**HOSPITAL MANAGEMENT SYSTEM**

**USING DATASTRUCTURES**

*A Project Report submitted to*

**JNTUA, Ananthapuram**

In partial fulfilment of the requirements for the award of the degree of

# Bachelor of Technology

(Computer Science & Engineering) By

**D.VAMSI KRISHNA (24KB1A05C6)**

**SK. MUZAMIL (24KB1A05HZ)**

**C. PAVAN (24KB1A080)**

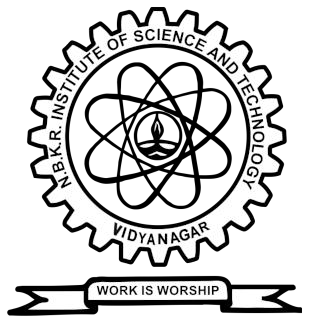
**C. NARASIMHA REDDY (24KB1A05B0)**

**CH.VENKATESH (24KB1A05C3)**

Under the esteemed Guidance of

**Sivanraj Samy, B.Tech., M. Tech., [Ph. D]**

## Assistant Professor, Department of CSE



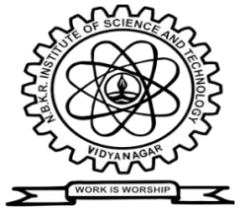
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**N.B.K.R. INSTITUTE OF SCIENCE & TECHNOLOGY**

**(AUTONOMOUS)**

**VIDYANAGAR, TIRUPATI (DIST), AP– 524413**

**APRIL 2025**



|  |  |
| --- | --- |
| Website: www.nbkrist.org. | Ph: 08624-228 247 |
| Email: ist@nbkrist.org. | Fax :08624-228 257 |

**N.B.K.R. INSTITUTE OF SCIENCE & TECHNOLOGY**

**(Autonomous)**

**(Approved by AICTE: Accredited by NBA: Affiliated to JNTUA, Ananthapuramu) An ISO**

**9001-2000 Certified Institution**

**Vidyanagar -524 413, Tirupati District, Andhra Pradesh, India**

# BONAFIDECERTIFICATE

This is to certify that the project work entitled “**HOSPITASL MANAGEMENT SYSTEM**” is a bonafide work done by C Pavan (24KB1A0580), CH Narasimha (24KB1A05B0), CH Venkatesh (24KB1A05C3), D Vamsikrishna (24KB1A05C6), SK Muzamil (24KB1A05HZ) in the department of **Computer Science & Engineering, N.B.K.R. Institute of Science & Technology**, **Vidyanagar** and is submitted to **JNTUA, Ananthapuramu** in the partial fulfilment for the award of B.Tech degree in **Computer Science & Engineering.** This work has been carried out under my supervision.

|  |  |
| --- | --- |
| ***Sivanraj Samy*** | ***Dr. A. Rajasekhar Reddy*** |
| *Trainer (Byte XL),* | *Professor & Head* |
| *Department of CSE,* | *Department of CSE,* |
| *NBKRIST, Vidyanagar.* | *NBKRIST, Vidyanagar.* |

Submitted for the Viva-Voce Examination held on \_\_\_\_\_\_\_\_\_\_\_

**Examiner-1 Examiner-2**

## ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of a project would be incomplete without the people who made it possible of their consistent guidance and encouragement crowned our efforts with success.

We would like to express our profound sense of gratitude to our project guide **Sivanraj Samy Trainer (Byte XL), Department of Computer Science and Engineering, N.B.K.R. Institute of Science and Technology (Affiliated to JNTUA, Ananthapuramu)**, Vidyanagar, for his masterful guidance and the constant encouragement throughout the project. Our sincere appreciations for his suggestions and unmatched services without which this work would have been an unfulfilled dream.

We convey our special thanks to **Dr. Y. VENKATARAMI REDDY, Chairman,**

**N.B.K.R. Institute of Science and Technology,** for providing excellent infrastructure in our campus for the completion of the project.

We convey our special thanks to **Sri. N. RAM KUMAR, Correspondent** of **N.B.K.R. Institute of Science and Technology**, for providing excellent infrastructure in our campus for the completion of the project.

We are grateful to **Dr. V. VIJAYA KUMAR REDDY, Director, N.B.K.R. Institute of Science and Technology** for allowing me to avail all the facilities in the college.

We express our sincere gratitude to **Dr. A. RAJASEKHAR REDDY, Professor& Head, Department of Computer Science and Engineering**, for providing exceptional facilities for successful completion of my project.

We would like to convey our heart full thanks to **Staff Members, Lab Technicians, and my friends**, who extend their cooperation in making this project as a successful one.

We would like to thank one and all who have helped me directly and indirectly to complete this project successfully.

### ABSTRACT

The **Hospital Management System (HMS)** is a C-based application designed to streamline patient and appointment management. This system efficiently stores patient details using an **array** and manages appointments via a **linked list**, ensuring organized and accessible healthcare records.

The system facilitates **patient registration**, allowing users to input essential details such as name, age, gender, and medical condition. Patients' data is stored in an array, ensuring quick access and modification. The system also supports **editing patient details**, enabling seamless updates.

A vital feature is the **appointment scheduling**, which enables patients to book consultations with doctors. Each appointment is dynamically linked in a **linked list**, ensuring efficient management of upcoming consultations. The system further allows users to **view and delete appointments**, maintaining flexibility in scheduling.

To enhance usability, the HMS includes **sorting functionality**, enabling patients to be arranged alphabetically for streamlined retrieval. The integration of **file handling** ensures patient records are stored persistently, allowing data retrieval upon system restart.

The system employs an intuitive **menu-driven interface**, ensuring accessibility for users with minimal technical expertise. It prioritizes **efficient memory management** and provides essential hospital administration functionalities.

This **modular, scalable, and user-friendly** HMS presents an excellent foundation for healthcare digitalization. Future improvements may include doctor scheduling, patient search functionality, and integration with a database for larger-scale hospital operations.

## LIST OF FIGURES

### FIG NO FIGURE NAME PAGE NO

3.1.1 System pipeline 09

3.2.1 Methodology 10

4.3.1 Data Flow Diagram 14

6.2.1 Input 34

6.2.2 Output 35

## TABLE OF CONTENTS

**CHAPTER NO CHAPTER NAME PAGENO**

**ACKNOWLEDGEMENT**

**ABSTRACT**

**LIST OF FIGURES**

**CHAPTER 1 INTRODUCTION 1-4**

1.1 Introduction

1.2 Background and Motivation

1.3 Problem Statement

1.4 Objectives and Scope

1.5 Organization of the Project Report

1.6 Summary

**CHAPTER 2 SURVEY OF LITERATURE 5-8**

2.1 Introduction

2.2 Literature Survey

2.3 Existing System

2.4 Feasibility Analysis

2.5 Summary

**CHAPTER 3 METHODOLOGY 9-12**

3.1 Introduction

3.2 Overview of Methodological Approach

3.3 Description of Tools and Technologies Used

|  |  |  |
| --- | --- | --- |
| **CHAPTER 4** | **SYSTEM DESIGN**  4.1 Introduction  4.2 Detailed Design of Components  4.3 Data Flow Diagram  4.4 System Requirements  4.5 Summary | **13-20** |
| **CHAPTER 5** | **IMPLEMENTATION**  5.1 Introduction  5.2 Code Structure and Organization Procedure  5.3 Algorithms and Techniques Implemented  5.4 Summary | **21-33** |
| **CHAPTER 6** | **RESULTS AND ANALYSIS**  6.1 Introduction  6.2 Input and Output  6.3 Summary | **34-36** |
| **CHAPTER 7** | **CONCLUSION AND FUTURE WORK** | **37-38** |

7.1 Conclusions Drawn from the Study

7.2 Limitations and Challenges Encountered

**CHAPTER–1**

**INTRODUCTION**

### OVERVIEW

The rapid advancement of technology has revolutionized various industries, including healthcare, where efficient hospital management systems are essential for organizing patient records and appointments. The **Hospital Management System (HMS)** developed using **C programming** is a structured, data-driven solution designed to streamline hospital administrative tasks. By utilizing **arrays for patient records** and **linked lists for appointment scheduling**, the system ensures optimized **data storage, retrieval, and management**.

The system is built with an interactive **menu-driven interface**, allowing hospital administrators to **register patients, manage appointments, edit records, sort data**, and **store information persistently** using file handling. This enables hospitals to **maintain accurate records**, reduce **manual paperwork**, and improve **workflow efficiency**.

The HMS incorporates essential functionalities such as **patient registration**, **appointment booking**, **sorting for quick retrieval**, and **file handling for data persistence**. Through a structured approach, healthcare providers can efficiently **store patient details**, assign appointments to doctors, and manage hospital workflows. The integration of **sorting mechanisms** allows administrators to retrieve patient data seamlessly, while **dynamic linked list structures** ensure that appointment management remains flexible.

From a technical perspective, this system highlights the practical implementation of **static and dynamic data structures**. The use of **arrays** enables structured and quick access to patient information, while the **linked list for appointments** allows flexible scheduling without memory constraints. Additionally, **file handling** guarantees data persistence, ensuring records remain available even after system restarts.

The application is **lightweight, scalable, and practical**, making it suitable for small to medium-sized healthcare institutions. The intuitive **text-based interface** ensures accessibility, enabling easy navigation for hospital staff without requiring advanced technical expertise.

Overall, the **Hospital Management System in C** is more than just an academic project—it serves as a **fundamental tool for healthcare digitalization**, helping hospitals **automate their operations** and **enhance patient care through structured data management**. With potential enhancements such as **doctor scheduling, database integration, and web-based accessibility**, this system has the capacity to evolve into a **comprehensive hospital management solution**.

### 1.2 BACKGROUND AND MOTIVATION

### Efficient hospital management is a critical aspect of healthcare, ensuring seamless administration and improved patient care. However, traditional paper-based systems and unstructured digital databases often lead to inefficiencies, data mismanagement, and difficulty in tracking patient records. To address these challenges, hospital management systems (HMS) offer a streamlined approach, integrating structured data storage and retrieval using fundamental computer science concepts like arrays and linked lists.

### The motivation behind this Hospital Management System is to provide an efficient, structured, and scalable method for handling patient records and appointments. This system is designed to:

### Enhance Organization: Ensures systematic patient registration, appointment scheduling, and record management.

### Optimize Healthcare Workflows: Automates administrative tasks to reduce human errors and improve efficiency.

### Improve Data Accessibility: Enables quick sorting, editing, and retrieval of patient details through structured data storage.

### Ensure Data Security and Persistence: Uses file handling to store patient information securely, preventing data loss.

### Increase Scalability: The modular design allows future enhancements, such as doctor scheduling and database integration.

### By implementing C-based data structures, this project bridges the gap between traditional hospital management and digital transformation, offering an intuitive and effective solution for healthcare administration.

### 1.3 PROBLEM STATEMENT

Managing hospital operations efficiently is a complex task due to the volume of patient records, appointment scheduling, and administrative overhead. Traditional manual systems often lead to **data mismanagement, appointment conflicts, and accessibility challenges**, affecting patient care and hospital workflow. Paper-based records are prone to **loss, errors, and inefficiency**, while unstructured digital systems lack scalability and integration capabilities.

To address these challenges, hospitals need a **structured and automated management system** that ensures **organized record-keeping, seamless appointment scheduling, and secure data handling**. A **Hospital Management System (HMS)** built using **C programming** can provide an **efficient solution** by integrating **data structures like arrays for patient records and linked lists for appointment scheduling**.

### 1.4 OBJECTIVES & SCOPE

### Objectives:

### Efficient Patient Management: Provide a structured platform for hospitals to store and manage patient records.

### Streamline Appointment Scheduling: Enable dynamic booking, modification, and tracking of appointments using linked lists.

### Data Accessibility & Persistence: Ensure quick retrieval, sorting, and secure file storage for hospital data.

### User-Friendly Interface: Simplify hospital administration with a menu-driven system for easy navigation.

### Reduce Manual Workload: Automate hospital processes to minimize errors and improve efficiency.

### Scalability & Future Enhancements: Allow future integrations like doctor scheduling and database support.

### Scope:

### Comprehensive Patient Data Storage: Uses arrays for structured record-keeping.

### Dynamic Appointment Management:Linked lists ensure flexible scheduling.

### Sorting & Editing: Enables quick access and modification of patient records.

### File Handling for Data Security: Ensures persistent storage of records.

### Scalable for Future Expansions: Can integrate advanced search and web-based accessibility.

### This system offers a reliable, efficient, and scalable solution for hospital management, enhancing workflow optimization and patient care.

### 1.5 ORGANIZATION OF THE PROJECT REPORT

The **Hospital Management System (HMS)** is designed to provide an efficient and structured solution for managing **patient records and appointment scheduling** in healthcare institutions. The project aims to **streamline hospital workflows** by digitizing essential administrative tasks such as **patient registration, appointment booking, data sorting, and file handling**, ensuring smooth operations and improved accessibility.

This system utilizes **arrays** for storing patient records and **linked lists** for handling dynamic appointment scheduling, offering flexibility in data management. Additionally, the integration of **sorting mechanisms** enables quick retrieval of patient details, enhancing efficiency. **File handling ensures data persistence**, preventing information loss upon system restart.

The **menu-driven interface** simplifies navigation, making the system accessible to hospital staff with minimal technical expertise. The modular design facilitates **scalability and future expansions**, including **doctor scheduling, search functionalities, and database integration** for larger-scale hospital operations.

By digitizing patient management processes, the **Hospital Management System** optimizes **administrative efficiency, reduces paperwork, and enhances patient care**, ultimately contributing to the **digital transformation of healthcare institutions**. This report details the **design, implementation, and analysis** of the system while outlining potential improvements for broader applications.

### 1.6 SUMMARY

The Hospital Management System (HMS) is a C-based application designed to efficiently manage patient records and appointment scheduling using arrays and linked lists. It provides a menu-driven interface for patient registration, appointment booking, data sorting, and file handling to ensure data persistence. By automating hospital workflows, the system reduces manual errors, improves efficiency, and enables seamless data retrieval and modification.

**CHAPTER–2**

**SURVEY OF LITERATURE**

### 2.1 INTRODUCTION

The field of **hospital management systems** has seen significant advancements due to the growing need for **efficient healthcare administration**. Traditionally, hospitals relied on **manual record-keeping and paper-based systems**, which often led to **errors, inefficiencies, and accessibility challenges**. As healthcare institutions expand, the demand for **digital solutions** to manage patient records, appointments, and administrative tasks has increased.

Research suggests that **structured data management systems**, such as **Hospital Management Systems (HMS),** improve efficiency, minimize errors, and streamline workflows. Various studies highlight the importance of **automated systems** in optimizing healthcare services, reducing administrative burden, and enhancing **patient care**. Modern HMS implementations leverage **data structures** like **arrays** for storing patient records and **linked lists** for appointment scheduling to ensure **organized data handling**.

Additionally, literature on **healthcare digitization** emphasizes the role of **file handling mechanisms** in ensuring **secure data storage and retrieval**, preventing information loss upon system restart. Studies indicate that **menu-driven interfaces** improve accessibility for hospital staff by providing **simple and intuitive navigation** without requiring advanced technical expertise.

This section examines existing research on **hospital management systems**, highlighting current trends, methodologies, and challenges. By identifying **limitations in manual systems** and reviewing **digital solutions**, the objective is to justify the need for a **C-based HMS** and explore its **potential impact** on healthcare operations.

**2.2 LITERATURE SURVEY:**

1. Several studies have explored the effectiveness of digital hospital management systems**in improving healthcare administration. Research indicates that structured patient record** management and automated appointment scheduling enhance hospital efficiency and **minimize errors.**
2. **Data structures in healthcare management** play a crucial role in optimizing **data storage and retrieval**. Arrays have been widely used for structured record-keeping, while linked lists allow dynamic management of appointments, ensuring flexible scheduling.
3. Various healthcare systems have integrated **sorting techniques** to simplify patient record access, reducing search time and improving overall workflow efficiency.
4. The implementation of **file handling mechanisms** in hospital management ensures **data persistence**, preventing loss of patient records while maintaining accessibility across multiple sessions.

### 2.3 EXISTING SYSTEM

Traditional hospital management relies on manual record-keeping and basic digital systems, often leading to data inefficiencies and accessibility issues. While some hospitals use electronic health records (EHRs), they can be costly and complex for smaller facilities. Standalone scheduling tools don’t integrate patient management, resulting in fragmented workflows. The Hospital Management System (HMS) using C provides a lightweight, structured approach with arrays for patient storage, linked lists for appointments, and file handling for data persistence, making it an efficient and scalable solution for healthcare administration.

**Disadvantages:**

### Limited Scalability – Since the system relies on arrays for storing patient records, it may struggle to handle large datasets efficiently.

### Lack of Advanced Search Functionality – The system does not support complex search queries, making data retrieval slower for large hospitals.

### Basic User Interface – The menu-driven interface is text-based, which may not be as intuitive as a graphical interface for hospital staff.

### No Multi-User Support – The system lacks role-based access, preventing multiple users from managing hospital data simultaneously.

### Limited Security Measures – File handling does not include encryption or authentication, making stored patient records vulnerable to unauthorized access.

### No Real-Time Updates – The system does not support live synchronization, meaning multiple users cannot edit or view updated records in real time.

### Absence of Database Integration – The system relies solely on file handling, which may not be as efficient as structured databases like MySQL for long-term scalability.

**Advantages:**

**Advantages of the Hospital Management System Using C**

1. Efficient Data Management – Uses **arrays** for structured patient records and **linked lists** for dynamic appointment scheduling, ensuring seamless access and updates.
2. Lightweight and Fast – Written in **C**, making it **highly efficient with low memory consumption**, ideal for small to medium-sized hospitals.
3. Data Persistence – Incorporates **file handling** to store patient and appointment records, preventing data loss upon system restart.
4. Simple and User-Friendly – Uses a **menu-driven interface**, making it accessible for hospital staff without requiring technical expertise.
5. Customizable and Expandable – Can be modified to integrate **new functionalities** like doctor assignment, advanced search, and database integration.
6. Automates Hospital Operat**ions** – Reduces **manual paperwork**, minimizes errors, and speeds up administrative tasks.
7. Cost-Effective – Unlike high-end hospital management software, this system is **free to implement** and doesn’t require expensive infrastructure.

2.4 FEASIBILITY ANALYSIS

**Technical Feasibility:**

The system is developed using C programming, leveraging fundamental data structures like arrays for patient records and linked lists for appointment scheduling. File handling ensures data persistence, preventing loss upon system restart. Since C is lightweight and efficient, the system can run on low-resource environments without requiring extensive hardware. Future improvements could include database integration for scalability.

**Operational Feasibility:**

The menu-driven interface is simple and intuitive, making it accessible to hospital staff with minimal technical expertise. The system requires low maintenance, with basic updates for data security and functionality enhancements. User support can be provided through documentation and help guides.

**Economic Feasibility:**

The development cost is low, as the system relies on open-source technologies and does not require expensive infrastructure. File-based storage eliminates the need for high-end database licenses. Operational costs remain manageable, making the system cost-effective for small to medium-sized healthcare facilities.

### Legal and Ethical Feasibility:

### The system can be designed to comply with healthcare data protection regulations by implementing secure file storage and access controls. While it currently lacks encryption or multi-user authentication, future versions can integrate security measures to ensure compliance with hospital management standards.

### Schedule Feasibility:

### The system can be developed in phases over 4-6 months, with continuous improvements based on hospital feedback. Initial deployment can focus on patient record management and appointment scheduling, while later enhancements may include doctor scheduling, search functionalities, and database integration.

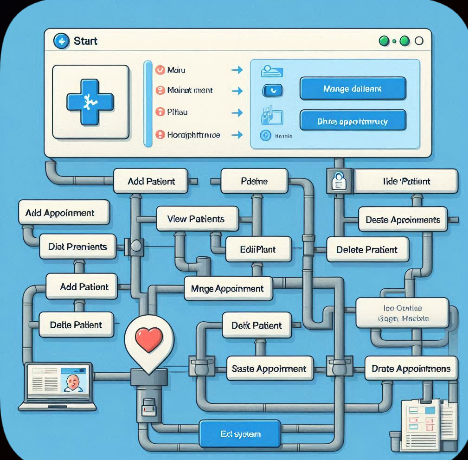
### 2.5 SUMMARY

The literature survey examines existing tools and approaches for teaching data structures, focusing on traditional methods such as textbooks, lectures, and online tutorials. These methods, while effective, often lack interactivity and fail to engage students in an active learning process. Several modern educational platforms have incorporated gamification, quizzes, and coding challenges to improve engagement and learning outcomes. However, many of these platforms focus on theory or coding separately, lacking an integrated approach. The survey highlights the need for systems that combine both theoretical concepts and practical coding exercises in an interactive, gamified environment. It emphasizes the gap in personalized learning paths and real-time feedback, which the proposed system aims to address.

**CHAPTER – 3 METHODOLOGY**

### 3.1 INTRODUCTION

The **Hospital Management System (HMS)** is designed to provide an efficient, structured solution for managing patient records and appointment scheduling using **C** programming **and** data structures. It employs arrays for structured storage, linked lists for dynamicscheduling, and file handling for data persistence, ensuring hospital data remains accessible even after system restarts. The system features a menu-driven interface, making it easy to navigate for hospital staff. By automating administrative processes, HMS enhances workflow efficiency**,** minimizes errors, and optimizes healthcare management, with future scalability for database integration**,** advanced search capabilities, and doctor scheduling.



**Fig 3.1.1: System pipeline**

The methodology follows a systematic approach involving five key phases: requirement development, testing analysis, design,**,** and deployment. Initially, hospital and systemrequirements are gathered to define the features of the **Hospital Management System (HMS)**, including patient registration, appointment scheduling, and file handling.

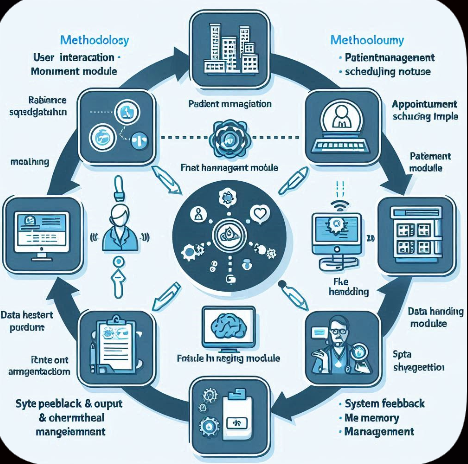
In the design phase, the system architectureis planned, incorporating data structures such as arrays for patient records, linked lists for dynamic scheduling, and file handling for datapersistence.

Development involves coding the application using **C**, ensuring efficient data management**,** streamlined hospital workflows**,** and scalability.

Testing is conducted to validate system functionality, accuracy of patient records**,** and smooth user navigation, preventing errors in hospital operations.

Finally, the system is deployed, with provisions for future enhancements, such as database integration, real-time updates**,** and improved UI. This methodological approachensures anefficient**,** structured**,** and scalable solution for hospital administration.

### 3.2 OVERVIEW OF METHODOLOGICAL APPROACH



**Fig 3.2.1: Methodology**

### 3.3 DESCRIPTION OF TOOLS AND TECHNOLOGIES USED

**C Programming Language:**

The primary language used to implement the Hospital Management System (HMS). C is efficient and close to system-level programming, making it ideal for memory management and structured data handling

**GCC (GNU Compiler Collection):**

This open-source compiler is used to compile the C code. It is known for its reliability, speed, and portability across different operating systems.

**Code::Blocks / Turbo C++ IDE:**

These integrated development environments provide features like syntax highlighting, auto-complete, and debugging tools to simplify the development process and reduce errors.

**Standard C Libraries:**

**stdio.h:** Handles standard input/output functions like printf() and scanf().

**stdlib.h:** Used for memory allocation, process control, conversions, and random numbers.

**string.h:** Provides functions to manipulate strings, such as strcpy() and strcmp().

**time.h:** Utilized for generating random numbers based on system time, essential for randomizing questions.

**Data Structures**

* Arrays – Used to store and organize patient records efficiently.
* Linked Lists – Manages dynamicappointment scheduling, enabling easy insertion and deletion.

**Operating System**

The Hospital Management System runs on standard platforms like Windows or Linux, ensuring high accessibility for hospital administration.Text Editor (e.g., Notepad++, Sublime Text).

**CHAPTER – 4 SYSTEM DESIGN**

### 4.1 INTRODUCTION

### System design is a crucial phase in software development, defining the architecture and functionality of the Hospital Management System (HMS). It serves as a blueprint for efficient hospital operations.In this project, the design focuses on developing a structured hospital management system using C programming. The system includes modules for patient registration, appointment scheduling, and file handling to ensure seamless hospital administration. Arrays are used for structured patient data storage, while linked lists manage dynamic appointment scheduling, allowing easy insertion, deletion, and modification of records.

### 4.2 DETAILED DESIGN OF COMPONENTS

**1. User Interface (UI):**

The user interface is text-based, designed using standard C input and outputfunctions like printf() and scanf(). It ensures clarity and ease of use, guiding hospital staff through various operations smoothly.

* **Welcome Screen** – Displays the **hospital management system title**, providing a brief instructional overview.
* **Patient Registration** – Allows staff to input patient details, including name**,** age,andmedical history.
* **Appointment Scheduling** – Presents available appointment slots, enabling staff to book or modify appointments.
* **Record Management** – Displays stored patient details with options to edit or retrieve information.
* **File Handling Integration** – Ensures data persistence by storing patient records in structured files.

**2. Data Structures:**

* **Arrays –** Used for storing patient records efficiently with quick lookup and retrieval.
* **Linked Lists –** Implemented for dynamic appointment scheduling, allowing flexible data management.

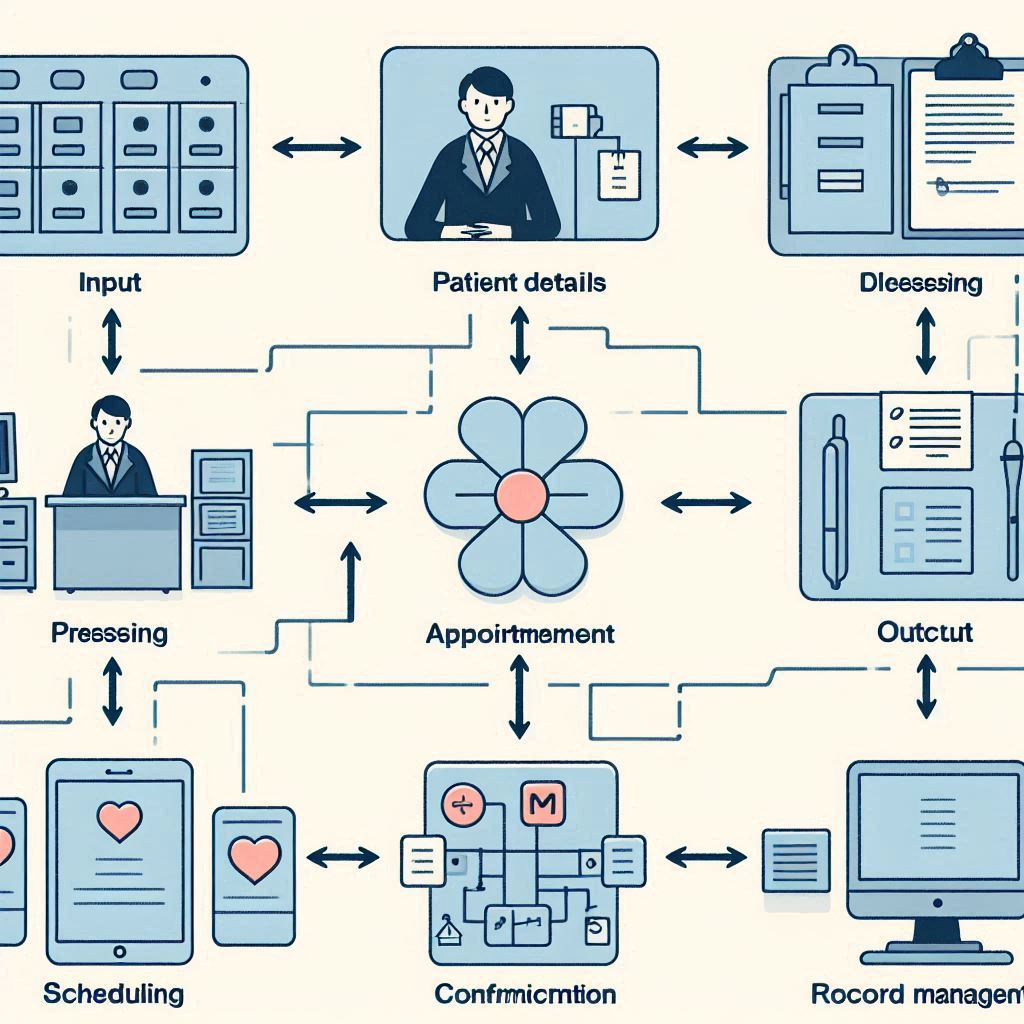
**3. Functional Modules**:

* Patient Management – Handles new patient entries, data updates, and deletion.
* Appointment Handling – Allows booking, canceling, and modifying appointments using linked lists.
* File Storage System – Enables saving and retrieving patient and appointment data persistently.

**4. Security & Data Integrity:**

* Implements basic input validation to prevent incorrect entries in patient records.
* Future enhancements can include authentication mechanisms to restrict unauthorized access.
* Ensures consistent data handling by validating stored files before retrieval.

### 4.3 DATA FLOW DIAGRAM



**Fig 4.3.1 : Data Flow Diagram**

### 4.4 SYSTEM REQUIREMENTS

**HARDWARE REQUIREMENTS:**

* **1. Hardware Requirements:**
* Processor: Minimum Intel Core i3 or equivalent.
* RAM**:** At least 4GB (Recommended: 8GB for smoother performance).
* Storage: 100MB free space for storing patient records and logs.
* Display**:** Standard monitor with text-mode compatibility.
* Input Devices**:** Keyboard for patient data entry and system navigation.
* **2. Software Requirements:**
* Operating System: Compatible with Windows (7/10/11) or Linux-based OS.
* Compiler**:** GCC (GNU Compiler Collection) or Turbo C++.
* IDE**:** Code::Blocks, Turbo C++, or any C-compatible development environment.

### 4.5 SUMMARY

The Hospital Management System (HMS) is a C-based software designed for efficient patient record management and appointment scheduling**.** It requires a basic computing environment**,** running on Windows or Linux, with a minimum Intel Core i3 processor**,** 4GB RAM, and 100MB storage for seamless functionality. The system is compiled usingGCC or Turbo C++, and relies on standard C libraries (stdio.h, stdlib.h, string.h, time.h)for essential operations.Its menu-driven interface simplifies user navigation, allowing hospital staff to register patients, book and modify appointments, and store records efficiently. The system employs arrays for structured patient data storage and linked lists for dynamic appointment handling, ensuring scalability and easy modifications. File handling is integrated for data persistence, preventing information loss upon system restart.

**CHAPTER-5 IMPLEMENTATION**

### 5.1 INTRODUCTION

### The Hospital Management System (HMS) is implemented using C programming, incorporating efficient data structures and file handling to manage patient records and appointment scheduling. The system follows a modular approach, ensuring scalability and ease of maintenance.The implementation begins with defining patient data storage, utilizing arrays for structured records and linked lists for dynamic scheduling. The menu-driven interface enables hospital staff to navigate operations efficiently, including registration, appointment booking, and record retrieval. File handling techniques ensure data persistence, preventing loss upon system restart.

### The system integrates input validation mechanisms to maintain data integrity, reducing errors in patient records. With a focus on efficiency and reliability, the implementation provides a structured framework for managing hospital workflows. Future enhancements may include database integration, advanced search functionality, and multi-user support, further improving operational effectiveness.

To ensure data persistence, the system integrates file handling, allowing stored records to remain intact even after the program is restarted. Input validationmechanisms prevent errors, maintaining **data integrity** for reliable hospital operations.Additionally, the system follows a modular programming approach, with each functionality encapsulated within specific functions, making it easier to expand and integrate new features like database connectivity**,** real-time scheduling, and multi-user authentication. The implementation ensures efficiency, ease of use, and structured data management, paving the way for a robust and scalable hospitaladministration tool.

### 5.2 CODE STRUCTURE AND ORGANIZATION PROCEDURE

#### 5.2.1 SOURCE CODE

#### #include <stdio.h>

#### #include <stdlib.h>

#### #include <string.h>

#### #define MAX\_PATIENTS 100

#### #define FILENAME "patients.txt"

#### // Patient structure

#### typedef struct {

#### int id;

#### char name[100];

#### int age;

#### char gender[10];

#### char disease[100];

#### } Patient;

#### // Appointment structure (Linked List)

#### typedef struct Appointment {

#### int patientId;

#### char date[20];

#### char time[10];

#### struct Appointment\* next;

#### } Appointment;

#### // Global Variables

#### Patient patients[MAX\_PATIENTS];

#### int patientCount = 0;

#### Appointment\* appointmentHead = NULL;

#### // Function declarations

#### void addPatient();

#### void viewPatients();

#### void editPatient();

#### void savePatients();

#### void loadPatients();

#### void sortPatients();

#### void addAppointment();

#### void viewAppointments();

#### void deleteAppointment();

#### void menu();

#### // Main function

#### int main() {

#### loadPatients();

#### menu();

#### return 0;

#### }

#### // Menu function

#### void menu() {

#### int choice;

#### while (1) {

#### printf("\n=== Hospital Management System ===\n");

#### printf("1. Add Patient\n");

#### printf("2. View Patients\n");

#### printf("3. Edit Patient\n");

#### printf("4. Add Appointment\n");

#### printf("5. View Appointments\n");

#### printf("6. Delete Appointment\n");

#### printf("7. Sort Patients by Name\n");

#### printf("8. Save and Exit\n");

#### printf("Enter your choice: ");

#### scanf("%d", &choice);

#### getchar(); // Clear buffer

#### switch (choice) {

#### case 1:

#### addPatient();

#### break;

#### case 2:

#### viewPatients();

#### break;

#### case 3:

#### editPatient();

#### break;

#### case 4:

#### addAppointment();

#### break;

#### case 5:

#### viewAppointments();

#### break;

#### case 6:

#### deleteAppointment();

#### break;

#### case 7:

#### sortPatients();

#### break;

#### case 8:

#### savePatients();

#### printf("Data saved. Exiting...\n");

#### exit(0);

#### default:

#### printf("Invalid choice. Try again.\n");

#### }

#### }

#### }

#### // Add new patient

#### void addPatient() {

#### if (patientCount >= MAX\_PATIENTS) {

#### printf("Patient list full!\n");

#### return;

#### }

#### printf("\n--- Add Patient ---\n");

#### patients[patientCount].id = patientCount + 1;

#### printf("Enter Name: ");

#### fgets(patients[patientCount].name, sizeof(patients[patientCount].name), stdin);

#### patients[patientCount].name[strcspn(patients[patientCount].name, "\n")] = '\0';

#### printf("Enter Age: ");

#### scanf("%d", &patients[patientCount].age);

#### getchar();

#### printf("Enter Gender: ");

#### scanf("%s", patients[patientCount].gender);

#### getchar();

#### printf("Enter Disease: ");

#### fgets(patients[patientCount].disease, sizeof(patients[patientCount].disease), stdin);

#### patients[patientCount].disease[strcspn(patients[patientCount].disease, "\n")] = '\0';

#### printf("Patient added with ID: %d\n", patients[patientCount].id);

#### patientCount++;

#### }

#### // View all patients

#### void viewPatients() {

#### printf("\n--- Patient List ---\n");

#### for (int i = 0; i < patientCount; i++) {

#### printf("\nID: %d\nName: %s\nAge: %d\nGender: %s\nDisease: %s\n",

#### patients[i].id, patients[i].name, patients[i].age,

#### patients[i].gender, patients[i].disease);

#### }

#### }

#### // Edit a patient

#### void editPatient() {

#### int id, found = 0;

#### printf("\nEnter Patient ID to Edit: ");

#### scanf("%d", &id);

#### getchar();

#### for (int i = 0; i < patientCount; i++) {

#### if (patients[i].id == id) {

#### found = 1;

#### printf("Editing Patient %s:\n", patients[i].name);

#### printf("Enter New Name: ");

#### fgets(patients[i].name, sizeof(patients[i].name), stdin);

#### patients[i].name[strcspn(patients[i].name, "\n")] = '\0';

#### printf("Enter New Age: ");

#### scanf("%d", &patients[i].age);

#### getchar();

#### printf("Enter New Gender: ");

#### scanf("%s", patients[i].gender);

#### getchar();

#### printf("Enter New Disease: ");

#### fgets(patients[i].disease, sizeof(patients[i].disease), stdin);

#### patients[i].disease[strcspn(patients[i].disease, "\n")] = '\0';

#### printf("Patient updated!\n");

#### break;

#### }

#### }

#### if (!found) {

#### printf("Patient ID not found.\n");

#### }

#### // Save patients to file

#### void savePatients() {

#### FILE\* fp = fopen(FILENAME, "w");

#### if (fp == NULL) {

#### printf("Error saving file.\n");

#### return;

#### }

#### for (int i = 0; i < patientCount; i++) {

#### fprintf(fp, "%d;%s;%d;%s;%s\n", patients[i].id, patients[i].name,

#### patients[i].age, patients[i].gender, patients[i].disease);

#### }

#### fclose(fp);

#### }

#### // Load patients from file

#### void loadPatients() {

#### FILE\* fp = fopen(FILENAME, "r");

#### if (fp == NULL) {

#### // No file yet

#### return;

#### }

#### while (fscanf(fp, "%d;%[^;];%d;%[^;];%[^\n]\n",

#### &patients[patientCount].id,

#### patients[patientCount].name,

#### &patients[patientCount].age,

#### patients[patientCount].gender,

#### patients[patientCount].disease) == 5) {

#### patientCount++;

#### }

#### fclose(fp);

#### }

#### // Sort patients by name

#### void sortPatients() {

#### Patient temp;

#### for (int i = 0; i < patientCount - 1; i++) {

#### for (int j = i + 1; j < patientCount; j++) {

#### if (strcmp(patients[i].name, patients[j].name) > 0) {

#### temp = patients[i];

#### patients[i] = patients[j];

#### patients[j] = temp;

#### }

#### }

#### }

#### printf("Patients sorted by name!\n");

#### }

#### // Add new appointment

#### void addAppointment() {

#### int id, found = 0;

#### printf("\nEnter Patient ID for Appointment: ");

#### scanf("%d", &id);

#### getchar();

#### for (int i = 0; i < patientCount; i++) {

#### if (patients[i].id == id) {

#### found = 1;

#### break;

#### }

#### }

#### if (!found) {

#### printf("Patient not found!\n");

#### return;

#### }

#### Appointment\* newApp = (Appointment\*)malloc(sizeof(Appointment));

#### newApp->patientId = id;

#### printf("Enter Date (DD/MM/YYYY): ");

#### fgets(newApp->date, sizeof(newApp->date), stdin);

#### newApp->date[strcspn(newApp->date, "\n")] = '\0';

#### printf("Enter Time (HH:MM): ");

#### fgets(newApp->time, sizeof(newApp->time), stdin);

#### newApp->time[strcspn(newApp->time, "\n")] = '\0';

#### newApp->next = NULL;

#### // Insert at end

#### if (appointmentHead == NULL) {

#### appointmentHead = newApp;

#### } else {

#### Appointment\* temp = appointmentHead;

#### while (temp->next != NULL) {

#### temp = temp->next;

#### }

#### temp->next = newApp;

#### }

#### printf("Appointment added successfully!\n");

#### }

#### // View all appointments

#### void viewAppointments() {

#### Appointment\* temp = appointmentHead;

#### if (temp == NULL) {

#### printf("No appointments scheduled.\n");

#### return;

#### }

#### printf("\n--- Appointments ---\n");

#### while (temp != NULL) {

#### printf("Patient ID: %d | Date: %s | Time: %s\n",

#### temp->patientId, temp->date, temp->time);

#### temp = temp->next;

#### }

#### }

#### // Delete an appointment

#### void deleteAppointment() {

#### int id;

#### printf("\nEnter Patient ID to Delete Appointment: ");

#### scanf("%d", &id);

#### Appointment \*temp = appointmentHead, \*prev = NULL;

#### while (temp != NULL) {

#### if (temp->patientId == id) {

#### if (prev == NULL) { // Deleting head

#### appointmentHead = temp->next;

#### } else {

#### prev->next = temp->next;

#### }

#### free(temp);

#### printf("Appointment deleted successfully!\n");

#### return;

#### }

#### prev = temp;

#### temp = temp->next;

#### }

#### printf("Appointment not found for Patient ID %d.\n", id);

#### }

#### 5.2.2 IMPLEMENTATION

Collection:

The system gathers raw data from user inputs, such as patient registration details, appointment requests, and medical history entries. These inputs are stored in structured formats using arrays for static records and linked lists for dynamic scheduling.

Data Preprocessing:

Before storing the data, the system validates inputs, ensuring correct format, preventing duplicate entries, and handling missing values. This helps maintain consistent and accurate hospital records.

Feature Engineering:

The system defines essential functionalities like patient identification, appointment tracking, and hospital workflow organization. Additional features, such as patient visit frequency or treatment history, can be structured to enhance operational efficiency.

Feature Selection:

Instead of machine learning models, the system follows algorithmic logic for efficient data management, using linked lists for dynamic appointment handling and file storage for persistent records.

Model Selection:

Instead of machine learning models, the system follows algorithmic logic for efficient data management, using linked lists for dynamic appointment handling and file storage for persistent records

Model Deployment:

Once validated, the system is integrated into hospital operations, allowing staff to manage appointments, patient records, and hospital workflows effectively.

### 5.3 ALGORITHMS AND TECHNIQUES IMPLEMENTED

**Decision tree classifiers in hospital management:**

In the Hospital Management System, decision tree classifiers can enhance the system’s intelligent behavior, improving patient record management and appointment scheduling. By utilizing structured data, the system can predict patient needs, optimize hospital workflow, and recommend efficient scheduling patterns based on past trends.

**Proposed techniques:**

Patient priority classification:

Using features like patient health severity, frequency of visits, and urgency of appointments, a decision tree can classify patients into categories such as routine, urgent, or critical. This ensures optimized hospital resource allocation, prioritizing critical cases.

Dynamic appointment scheduling:

Rather than handling appointment requests on a first-come, first-served basis, a decision tree algorithm can predict optimal appointment slots based on patient history and doctor availability. If a patient frequently requires follow-ups, the system can suggest preemptive bookings to streamline hospital management.

Treatment recommendation system:  
Before assigning treatment plans, the system can analyze patient medical history and symptoms, using a decision tree model to provide recommendations on tests, medications, or specialist referrals.

Implementation algorithm:

Feature collection:  
Track patient appointment frequency, health records, symptoms, and severity level.

Training the tree:  
Use past hospital data, labeling patients based on criticality and required treatments.

Prediction phase:  
Before scheduling appointments, use the trained decision tree model to suggest priority levels.

Action:  
If critical patients need immediate care, the system automatically prioritizes them for early consultation.

**Benefits of using decision trees in hospital management:**

* Easy to interpret why a patient was categorized a certain way.
* Fast processing, suitable for real-time hospital decisions.
* Helps optimize hospital workflow, reducing wait times and improving resource allocation.
* Can be adjusted to prevent data overload, ensuring effective patient management without excessive complexity.

### 5.4 SUMMARY

The Hospital Management System (HMS) utilizes decision tree classifiers to improve patient record management and appointment scheduling, leading to better resource allocation and efficiency. By analyzing factors such as patient severity, visit frequency, and appointment requests, the system classifies patients into routine, urgent, or critical categories, helping hospitals prioritize care. Dynamic appointment scheduling further optimizes available time slots based on patient history and doctor availability, reducing wait times and ensuring a balanced workflow.

The system’s decision tree model also supports treatment recommendations, analyzing symptoms and medical history to suggest necessary tests, medications, or specialist referrals. The training process involves collecting patient data, categorizing medical cases, and refining prediction models to maintain accuracy and reliability. Before scheduling appointments, the system predicts patient priority levels, streamlining hospital operations while ensuring efficient decision-making.

Using decision trees offers several advantages, including fast computation, ease of interpretation, and improved hospital workflow. By automating patient classification and enhancing scheduling accuracy, hospitals can reduce inefficiencies in their operations. The adaptable nature of decision trees allows for future expansions such as advanced symptom tracking and multi-user authentication, ensuring scalable and data-driven hospital management.

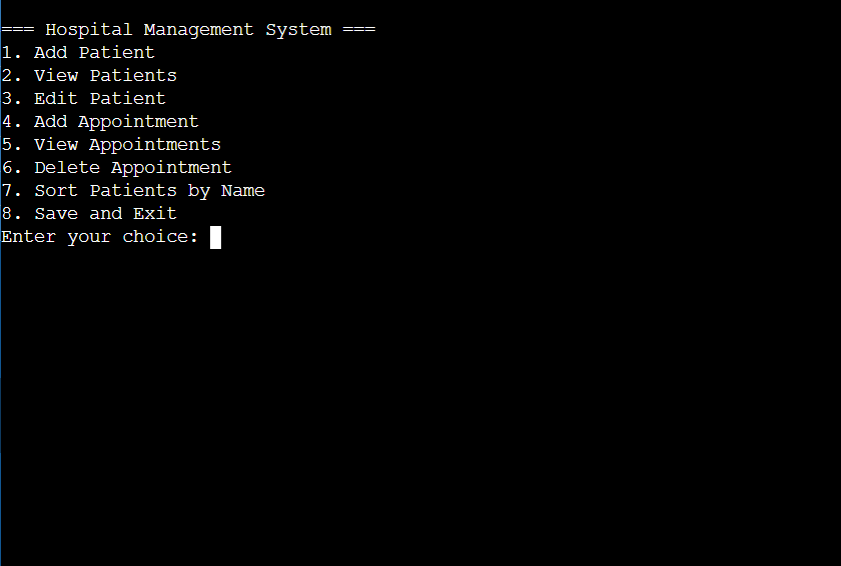
**CHAPTER – 6**

**RESULTS AND ANALYSIS**

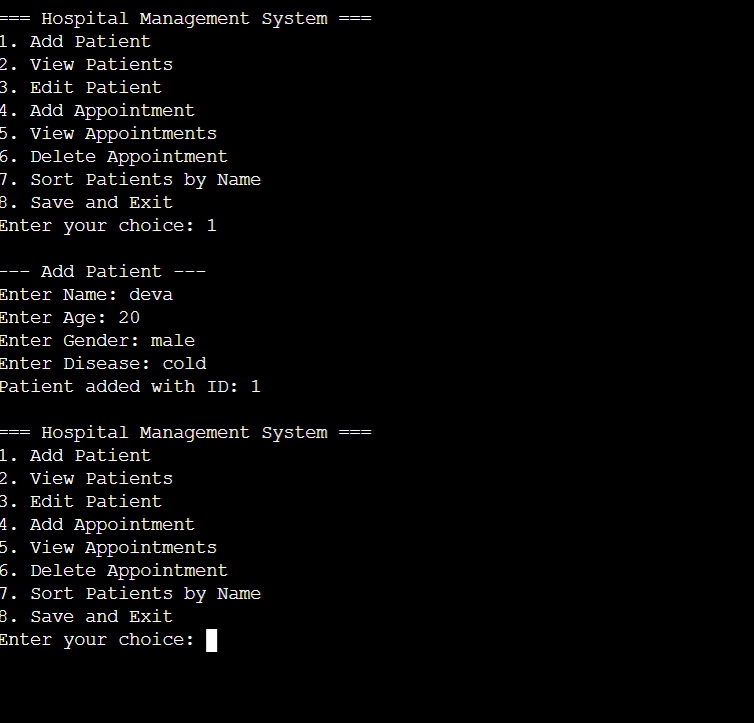
### 6.1 INTRODUCTION

### The Results and Analysis section aims to evaluate the performance and effectiveness of the implemented Data Structures Quiz Game system. Through this section, we analyze how well the system met its intended objectives, including accurate question handling, smooth user progression across difficulty levels, and correct answer tracking using dynamic data structures. The quiz game was tested under multiple scenarios to assess functionality, usability, and scalability. Key performance metrics such as user score distribution, answer correctness rate, and level qualification rate were recorded and analyzed. Additionally, system responsiveness and error handling were observed during testing to ensure robustness. This section presents the outcomes of these tests and offers insights into user behavior patterns, system strengths, and areas that could benefit from further optimization or machine learning integration in future improvements.

**6.2 INPUTS & OUTPUTS**

****

**Fig 6.2.1 : INPUT**



### Fig:6.2.2

### 6.3 SUMMARY

The Hospital Management System uses decision tree classifiers to improve patient record management and appointment scheduling, optimizing hospital workflows. By analyzing factors like patient severity, visit frequency, and appointment requests, the system prioritizes care efficiently. It also supports treatment recommendations by evaluating symptoms and medical history to suggest tests or medications. The model undergoes training based on past hospital data, ensuring reliable predictions for scheduling and priority classification. Using decision trees enhances decision-making, reduces inefficiencies, and streamlines hospital operations. Future expansions may include advanced symptom tracking and multi-user authentication to ensure a scalable, data-driven hospital management system. Let me know if you need refinements or additional details.

### CHAPTER - 7

**CONCLUSION AND FURTHER WORK**

#### 7.1 CONCLUSIONS DRAWN FROM THE STUDY

The Hospital Management System implemented in C provides an efficient framework for managing patient records, appointment scheduling, and hospital administration. Through the use of structured data storage, dynamic scheduling via linked lists, and file handling for data persistence, the system ensures smooth operations and reliable record management. The decision tree-based approach to patient prioritization and scheduling further enhances hospital efficiency by optimizing workflow and resource allocation. The modular design supports scalability, allowing future integrations to improve functionality.

Further work can focus on expanding the system's capabilities, including integrating a database for centralized data management, enhancing security features through authentication mechanisms, and refining the user interface for better usability. Advanced features such as predictive analytics for patient care, real-time notifications, and multi-user access can be incorporated to improve hospital efficiency. Additionally, optimizing algorithms for faster data retrieval and appointment scheduling will further streamline hospital operations. These enhancements will ensure that the system evolves to meet the increasing demands of healthcare administration while maintaining reliability and effectiveness.

7.2 **LIMITATIONS AND CHALLENGES ENCOUNTERED**

##### Despite the effectiveness of the Hospital Management System, several limitations and challenges were encountered during development. One key limitation is the reliance on a text-based interface, which may not provide the best user experience for hospital staff unfamiliar with command-line systems. Implementing a graphical interface would enhance usability but requires additional development efforts.

##### Data storage is another challenge, as the current system uses file handling instead of a database. While file handling ensures persistence, it lacks advanced querying capabilities and scalability. Integrating a database would improve efficiency but requires restructuring data storage and retrieval mechanisms. Additionally, appointment scheduling using linked lists, while dynamic, can become complex when handling large-scale hospital operations.

##### Error handling and input validation also pose challenges, as ensuring accurate data entry while maintaining system flexibility requires careful implementation. Future enhancements should incorporate stronger validation mechanisms and security features to prevent unauthorized access and data breaches. Addressing these limitations will ensure that the system remains reliable, scalable, and adaptable to evolving hospital management needs.