technologies such as C++ with Qt or Java, which would improve user interaction and make the system more user-friendly.

Currently, the project uses text files for data storage. This can be replaced with a **database management system** such as MySQL or SQLite to improve data security, scalability, and performance while enabling faster data retrieval.

A **multi-user authentication system** can be introduced to allow different access levels for administrators, doctors, and staff, ensuring better data protection and controlled access to sensitive information.

The system can be extended to include **appointment scheduling**, allowing patients to book, modify, or cancel appointments with doctors, thereby improving overall hospital workflow.

Advanced **data analytics and graphical reporting** can be integrated to visually represent statistics such as patient count, disease trends, income categories, and doctor workload.

Finally, the project can be enhanced by implementing **cloud-based storage**, enabling remote access, data backup, and real-time synchronization across multiple systems.

REFERENCES

1. TutorialsPoint, *C Programming Tutorials*, TutorialsPoint Online Learning.
2. GeeksforGeeks, *C Programming and File Handling Articles*.
3. Yashavant Kanetkar, *Let Us C*, BPB Publications, New Delhi.
4. OpenAI, ChatGPT, Large Language Model used for code understanding, logical explanation, and documentation support.