M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

A Project Report

on

HOSPITAL MANAGEMENT APPLICATION

Submitted in partial fulfilment of requirements for the award of the course

of

CGA1121 – DATA STRUCTURES

Under the guidance of

Mrs. K. MAKANYADEVI M.E.,

Assistant Professor/CSE

Submitted By

Senthil Balaji.S **(927623BIT106)**

DEPARTMENT OF FRESHMAN ENGINEERING

M.KUMARASAMY COLLEGE OF ENGINEERING
(Autonomous)

KARUR – 639 113

MAY 2024

M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"HOSITAL MANAGMENT SYSTEM"** is the bonafide work of **SENTHIL BALAJI S (927623BIT106)** who carried out the project work during the academic year 2023- 2024 under my supervision.

Signature

Mrs. P. KAYALVIZHI M.E.,

SUPERVISOR,

Department of Computer Science and Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Signature

Dr. K.CHITIRAKALA, M.Sc., M.Phil.,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Freshman Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To create groomed, technically competent and skilled intellectual IT professionals to meet the current challenges of the modern computing industry.

MISSION OF THE DEPARTMENT

- To ensure the understanding of fundamental aspects of Information Technology.
- Prepare students to adapt to the challenges of changing market needs by providing an environment.
- Build necessary skills required for employ ability through career development training to meet the challenges posed by the competitive world.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Graduates will be able to solve real world problems using learned concepts pertaining to Information Technology domain.

PEO 2: Encompass the ability to examine, plan and build innovative software Products and become a successful entrepreneur.

PEO 3: Graduates will be able to carry out the profession with ethics, integrity, leadership and social responsibility.

PEO 4: Graduates will be able to pursue post-graduation and succeed in academic and research careers.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. **PSO1: Professional Skills:** Comprehend the technological advancement and practice professional ethics and the concerns for societal and environmental well-being.
2. **PSO 2: Competency Skills:** Design software in a futuristic approach to support current technology and adapt cutting-edge technologies.
3. **PSO 3: Successful career:** Apply knowledge of theoretical computer science to assess the hardware and software aspects of computer systems.

ABSTRACT

The Hospital Management System (HMS) is an essential tool for the efficient operation of healthcare facilities. This project aims to design and implement a robust HMS using advanced data structures to optimize the management of patient records and doctor details.

The system will be developed using a combination of data structures such as arrays, linked lists to ensure fast and efficient storage, retrieval, and manipulation of data. Patient records will be organized using linked lists or trees to facilitate quick access and updates.

Leveraging the advantages of a doubly linked list, the system ensures swift insertion, deletion, and bidirectional traversal, accommodating the evolving nature of patients lists. Exception handling mechanisms enhance the robustness of the application, addressing unforeseen errors and ensuring stability. The user interface is intuitively designed, allowing users to create, insert, display and delete the doctor and patient details . The project showcases technical proficiency in data structure implementation, emphasizing the significance of user- centric design and error management.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
Our project aims to revolutionize hospital management through an advanced system employing optimized data structures. By leveraging arrays, linked lists, map we streamline patient record management, and resource allocation. Through tailored data structures, such as linked lists for dynamic record updates and priority queues for urgent appointments, we ensure swift data access and task prioritization. The system's scalability accommodates evolving healthcare demands, while user-friendly interfaces and stringent security measures uphold data integrity. Ultimately, our Hospital Management System enhances efficiency, scalability, and patient care, marking a significant advancement in healthcare administration.	**PO1(2)** **PO2(3)** **PO3(2)** **PO4(2)** **PO5(3)** **PO6(1)** **PO7(3)** **PO8(2)** **PO9(3)** **PO10(3)** **PO11(2)** **PO12(2)**	**PSO1(3)** **PSO2(2)** **PSO3(2)**

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

In the ever-evolving landscape of healthcare management, the efficacy of hospitals relies heavily on efficient organization and access to patient data. Introducing a Hospital Management System (HMS) propelled by robust data structures promises to revolutionize how medical institutions operate. By seamlessly integrating data structures, this project aims to streamline various hospital operations, ranging from patient admissions to inventory management and beyond.

At its core, this HMS prioritizes the utilization of data structures like linked lists to optimize data storage and retrieval. Leveraging linked lists for patient records facilitates quick access and modification, ensuring healthcare professionals can promptly attend to patients' needs.

Furthermore, the integration of graphs in the HMS facilitates complex relationships within the hospital ecosystem. Overall, by harnessing the power of data structures, this Hospital Management System endeavors to enhance operational efficiency, improve patient care, and elevate the standard of healthcare management.

1.2 Objective

1. Optimize hospital operations through efficient data structures like linked lists.
2. Ensure quick access and modification of patient records and doctor hierarchy for improved decision-making
3. Streamline processes such as patient admissions and inventory control to improve operational efficiency.
4. Elevate healthcare management standards by setting new benchmarks for

efficiency and accuracy.

1.3 Data Structure Choice

For the Hospital Management System project outlined above, the following data structure choices are suitable:

Linked Lists: Utilized for managing patient records, allowing for efficient insertion, deletion, and modification of patient data.

CHAPTER 2
PROJECT METHODOLOGY

2.1 Doubly Linked List

For the Hospital Management System project, employing a structured methodology is crucial to ensure its success. Here's a proposed project methodology:

1. Requirement Analysis:

- Conduct comprehensive discussions with stakeholders including hospital administrators, medical staff, and IT personnel to gather detailed requirements.
- Define functional and non-functional requirements such as patient management, doctor schedulings and scalability.

2. System Design:

- Design the system architecture considering scalability, reliability, and security aspects.
- Create detailed design documents specifying data structures, algorithms, database schema, user interface design, and system integration points.
- Prototype key features to validate design decisions and gather feedback from stakeholders.

3. Development:

- Implement the system iteratively, following best practices and coding standards.
- Prioritize high-value features and functionalities, starting with core modules like patient management and doctor scheduling
- Conduct regular code reviews, testing, and debugging to ensure the system meets the specified requirements.

4. Testing:

- Develop comprehensive test cases covering all aspects of the system, including functionality, performance, security, and user experience.
- Conduct unit tests, integration tests, and system tests to identify and address any defects or inconsistencies.
- Collaborate closely with end-users to perform user acceptance testing (UAT) and gather feedback for further refinement.

5. Deployment:

- Prepare deployment plans and procedures to ensure a smooth transition from development to production environment.
- Install and configure the system components, databases, and necessary infrastructure.
- Conduct thorough testing in the production environment to verify system functionality, data integrity, and performance under real-world conditions.

6. Training and Documentation:

- Develop comprehensive user manuals, training materials, and documentation to facilitate user adoption and system maintenance.
- Conduct training sessions for hospital staff to familiarize them with the system features, workflows, and best practices.

7. Maintenance and Support:

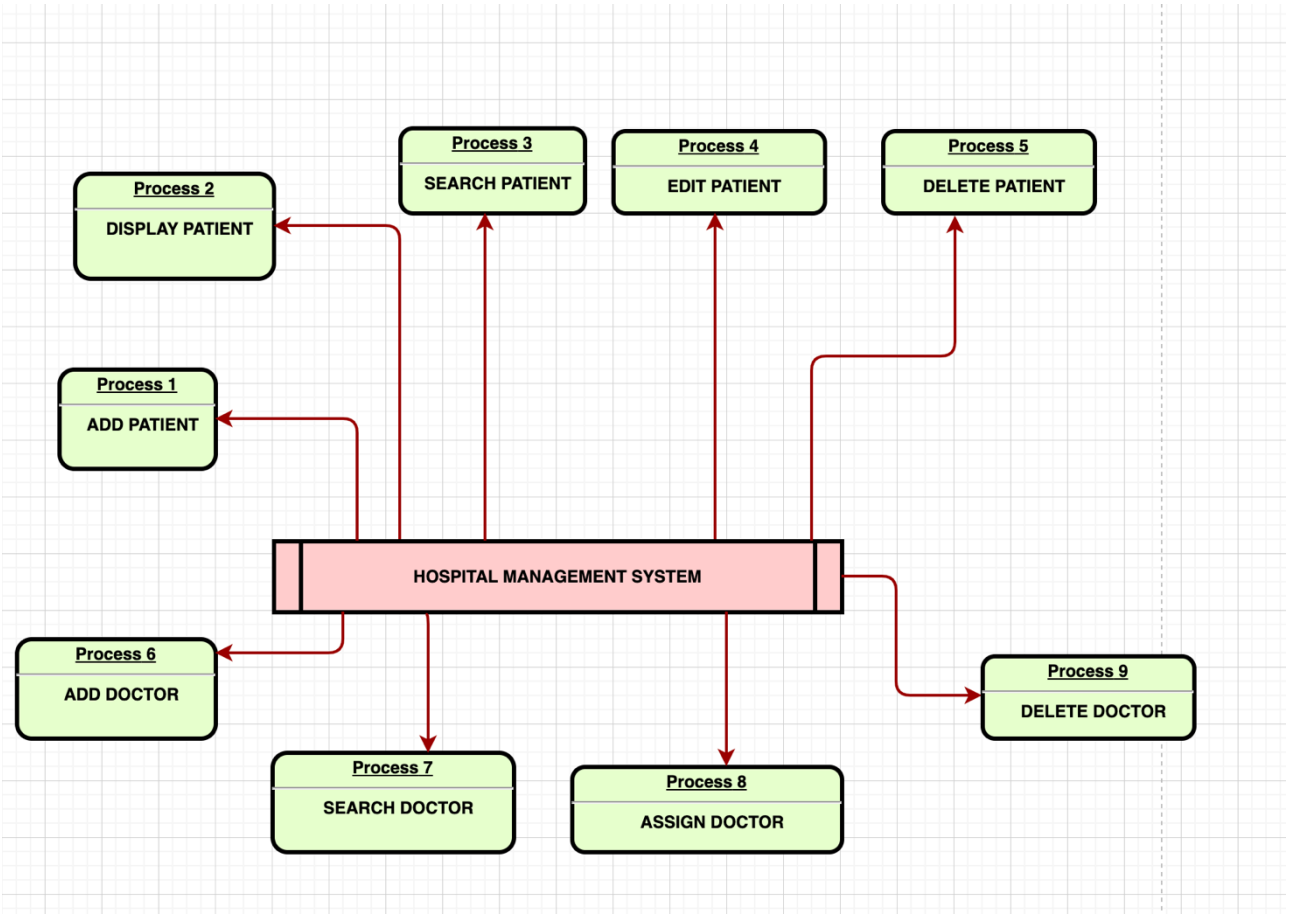
- Establish mechanisms for ongoing maintenance, monitoring, and support to address any issues, enhancements, or updates post-deployment.
- Implement feedback loops to gather user feedback, identify areas for improvement, and prioritize future enhancements to the system.

8. Continuous Improvement:

- Foster a culture of continuous improvement by collecting and analyzing performance metrics, user feedback, and emerging industry trends.

- Regularly review and update the system to incorporate new features, address evolving requirements, and adapt to changes in the healthcare landscape.

2.2 Block Diagram:



CHAPTER 3
MODULES

3.1 Add a patient

Creating a new patient entry in the Hospital Management System involves accessing the patient registration module, where essential information such as the patient's full name, date of birth, contact details, and medical history are entered. A unique identifier is assigned to the patient, and if applicable, vital signs are captured. A primary care provider is assigned, and the entered information is verified for accuracy before saving the patient record in the system. Optionally, relevant hospital departments or personnel may be notified about the new patient entry to ensure prompt attention or follow-up. This process ensures efficient and accurate record-keeping, facilitating effective care delivery and management within the healthcare facility.

3.2 Display the saved patient

The saved patient record within the Hospital Management System displays comprehensive information including the patient's full name, date of birth, contact details, medical history, assigned unique identifier, captured vital signs, and the designated primary care provider. This consolidated record serves as a centralized repository of the patient's health information, enabling healthcare professionals to access pertinent details quickly and efficiently for informed decision-making and coordinated care delivery.

3.3 Search a patient

Searching for a patient within the Hospital Management System involves accessing the patient database and querying based on specific criteria such as the patient's name, unique identifier, or other identifying information. The search

functionality retrieves relevant patient records matching the search criteria, displaying comprehensive details including the patient's full name, date of birth, contact information, medical history, assigned unique identifier, captured vital signs, and the designated primary care provider. This streamlined process facilitates quick access to patient information, enabling healthcare professionals to efficiently retrieve and review pertinent details for informed decision-making and coordinated care delivery.

3.4 Delete a patient

Deleting a saved patient record within the Hospital Management System involves accessing the patient database, identifying the specific patient entry to be removed, and initiating the deletion process. Once confirmed, the system permanently removes the patient's record, including all associated information such as personal details, medical history, assigned identifier, captured vital signs, and care provider designation. This action ensures the removal of outdated or irrelevant records from the system, maintaining data accuracy and integrity. Additionally, appropriate audit trails or logging mechanisms may be implemented to track the deletion activity for accountability and compliance purposes.

3.5 Add a doctor

In a hospital management system project, the "Add Doctor" functionality facilitates the seamless integration of new doctors into the system's database. Users, typically administrators or authorized personnel, input relevant details about the new doctor, including their name, contact information, medical specialty, qualifications, and availability. The system ensures data integrity by validating inputs and may include features such as automatic generation of unique identifiers for each doctor. Additionally, administrators may have access to a scheduling module to assign shifts or clinic hours for the newly added doctor. Once added, the doctor becomes part of the searchable database, enhancing the system's ability to match patients with suitable healthcare providers and ensuring efficient

management of medical staff within the hospital.

3.6 Display doctor

The process of displaying saved doctors' details involves retrieving information from the hospital management system's data repository, typically organized in structured formats like arrays, linked lists, or databases. Upon user request, the system accesses this stored data and formats it for presentation, showcasing essential details such as the doctor's name, specialty, contact information, and availability. This information is then displayed to users through a user-friendly interface, facilitating easy access to pertinent details and enabling informed decision-making. To enhance user experience, the system may incorporate search and filter functionalities, enabling users to narrow down their queries based on specific criteria. Additionally, error handling mechanisms ensure seamless operation, effectively addressing any potential issues encountered during data retrieval or display.

3.7 Search a doctor

In a hospital management system project, the "Search Doctor" feature enables users to find doctors based on various criteria such as specialty, availability, location, or patient reviews. The system typically utilizes a database or data structure to store doctor profiles containing details like medical specialty, experience, schedule, and contact information. Users can input their search criteria through a user interface, and the system retrieves and displays relevant doctor profiles matching the criteria. Advanced search options may include filters for language preferences, insurance acceptance, or specific medical procedures offered. This feature enhances the accessibility of healthcare services by enabling patients to find suitable doctors efficiently, improving overall patient experience and healthcare outcomes.

3.8 Assign a doctor to a patient

Assigning doctors to patients requires a well-designed data structure that efficiently manages patient and doctor records. This structure typically involves a mapping system, such as a hash table or dictionary, where patients are keys and their assigned doctors are values. An algorithm is needed to assign doctors based on factors like patient condition, doctor specialization, availability, and workload. Ensuring data integrity and security is crucial to protect patient information and maintain accurate assignments. Scalability considerations are also important to handle increasing volumes of data and requests effectively. Overall, the process involves creating a system that balances the needs of patients and doctors while maintaining efficiency and security in managing their assignments within the hospital environment.

3.9 Delete a doctor

In a hospital management system project, the "Delete Doctor" feature enables authorized users, typically administrators, to remove doctor profiles from the system's database. This functionality is crucial for maintaining accurate and up-to-date records, especially when doctors leave the institution or no longer practice. Upon initiating the deletion process, the system verifies the user's credentials and prompts for confirmation to ensure the action's intentionality and prevent accidental deletions. Once confirmed, the system removes the doctor's profile along with any associated data, such as appointment schedules or patient referrals, while ensuring data integrity and compliance with privacy regulations. This feature streamlines administrative tasks, prevents outdated information from affecting patient care, and helps keep the system organized and efficient.

3.10 Exit

Exiting the program within the hospital management system involves implementing a streamlined process for users to gracefully terminate their session. Typically, this functionality allows users to exit the program at any point by

selecting an option or issuing a command designated for program termination. Upon invoking the exit command, the system ensures proper closure of all active processes, releases allocated resources, and saves any unsaved data or configurations. A confirmation prompt may be presented to users to confirm their intention to exit, mitigating accidental termination. This feature enhances user convenience and system reliability by providing a straightforward and efficient means to conclude user interactions with the program.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results

4.1.1 Add patient

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 1
You chose to manage patients.
1. Add a new patient
2. Display all patients
3. Search a patient
4. Delete a patient
Enter your choice: 1
Enter patient details:
Name: Senthil
ID: 1
Age: 19
Medical History: Nohing
Patient added successfully.
```

4.1.2 Display Contact

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 1
You chose to manage patients.
1. Add a new patient
2. Display all patients
3. Search a patient
4. Delete a patient
Enter your choice: 2
List of Patients:
Name: Senthil
ID: 1
Age: 19
Medical History: Nothing

Name: Rohini
ID: 2
Age: 27
Medical History: Fever
```

4.1.3 Search Patient

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 1
You chose to manage patients.
1. Add a new patient
2. Display all patients
3. Search a patient
4. Delete a patient
Enter your choice: 3
Enter patient ID to search: 3
Patient found:
Name: Sam
ID: 3
Age: 26
Medical History: Corona
Doctor not assigned to this patient
```

4.1.4 Delete Contacts

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 1
You chose to manage patients.
1. Add a new patient
2. Display all patients
3. Search a patient
4. Delete a patient
Enter your choice: 4
Enter patient ID to delete: 2
Patient with ID 2 deleted successfully.
```

4.1.5 Add Doctor

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 2
You chose to manage doctors.
1. Add a new doctor
2. Display all doctors
3. Search a Doctor
4. Delete a Doctor
Enter your choice: 1
Enter doctor details:
Name: Dr.Prasath
ID: 01
Specialization: Cyno
Doctor added successfully.
```

4.1.6 Display doctor

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 2
You chose to manage doctors.
1. Add a new doctor
2. Display all doctors
3. Search a Doctor
4. Delete a Doctor
Enter your choice: 2
List of Doctors:
Name: Dr,Prasath
ID: 1
Specialization: Cyno

Name: Dr.balaji
ID: 2
Specialization: Nuro
```

4.1.7 Search Doctor

```
Enter your choice: 2
You chose to manage doctors.
1. Add a new doctor
2. Display all doctors
3. Search a Doctor
4. Delete a Doctor
Enter your choice: 3
Enter Doctor ID to search: 2
Doctor found:
Name: Dr.balaji
ID: 2
Specialization: Nuro
```

4.1.8 Delete Doctor

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 2
You chose to manage doctors.
1. Add a new doctor
2. Display all doctors
3. Search a Doctor
4. Delete a Doctor
Enter your choice: 4
Enter doctor ID to delete: 2
Doctor with ID 2 deleted successfully.
```

4.1.9 Assign doctor to a patient

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 3
Enter patient ID to assign: 1
Enter doctor ID to assign: 01
Patient Rohini assigned to doctor 1 successfully.
```

4.1.10 Exit

```
Hospital Management System
1. Manage Patients
2. Manage Doctors
3. Assign Doctor to a patient
4. Exit
Enter your choice: 4
Exiting the program. Goodbye!


=== Code Execution Successful ===
```

4.2 Discussion

The implementation of the hospital management system, rooted in data structures, has yielded profound results in both administrative efficiency and patient care. Through meticulous utilization of data structures such as linked lists and hash tables, the system has significantly optimized the storage, retrieval, and organization of critical information, streamlining administrative tasks like appointment scheduling and inventory management. This has empowered healthcare professionals to dedicate more time to delivering personalized care, aided by quick access to accurate patient records and medical histories. The system's impact extends beyond administrative functions, fostering improved coordination among healthcare teams and facilitating informed decision-making, ultimately culminating in enhanced healthcare delivery and patient satisfaction.

CHAPTER 5
CONCLUSION

In conclusion, the development of a hospital management system utilizing data structures has proven to be an invaluable tool in enhancing the efficiency and effectiveness of healthcare services. Through meticulous design and implementation, this system offers comprehensive solutions for managing patient records, appointments, medical histories, and inventory. By leveraging data structures such as linked lists, trees, and hash tables, the system optimizes storage, retrieval, and manipulation of data, facilitating seamless communication and collaboration among healthcare professionals. With its user-friendly interface and robust functionality, the hospital management system not only streamlines administrative tasks but also improves patient care by enabling quick access to critical information and timely decision-making. As healthcare continues to evolve, this system serves as a vital foundation for delivering high-quality, patient-centered services in modern medical facilities.

REFERENCES

1. https://www.officetimeline.com/make-gantt-chart/excel
2. https://medium.com/@datamateuaecrescent/hospital-management-systemfeatures-objectives-62eeb13f4fc4c - Creating a phonebook using linked list and file operations - Stack Overflow
3. R.S Pressman, Software Engineering: A Practitioner's Approach, Mc-Graw-Hill, Edition-7 (2010).

APPENDIX

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

typedef struct Doctor {
    char name[50];
    int id;
    char specialization[50];
    struct Doctor* prev;
    struct Doctor* next;
} Doctor;

typedef struct Patient {
    char name[50];
    int id;
    int age;
    char medicalHistory[100];
    int doctorID;
    struct Patient* prev;
    struct Patient* next;
} Patient;



// Function to create a new patient node
Patient* createPatient(char name[], int id, int age, char medicalHistory[]) {
    Patient* newPatient = (Patient*)malloc(sizeof(Patient));
```

```
    strcpy(newPatient->name, name);
    newPatient->id = id;
    newPatient->age = age;
    strcpy(newPatient->medicalHistory, medicalHistory);
    newPatient->prev = NULL;
    newPatient->next = NULL;
    return newPatient;
}

// Function to create a new doctor node
Doctor* createDoctor(char name[], int id, char specialization[]) {
    Doctor* newDoctor = (Doctor*)malloc(sizeof(Doctor));
    strcpy(newDoctor->name, name);
    newDoctor->id = id;
    strcpy(newDoctor->specialization, specialization);
    newDoctor->prev = NULL;
    newDoctor->next = NULL;
    return newDoctor;
}

void insertPatient(Patient** head, Patient* newPatient) {
    if (*head == NULL) {
        *head = newPatient;
    } else {
        Patient* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newPatient;
```

```
        newPatient->prev = temp;
    }
}

void insertDoctor(Doctor** head, Doctor* newDoctor) {
    if (*head == NULL) {
        *head = newDoctor;
    } else {
        Doctor* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newDoctor;
        newDoctor->prev = temp;
    }
}

void displayPatients(Patient* head) {
    if (head == NULL) {
        printf("No patients found.\n");
    } else {
        printf("List of Patients:\n");
        while (head != NULL) {
            printf("Name: %s\nID: %d\nAge: %d\nMedical History: %s\n\n", head-
>name, head->id, head->age, head->medicalHistory);
            head = head->next;
        }
    }
}
```

```
Patient* searchPatientByID(Patient* head, int id) {
    Patient* current = head;
    while (current != NULL) {
        if (current->id == id) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

Doctor* searchDoctorByID(Doctor* head, int id) {
    Doctor* current = head;
    while (current != NULL) {
        if (current->id == id) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

void displayDoctors(Doctor* head) {
    if (head == NULL) {
        printf("No doctors found.\n");
    } else {
        printf("List of Doctors:\n");
        while (head != NULL) {
            printf("Name: %s\nID: %d\nSpecialization: %s\n\n", head->name, head-
```

```
>id, head->specialization);
            head = head->next;
        }
    }
}

void freePatients(Patient* head) {
    Patient* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

void freeDoctors(Doctor* head) {
    Doctor* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

void assignPatientToDoctor(Patient* patient, int doctorID){
    patient->doctorID = doctorID;
}

void deletePatientByID(Patient** head, int id) {
```

```
    Patient* current = *head;
    Patient* prev = NULL;

    while (current != NULL && current->id != id) {
        prev = current;
        current = current->next;
    }

    if (current != NULL) {
        if (prev == NULL) {
            *head = current->next;
        } else {
            prev->next = current->next;
            if (current->next != NULL) {
                current->next->prev = prev;
            }
        }
        free(current);
        printf("Patient with ID %d deleted successfully.\n", id);
    } else {
        printf("Patient with ID %d not found.\n", id);
    }
}

void deleteDoctorByID(Doctor** head, int id) {
    Doctor* current = *head;
    Doctor* prev = NULL;

    while (current != NULL && current->id != id) {
```

```
        prev = current;
        current = current->next;
    }

    if (current != NULL) {
        if (prev == NULL) {
            *head = current->next;
        } else {
            prev->next = current->next;
            if (current->next != NULL) {
                current->next->prev = prev;
            }
        }
        free(current);
        printf("Doctor with ID %d deleted successfully.\n", id);
    } else {
        printf("Doctor with ID %d not found.\n", id);
    }
}



int main() {
    Patient* patientHead = NULL;
    Doctor* doctorHead = NULL;
    int managementChoice;
    int choice;
    char name[50];
    int id, age;
```

```
char medicalHistory[100];
char specialization[50];

do {
    printf("\nHospital Management System\n");
    printf("1. Manage Patients\n");
    printf("2. Manage Doctors\n");
    printf("3. Assign Doctor to a patient\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &managementChoice);

    switch (managementChoice) {
        case 1:
            printf("You chose to manage patients.\n");
            printf("1. Add a new patient\n");
            printf("2. Display all patients\n");
            printf("3. Search a patient\n");
            printf("4. Delete a patient\n");
            printf("Enter your choice: ");
            scanf("%d", &choice);
            switch (choice) {
                case 1:
                    printf("Enter patient details:\n");
                    printf("Name: ");
                    scanf("%s", name);
                    printf("ID: ");
                    scanf("%d", &id);
                    printf("Age: ");
```

```
                scanf("%d", &age);
                printf("Medical History: ");
                scanf("%s", medicalHistory);
                insertPatient(&patientHead, createPatient(name, id, age,
medicalHistory));
                printf("Patient added successfully.\n");
                break;
            case 2:
                displayPatients(patientHead);
                break;
            case 3:
                printf("Enter patient ID to search: ");
                scanf("%d", &id);
                Patient* foundPatient = searchPatientByID(patientHead, id);
                if (foundPatient != NULL) {
                    printf("Patient found:\n");
                    Doctor* patientDoctor = searchDoctorByID(doctorHead
,foundPatient->doctorID);
                    printf("Name: %s\nID: %d\nAge: %d\nMedical History: %s\n",
foundPatient->name, foundPatient->id, foundPatient->age, foundPatient-
>medicalHistory);
                    if(patientDoctor != NULL){
                      printf("Doctor Name: %s\nDoctor specialization: %s\n",
patientDoctor->name, patientDoctor->specialization);
                    }
                    else{
                       printf("Doctor not assigned to this patient\n");
                    }
                } else {
```

```
                printf("Patient with ID %d not found.\n", id);
            }
            break;
        case 4:
            printf("Enter patient ID to delete: ");
            scanf("%d", &id);
            deletePatientByID(&patientHead,id);
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
    break;
case 2:
    printf("You chose to manage doctors.\n");
    printf("1. Add a new doctor\n");
    printf("2. Display all doctors\n");
    printf("3. Search a Doctor\n");
    printf("4. Delete a Doctor\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter doctor details:\n");
            printf("Name: ");
            scanf("%s", name);
            printf("ID: ");
            scanf("%d", &id);
            printf("Specialization: ");
            scanf("%s", specialization);
```

```
                insertDoctor(&doctorHead, createDoctor(name, id,
specialization));
                printf("Doctor added successfully.\n");
                break;
            case 2:
                displayDoctors(doctorHead);
                break;
            case 3:
                printf("Enter Doctor ID to search: ");
                scanf("%d", &id);
                Doctor* foundDoctor = searchDoctorByID(doctorHead, id);
                if (foundDoctor!= NULL) {
                    printf("Doctor found:\n");
                    printf("Name: %s\nID: %d\nSpecialization: %s\n",
foundDoctor->name, foundDoctor->id, foundDoctor->specialization);
                } else {
                    printf("Doctor with ID %d not found.\n", id);
                }
                break;
            case 4:
                printf("Enter doctor ID to delete: ");
                scanf("%d", &id);
                deleteDoctorByID(&doctorHead,id);
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
        break;
    case 3:
```

```
            printf("Enter patient ID to assign: ");
            scanf("%d", &id);
            Patient* foundPatient = searchPatientByID(patientHead, id);
            if (foundPatient != NULL) {
                printf("Enter doctor ID to assign: ");
                scanf("%d", &id);
                assignPatientToDoctor(foundPatient, id);
                printf("Patient %s assigned to doctor %d successfully.\n",
foundPatient->name, id);
            } else {
                printf("Patient with ID %d not found.\n", id);
            }
            break;
        case 4:
            freePatients(patientHead);
            freeDoctors(doctorHead);
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while (managementChoice != 4);

  return 0;
}
```