**PROJECT REPORT ON “Hospital Management System ”**

By

Nurjahan Akter (IT23045)

IT-23045

Session 2022

Under the Supervision of

**Dr. Ziaur Rahman**

**Associate Professor**

**Department of Information and Communication Technology**



**Department of Information and Communication Technology**

**Mawlana Bhashani Science and Technology University**

Santosh, Tangail – 1902 , Bangladesh

October ,2024

1

--------------------------------------------------------------------------------------------------------------------------

**Table of Contents**

**1. Introduction** ………………………..……………………………………………………………………………….. 04

1.1 Aim ………………………..…………………………………………………. 05

1.2 Objective……….……………………….…………………………………. 05

1.3 Summary …………………………….……………………………………. 06

**2**. **Design & System Thinking**…….….……………………………..………..06

2.1 Introduction to C………………….…………………………………….. 06

2.2 Problem define…………………….……………………………………. 07

2.3 System thinking approach……….……………………………………07

2.4 Methodology ……………….………………………………………………07

2.5 System requirement…………………………………………………….07

**3. Implementation** ………………..……………………………………… ……………………………………………..08

3.1 Overview of the code structure.……………………………………08

3.2 Global variable. ………………………………….…………………………08

3.3 Main function ……………………………………………………………….09

3.4 Question handling………………………………………………………….09

3.5 Error handling…………………………………………………………………09

**4.** **Testing** ……………………………………………………….……….………..……………………………………………..10

4.1 Unit testing ……………….………………………….………….…………….10

4.2 Function testing……………………………………..……...………………..10

4.3 Boundary testing……………………………………………………………….11

4.4 User experience testing……………………………………………………..12

**5. Conclusion** …………………………………………….………………….................12

5.1 Draw back and limitations …………….…………..……………………….12

2

Declaration

I hereby declare that this report on the Hospital Management System is my original work, created as part of my academic requirements. The information presented here is accurate to the best of my knowledge, and any references used are duly acknowledged.

3

Acknowledgment

I express my heartfelt gratitude to my supervisor, \*\*Dr. Ziaur Harman\*\*, for his invaluable guidance, support, and encouragement throughout this project. His expertise and constructive feedback were instrumental in the successful completion of this work. Thank you for your mentorship and dedication.

4

Chapter 1

Introduction

The Hospital Management System is a project focused on handling patient data management in any medical facility. It is designed using C and provides automation in certain fields such as admitting patients, discharging, and billing by easy management of records, which includes searching and updating. The paperwork is reduced to a minimum, administrative burdens are reduced, hence making access to data easier, efficient, and enhancing the management of healthcare services accordingly.

* 1. Aim

The aim of this project is to design and implement a simple, yet effective, \*\*Hospital Management System (HMS)\*\* using the C programming language. This HMS software is tailored for small to mid-sized healthcare facilities to facilitate the management and handling of patient information in a structured manner. By digitizing patient records, the system reduces administrative overhead, minimizes paperwork, and streamlines essential tasks such as patient admission, discharge, billing, and record-keeping. Ultimately, the goal of this project is to provide an efficient, reliable, and accessible platform for healthcare administrators to manage patient data and processes.

* 1. Objectives

Following are the objectives of the project on the Hospital Management System:

1. \*\*Patient Record Management\*\*: Design a structured data system for storing all the primary information about patients, such as personal information, case history, visit type, dates of admission, and billing information.  
  
2. \*\*Admission and Discharge Management\*\*: Automate patient admissions and discharges by capturing relevant information, computing hospital stay and thereby arming with all the relevant billing information as per the visit types.  
3. \*\*Search and Update Capabilities\*\*: Provide the user with the ability to search for patient information by ID quickly and retrieve it; an interface to update the records when necessary for further treatment.   
 5  
4. \*\*Billing Functionality\*\*: Add a billing module wherein charges of patients get pre-estimated based on admission type, stay duration, among other parameters, thereby simplifying the financial tracking.  
  
5. \*\*Easy-to-Use Interface\*\*: Design an easy-to-use and intuitive menu-driven interface whereby the user is helped through several functions of the system to as low an input level as possible; this will reduce the learning curve for healthcare workers using the system.

* 1. Overview  
       
     Some of the key functions in healthcare facilities pertain to the management of patient information; however, with increased patient flow, the manual management system is often characterized by delays, errors, and inefficiencies. The Hospital Management System targets these challenges by targeting the introduction of a system that automates core functions regarding patient admission, discharge, and billing.  
       
     This HMS project is designed to handle a maximum of 100 patients. Features that will be provided include the addition of new patient records, showing of stored information, and editing or discharge of the patients when necessary. The design provides for a record-keeping structure that captures important information about a patient, such as patient ID, age, type of visit, and disease information, and stores it in a retrievable and modifiable fashion. The project has covered patients, even in terms of financial coverage, since it includes a billing module that calculates the fees of the patients in terms of admission and daily visitors.  
       
     The system will thus provide better organization of data, easy access to records, and smooth administration of small- and medium-sized healthcare facilities. HMS is developed in the C Language, using effective memory handling, structure-based data management, and basic control functions. The HMS solution thus offers an effective, lightweight tool for small hospital management.

Chapter 2

. Design & System Thinking

2.1 Introduction to C  
  
The C is a general-purpose programming language that is best suited for developing structured and

6

efficient applications that can give users direct access to system resources. The use of functions, data structures, and libraries makes it straightforward to enable modular programming. As a matter of fact, it is one of the very reasons for developing the HMS project because it requires handling records, data manipulation, and processing efficiently.

2.2 Problem Statement  
  
Manual handling of patient-related information in any healthcare organization involves so much time wastage and is full of errors. General issues revolve around inefficient retrieval of data, inaccurate billing, and non-availability of any smooth discharge processes that result in delays for care and dissatisfaction among patients, with administrative loads in facilities attending to a large number of patients. This project, therefore, targets the address of these issues through the design and implementation of an HMS characterized by structured record management, automated billing, and easy information retrieval.  
  
2.3 System Thinking Approach  
  
A system thinking approach looks at the Hospital Management System as an integrated set of elements interdependent upon others, performing in cooperation to meet the goal of efficiently managing patient information. This system-oriented approach will make sure that every function follows from and serves the overall objectives of enhancing accuracy in patient records, accessibility of data, and optimization of workloads. We will design the system to be dynamic in response to changing needs and interact seamlessly for a robust and holistic tool of management.

2.4 Methodology  
  
The project would follow a modular approach wherein the HMS would be divided into smaller feasible modules, which would then be integrated into one workable model. The major steps would include:  
  
1. \*\*Requirements Analysis\*\*: Determining the functions related to the management of patients, admission, billing, and maintenance of records.  
2. \*\*Design\*\*: Designing the structure of data in C using structures and functions for manipulating patient information in an organized fashion and assuring modularity.  
3. \*\*Implementation\*\*: Coding for each module, such as adding of records, search, edit, and discharge will be added in iterative refinement of functionality.  
4. \*\*Testing\*\*: Testing each module individually for correctness, followed by the integration test of the whole system.  
5. \*\*Deployment and Documentation\*\*: Ensuring the software meets the requirements of the user and documentation is adequate for maintenance and future enhancements..

\*\*2.5 System Requirements\*\*  
  
The Hospital Management System will be a console-based application, which requires the following:

7  
  
\*\*Hardware Requirements:\*\*  
- Minimum 2GB RAM  
- Processor speed of at least 1GHz  
- 10MB storage space for persisting data  
  
\*\*Software Requirements:\*\*  
- Operating System: The system should support a C compiler and hence shall work on Windows, Linux, and any system that has a C compiler.  
- Compiler: GCC is preferred but any compiler which can compile C programs is good to go.  
- Libraries Used: Standard libraries include `stdio.h`, `stdlib.h`, and `string.h`.  
  
They ensure HMS works flawlessly in normal computing environments for efficient patient management features in healthcare settings.

Chapter 3

------------------------------------------------------------------------------------------------------------------------------------------

Implementation

3.1 Overview of Code Structure  
  
The HMS code is segmented into various functions, with each function operating in a particular manner: adding patient records, displaying data, searching, and billing. The code has been modularized to give a separation of the core functionalities of an easily readable and maintainable code. Core modules include patient data handling functionalities: add, search, modify, and discharge, with records navigation; billing, orchestrated by the main menu.  
  
3.2 Global Variables  
  
The system utilizes global variables to control data across the application between different functions. A few of the important global variables are discussed below:  
  
- `struct Patient patients[MAX\_PATIENTS]`: Array of structure of patients where each element will have a record of patient id, name, age, disease and visit type.  
- `int patientCount`: Keeps a count of the records in current patients with a view to preventing the system from allowing more than `MAX\_PATIENTS`.  
  
These global variables will allow multiple functions to access and manipulate patient data while allowing efficient data flow with little or no complex data passing.

8

3.3 Main Function  
In the program's entry point, use the `main()` function, which invokes the `mainMenu()` function for starting to display the system's main interface. The options on the main menu include: add record, display all records, find patient, edit patient information, and discharge patient. Using the loop structure, users can go inside every option and return back to the menu until they choose the option exit. The above design offers simplicity to the user by locating all functional entries of the system within one menu.  
  
3.4Question Handling  
  
User Input Gathering and prompting are among the central roles of the system. Individual functions are responsible for the module's user input:  
  
- \*\*`addRecord()`\*\*: Collects patient's ID, name, age, disease, and type of visit. The user is prompted for each field, while necessary fields-like age-and ID uniqueness-are being validated.  
Search for a record or edit any record by asking the user to input the ID of the patient. In both cases, the functions `searchRecord()` and `editRecord()` provide feedback if a record is found and if it has been successfully updated; otherwise, it will notify if no records are found.  
  
Discharge a patient with confirmation, with calculation of the final bill based on the visit type, by using their ID number.  
  
The following seemingly targeted functions enhance the data integrity aspect through better feeding of inputs in order to guide the user through each process:.

3.5 Error Handling  
  
Error handling is implemented in HMS for providing incorrect input to make the system function reliably. It checks for valid and unique input:  
  
• Input Validation: Checks age is between 0-120 and unique ID to avoid duplicity.  
• Choice out of range: mainMenu() checks for invalid choices, and it asks for re-input of choice from user if choice is out of range.  
Error Messages: Each function has appropriate feedback in case of errors-for example, an ID being searched and not found, or input values out of acceptable range.  
Gracious Exit: The system provides the user with a quit option in order to ensure users gracefully terminate the program.  
  
This kind of error handling contributes much to the robustness of the system for reliability and friendliness of the interface in managing the patients.

9

Chapter 4

. Testing

The HMS should be tested on a number of variables to ensure it works reliably. This testing was done in four phases: unit testing, function testing, boundary testing, and user experience testing. Each phase is designed to focus on specific aspects of the system and verified that it meets functional, performance, and usability requirements.

4.1 Unit Testing  
  
Unit testing of the individual functions was performed to verify everything works as expected. The main functions individually tested are:  
  
- \*\*`addRecord()`\*\*: It has been checked and implemented so that it adds new patient data, with validation for a unique ID and also with prompts in case certain fields are left unfilled, such as age, visit type, or disease.  
  
- \*\*`searchRecord()`\*\*: It was tested for proper implementation to locate a patient record based on the ID. Test cases were conducted when the ID exists for exact matches and error messages for when the ID does not exist.  
• `dischargePatient()`: The bill was estimated with respect to a visit type being a day-to-day or admitted patient and tested to see whether the record has been deleted post-discharge.  
• `editRecord()`: In case of editing of record is done perfectly, it tests for unique IDs.  
Unit testing for all functions was performed using valid and invalid inputs in order to make the code strong for a variety of user input and error conditions.

4.2 Function Testing  
  
Function testing focuses on how each module performs its own distinctive role within the system. Also included is the testing of the integration of functions in sequence to ensure a smooth workflow of functions. Key tests included:  
  
- \*\*Adding and displaying records\*\*: Ensuring added records appear correctly in the display list and that all information was displayed in a structured format.  
- \*\*Editing a record and search\*\*: The system was tested for retrieval of a record, editing, and saving the changes without loss of data and creation of duplicate records.  
- \*\*Billing accuracy\*\*: Testing whether the billing is correct according to the admission days/daily visit fees depending on the category of visit by the patient.  
 10  
With this testing, modules were interacting with each other correctly; continuity in updating and presentation of data was confirmed.  
  
4.3 Boundary Testing  
  
The boundary testing was, in fact, carried out to determine how the system handled input values at either edge of acceptable ranges for catching failures that might be unexpected. Some critical boundary tests included:  
  
Patient age: minimum and maximum allowable age inputs, along with a few just beyond that range, to ensure proper error handling; Admission days: boundary value testing was considered for values like 0 for daily visits and typical maximum values to ensure accurate billing.  
- \*\*Patient count\*\*: It allows a limit of `MAX\_PATIENTS` in the system. Exactly 100 patients were tested to see if it handles the upper boundary without any problems, which it did, and returned a corresponding error message when over the maximum limit.  
  
These tests show that the system responds correctly to edge cases and is stable over the breadth of expected input ranges.

4.4 User Experience Testing  
  
User experience testing considered how intuitive and user-friendly the system is. Responses from users were reviewed to determine whether:  
  
\* \*\*Ease of Navigation\*\*: How intuitively the main menu allows for navigation, with clear options for each function between functions.  
\* \*\*Prompt Clarity\*\*: If prompts and error messages guide users accordingly in understanding what was needed for each field.  
• Error Feedback: How informative error messages are; whether users can rapidly correct invalid entries.  
• Overall Responsiveness: How well the system performs in terms of response time when a user gives it a command, especially in search, edit, and add-record operations.  
  
Feedback indicated that HMS provides a smooth, easily accessible platform where clear prompts and error messages reduce input errors and enhance user experience.  
  
---  
  
This approach to testing meant HMS was sure to meet quality standards, both in functionality and user experience, as well as in reliability. Let me know if you'd like fur ther detail on any section!

11

Chapter 5

------------------------------------------------------------------------------------------------------------------------------ . Conclusion  
  
HMS successfully meets its objectives relating to making the handling of patients' data easier, helping reduce the administrative burden, and increasing accessibility within healthcare facilities. The modular architecture and its functions, designed to maintain a record of patient admissions and discharges along with billing, have brought ease to working relevant to the same. Minimizing human errors and amplifying efficiency due to automation, HMS has been able to help increase healthcare service management much better.  
  
 5.1 Disadvantages and Shortcomings  
  
Although the HMS implements its basic functionalities, it has some drawbacks that make it inappropriate for a large and more complex environment. Some of the disadvantages are:  
  
- \*\*Limited Storage of Data:\*\* The HMS stores information about the patient in an in-memory array, whose maximum limit is 100 patients. Thus, this severely limits its scalability, making it highly unsuitable for big hospitals or heavy throughput facilities.  
No Persistent Storage: With no kind of persistent data storage, say, integration with a database, all records in the system get lost upon shutdown. This part of the design greatly limits the HMS to just single-session use, requiring it to reset on every use-a highly infeasible solution in real-world applications.  
  
- \*\*No Multi-User Support\*\*: The system currently is designed as a single-user application, which seriously impairs access and can further decrease efficiency in those facilities where several staff members may need to access the system at the same time.  
  
- \*\*Minimal Security\*\*: HMS doesn't implement any security features for keeping patient information secret. The possible consequence could be some issues related to privacy if it were applied in a real health environment.  
  
- \*\*Basic User Interface\*\*: Since console-based, the user interface is not graphical in nature. This makes it less user-friendly and less pleasing to look at compared to the more modern GUI-based systems. Furthermore, console interfaces limit accessibility to users with limited exposure to or experience in command line navigation.  
  
5.2 Future Improvements  
  
In view of these limitations, some of the improvement[s] that could be made to future versions of the HMS are as follows:  
 12  
- \*\*Database Integration\*\*: Adding a database will help the system persistently save its data, so that even when it goes shutdown-a very critical moment for continuous health care.records are maintained at all times.  
  
- \*\*Scalability\*\*: Dynamic data structures or a scalable database back-end shall be permitted so that the system is customized to suit every facility, whether small, medium, or large.  
- \*\*Multi-User Functionality\*\*: A multi-user architecture with role-based access control would enable multiple users to use it simultaneously, therefore increasing efficiency and ease of use for larger institutions.  
  
- \*\*Security Enhancements\*\*: The incorporation of basic security features like login authentication, data encryption, and access restrictions would provide a great deal of security for sensitive patient data and enable compliance with data privacy obligations.  
  
5.3 Conclusion  
  
But these limitations notwithstanding, HMS forms a good starting point in managing patient information for small-scale or educational purposes. It shows the efficacy of structured programming in real-time C applications and points out further possibilities of improvement by addition of more features and use of other technologies. If HMS is designed to overcome most of its present limitations, it would emerge as a potent tool that can be applied in different healthcare settings, enabling better efficiency in operations and providing improved care to patients.  
  
---

.

13