H.M.S. Documentation

bsse f24 | 4361 / 4347 / 4371 / 4370

Complete documentation on the hospital management system (c++).

Iqbal hassan tariq /m. abdullah malik /m. hasnain baig / m. bilal yasir

2024

1. This project has been built using Visual Studio 2022 Enterprise Edition on Windows 11.
2. The documentation and presentations are being made through Microsoft Word & PowerPoint 365 respectively.
3. The documentation is in the following order:

* Explanation of the Program File and its contents.
* Explanation of the Billing class.
* Explanation of the AdmissionRecord1 class, ( it was renamed from AdmissionRecords due to past conflicts.)
* Explanation of the Appointment class.
* Explanation of the Doctor class.
* Explanation of the Patient class.

1. The program initially intended to use enumerations for the patient and doctor types, the files were made but never actually used, they are still in the code.

**System Requirements**

* C++ compiler (e.g., GCC, Clang, Visual Studio).
* Basic knowledge of command-line interface (CLI).

**Main Program Documentation:**

**Overview:** The main program is the entry point for the **Hospital Management System**, where users can interact with different aspects of the system, such as billing, admission records, appointments, doctors, and patient details. The program displays a main menu and provides options to perform various tasks for each category. It uses multiple classes (Billing, AdmissionRecord1, Appointment, Doctor, Patient) to manage the respective functionalities.

**Functionality:** The system is designed with a modular approach, where each module handles a specific task. The options presented to the user depend on the category selected. Here is a breakdown of the various modules:

**Main Menu:**

* **Options:** The main menu provides the following options for the user to select:
  1. Billing Options
  2. Admission Record Options
  3. Appointment Options
  4. Doctor Options
  5. Patient Options
  6. Exit the system
* **Looping:** The program continuously displays the main menu until the user chooses to exit (option 0).

**Modules:**

1. **Billing Options:**
   * **Features:**
     + **Generate Bill:** Allows generating new bills.
     + **Display All Bills:** Displays all bills in the system.
     + **Save Bills:** Saves the bill data to a file.
     + **Load Bills:** Loads the bill data from a file.
   * **Flow:** Users are prompted to select from these options, and each action is handled by the Billing class.
2. **Admission Record Options:**
   * **Features:**
     + **Add Admission Record:** Allows adding new patient admission records.
     + **Discharge Patient:** Handles patient discharge.
     + **Display All Admission Records:** Displays all records in the system.
     + **Save Records to File:** Saves admission records to a file.
     + **Load Records from File:** Loads admission records from a file.
   * **Flow:** These options are managed by the AdmissionRecord1 class.
3. **Appointment Options:**
   * **Features:**
     + **Schedule a New Appointment:** Allows scheduling new patient appointments.
     + **Display All Appointments:** Displays all scheduled appointments.
     + **Search for an Appointment by ID:** Allows searching for appointments by their ID.
     + **Cancel an Appointment:** Cancels a specific appointment.
   * **Flow:** These options are managed by the Appointment class.
4. **Doctor Options:**
   * **Features:**
     + **Add Doctor:** Adds new doctor details to the system.
     + **Assign Doctor to Patient:** Assigns a doctor to a specific patient.
     + **Display Doctor Consultation Statistics:** Displays statistics on consultations with doctors.
     + **Save Doctors to File:** Saves doctor data to a file.
     + **Load Doctors from File:** Loads doctor data from a file.
   * **Flow:** These actions are managed by the Doctor class.
5. **Patient Options:**
   * **Features:**
     + **Add Patient Details:** Adds new patient details (ID, name, age, diagnosis).
     + **Search for Patient Details:** Searches for patient details by ID.
     + **Display All Patients:** Displays details of all patients in the system.
     + **Save Patient Details:** Saves patient data to a file.
     + **Load Patient Details:** Loads patient data from a file.
   * **Flow:** These actions are managed by the Patient class.

**Function Prototypes:** The program defines the following function prototypes:

* **billingOptions()**: Handles billing-related actions.
* **admissionRecordOptions()**: Handles admission record actions.
* **appointmentOptions()**: Handles appointment-related actions.
* **doctorOptions()**: Handles doctor-related actions.
* **patientOptions()**: Handles patient-related actions.

Each of these functions manages the corresponding module, interacting with their respective classes to perform tasks.

**Main Program Flow:**

1. **User Interaction:**
   * The program greets the user with a welcome message.
   * It continuously displays the main menu and prompts the user for their choice.
2. **Switch-Case Structure:**
   * Based on the user's choice, the program enters the corresponding case within the switch statement. For example, selecting 1 leads to the billingOptions() function.
   * Each module handles its respective tasks in a loop until the user decides to return to the main menu (option 0).
3. **Exit Condition:**
   * The program exits when the user selects option 0 from the main menu. A goodbye message is displayed before exiting the loop.

**Example of Program Flow:**

* Upon running the program, the user is prompted to choose an option from the main menu.
* If the user selects 1 (Billing Options), they are taken to the billing menu, where they can choose between generating a bill, displaying bills, saving bills, or loading bills.
* After completing an action, the user can return to the main menu and select another option.

**Error Handling:**

* The program includes basic error handling, such as checking for invalid menu choices (e.g., entering an option outside the valid range).
* If an invalid choice is entered, the program prompts the user to try again.

**Billing Class Documentation**

**Overview:** The Billing class is designed to manage billing operations in a hospital management system. It allows for generating new bills, displaying all generated bills, and saving/loading bills to/from a file. The class keeps track of the billing details, including consultation fees and additional charges, and calculates the total amount for each bill.

**Class Members:**

**Private Members:**

* int billingID[100]: An array to store billing IDs. The class supports a maximum of 100 bills.
* int patientID[100]: An array to store the IDs of the patients associated with each bill.
* double consultationFee[100]: An array to store the consultation fees for each bill.
* double additionalCharges[100]: An array to store any additional charges for each bill.
* int billCount: A variable to track the total number of generated bills.

**Public Methods:**

* **Billing()**
  + **Purpose:** Constructor that initializes billCount to 0 and loads existing bills from a file, if available.
  + **Details:** This constructor ensures that the class starts with no bills, and it loads any previously saved bills from the file at the start.
* **void generateBill()**
  + **Purpose:** Allows the user to generate a new bill by entering billing ID, patient ID, consultation fee, and additional charges.
  + **Details:** The method checks if the total number of bills exceeds the maximum limit of 100. If there is space, it generates a new bill and saves the updated list to a file.
* **void displayBills()**
  + **Purpose:** Displays all generated bills.
  + **Details:** For each bill, the method shows the billing ID, patient ID, consultation fee, additional charges, and the total amount (sum of consultation fee and additional charges). If no bills are available, an error message is displayed.
* **void saveBills()**
  + **Purpose:** Saves all bills to a file (bills.txt).
  + **Details:** Each bill's details (billing ID, patient ID, consultation fee, and additional charges) are written to the file. If the file cannot be opened, an error message is displayed.
* **void loadBills()**
  + **Purpose:** Loads existing bills from the file (bills.txt).
  + **Details:** The method reads the file and loads the bill data into the respective arrays. If the file does not exist, the system informs the user and starts with an empty set of bills.

**File Handling:** The Billing class uses file handling to persist bill records across program executions. The file operations are as follows:

* **Saving Bills:** The saveBills() method opens bills.txt in write mode and stores each bill’s details.
* **Loading Bills:** The loadBills() method opens bills.txt in read mode, parsing the data into the respective arrays. If the file does not exist, the system informs the user and starts with no records.

**Example Usage:**

cpp

Billing billingManager; // Create an instance of the Billing class

billingManager.generateBill(); // Generate a new bill

billingManager.displayBills(); // Display all bills

billingManager.saveBills(); // Save all bills to a file

billingManager.loadBills(); // Load bills from a file

**Error Handling:**

* The generateBill() method ensures that the maximum number of bills (100) is not exceeded.
* The displayBills() method checks if any bills exist and provides feedback if none are available.
* The saveBills() method ensures the file can be opened for writing, and it stops if the file cannot be accessed.
* The loadBills() method ensures the file exists before attempting to load the data, and it initializes the system with no bills if the file is missing.

**Limitations:**

* **Storage Limit:** The class can store up to 100 bills due to the fixed array size.
* **File Handling:** The file bills.txt is expected to be in the same directory as the program. If the file is missing, the system will start fresh with no bills.

**AdmissionRecord1 Class Documentation**

**Overview:** The AdmissionRecord1 class is designed to manage patient admission records in a hospital management system. It handles operations such as admitting patients, discharging them, displaying admission records, and saving/loading records to/from a file. The class tracks each patient's admission and discharge dates and ensures that records are persisted even after program restarts.

**Class Members:**

**Private Members:**

* int recordId[100]: An array to store record IDs for each patient admission.
* int patientId[100]: An array to store the IDs of patients corresponding to each admission record.
* string admissionDate[100]: An array to store the admission dates of patients.
* string dischargeDate[100]: An array to store the discharge dates of patients. An empty string indicates that a patient is not yet discharged.
* int recordCount: A variable to track the number of admission records.

**Public Methods:**

* **AdmissionRecord1()**
  + **Purpose:** Constructor that initializes recordCount to 0, clears any existing discharge dates, and loads any previously saved admission records from a file.
  + **Details:** The constructor ensures that the system starts fresh and loads data from admission\_records.txt if available.
* **void admitPatient()**
  + **Purpose:** Admits a new patient by generating a new record with the patient's record ID, patient ID, and admission date.
  + **Details:** The method ensures that the maximum number of records (100) is not exceeded. It stores the admission data and saves it to a file.
* **void dischargePatient()**
  + **Purpose:** Discharges a patient by updating their record with a discharge date.
  + **Details:** The method checks whether the patient has already been discharged. If not, it allows the user to enter the discharge date and updates the file.
* **void displayRecords()**
  + **Purpose:** Displays all admission records, including record ID, patient ID, admission date, and discharge date (or a message indicating that the patient is not discharged).
  + **Details:** If no records are available, an appropriate message is shown.
* **void saveRecords()**
  + **Purpose:** Saves all admission records to a file (admission\_records.txt).
  + **Details:** The method writes all records, including patient IDs, admission dates, and discharge dates, to the file. If the file cannot be opened, an error message is displayed.
* **void loadRecords()**
  + **Purpose:** Loads existing admission records from the file (admission\_records.txt).
  + **Details:** The method reads records from the file and loads the data into the respective arrays. If the file does not exist, it informs the user and starts with no records.

**File Handling:** The AdmissionRecord1 class uses file handling for persistence. It reads and writes to the admission\_records.txt file:

* **Saving Records:** The saveRecords() method writes each admission record (including patient ID, admission date, and discharge date) to the file.
* **Loading Records:** The loadRecords() method reads from the file and loads the records into the arrays, initializing any missing discharge dates as empty strings.

**Example Usage:**

cpp

AdmissionRecord1 admissionManager; // Create an instance of the AdmissionRecord1 class

admissionManager.admitPatient(); // Admit a new patient

admissionManager.dischargePatient(); // Discharge a patient

admissionManager.displayRecords(); // Display all admission records

admissionManager.saveRecords(); // Save all admission records to a file

admissionManager.loadRecords(); // Load admission records from a file

**Error Handling:**

* The admitPatient() method checks if the record count exceeds 100 before allowing new records to be added.
* The dischargePatient() method ensures that a patient cannot be discharged more than once and handles the case where the patient is already discharged.
* The displayRecords() method checks if any records are available and shows a message if there are none.
* The saveRecords() method ensures the file can be opened for writing, and stops if it cannot be accessed.
* The loadRecords() method checks if the file exists and starts fresh if the file is not found.

**Limitations:**

* **Storage Limit:** The system can store a maximum of 100 admission records due to the fixed array size.
* **File Handling:** The file admission\_records.txt must be in the same directory as the program. If the file is missing, the system will load with no records.

**Appointment Class Documentation**

**Overview:** The Appointment class is designed to manage and schedule patient appointments. It allows for scheduling new appointments, displaying existing appointments, searching for specific appointments, and canceling them. The class also includes functionality to save and load appointment records to/from a file for persistence.

**Class Members:**

**Private Members:**

* int appointmentID[100]: An array to store appointment IDs. The class supports a maximum of 100 appointments.
* int patientID[100]: An array to store the IDs of the patients associated with each appointment.
* int doctorID[100]: An array to store the IDs of the doctors associated with each appointment.
* string appointmentDate[100]: An array to store the dates of the appointments.
* string appointmentTime[100]: An array to store the times of the appointments.
* int appointmentCount: A variable to keep track of the total number of appointments.

**Public Methods:**

* **Appointment()**
  + **Purpose:** Constructor that initializes appointmentCount to 0 and loads existing appointments from a file, if available.
  + **Details:** This constructor ensures that the class starts with no appointments, and it loads any previously saved appointments from the file at the start.
* **void scheduleAppointment()**
  + **Purpose:** Allows the user to schedule a new appointment by entering appointment ID, patient ID, doctor ID, date, and time.
  + **Details:** The method checks for duplicate appointment IDs and conflicting appointment times before scheduling. It increments the appointmentCount and saves the updated list of appointments to a file.
* **void displayAppointments()**
  + **Purpose:** Displays all scheduled appointments.
  + **Details:** For each appointment, the method shows the appointment ID, patient ID, doctor ID, date, and time. If no appointments are scheduled, it displays an error message.
* **void searchAppointment()**
  + **Purpose:** Searches for a specific appointment by its appointment ID.
  + **Details:** The method asks for the appointment ID and searches through the scheduled appointments. If a match is found, it displays the details of the appointment.
* **void cancelAppointment()**
  + **Purpose:** Allows the user to cancel an appointment by its appointment ID.
  + **Details:** The method searches for the appointment by its ID and removes it from the list by shifting subsequent appointments down. It then updates the appointment list in the file.
* **void saveAppointments()**
  + **Purpose:** Saves all appointments to a file (appointments.txt).
  + **Details:** Each appointment's details (appointment ID, patient ID, doctor ID, date, and time) are written to the file. If the file cannot be opened, an error message is displayed.
* **void loadAppointments()**
  + **Purpose:** Loads existing appointments from the file (appointments.txt).
  + **Details:** The method reads the file and loads the appointment data into the respective arrays. If the file does not exist, the system informs the user and starts with an empty set of appointments.

**File Handling:** The Appointment class uses file handling to persist appointment records across program executions. The file operations are as follows:

* **Saving Appointments:** The saveAppointments() method opens appointments.txt in write mode and stores each appointment’s details.
* **Loading Appointments:** The loadAppointments() method opens appointments.txt in read mode, parsing the data into the respective arrays. If the file does not exist, the system informs the user and starts with no records.

**Example Usage:**

cpp

Copy code

Appointment appointmentManager; // Create an instance of the Appointment class

appointmentManager.scheduleAppointment(); // Schedule a new appointment

appointmentManager.displayAppointments(); // Display all appointments

appointmentManager.searchAppointment(); // Search for a specific appointment

appointmentManager.cancelAppointment(); // Cancel an existing appointment

appointmentManager.saveAppointments(); // Save appointments to a file

appointmentManager.loadAppointments(); // Load appointments from a file

**Error Handling:**

* The scheduleAppointment() method ensures that the maximum number of appointments (100) is not exceeded and checks for duplicate appointment IDs or conflicting appointment times.
* The searchAppointment() method provides feedback if no appointment is found with the specified ID.
* The cancelAppointment() method ensures that only valid appointments are canceled, and it properly shifts the remaining appointments in the array.
* The saveAppointments() method checks if the file can be opened for writing and displays an error if it cannot.
* The loadAppointments() method verifies the existence of the file before attempting to load appointments.

**Limitations:**

* **Storage Limit:** The class can store up to 100 appointments due to the fixed array size.
* **File Handling:** The file appointments.txt is expected to be in the same directory as the program. If the file is missing, the system will start fresh with no appointments.

**Doctor Class Documentation**

**Overview:** The Doctor class manages the information of doctors in a hospital management system. It handles operations such as adding new doctors, assigning doctors to patients, displaying consultation statistics, and saving/loading doctor data to/from a file. This class tracks doctor information, including IDs, names, specializations, and consultation counts.

**Class Members:**

**Private Members:**

* int doctorID[100]: An array to store the doctor IDs for each doctor.
* string doctorName[100]: An array to store the names of the doctors.
* string specialization[100]: An array to store the specializations of the doctors.
* int consultations[100]: An array to store the number of consultations each doctor has had.
* int doctorCount: A variable to track the number of doctors currently in the system.

**Public Methods:**

* **Doctor()**
  + **Purpose:** Constructor that initializes doctorCount to 0, sets consultation counts to 0, and loads existing doctor data from a file (doctors.txt).
  + **Details:** Ensures that the class starts fresh with no doctors or loads existing doctor data from the file.
* **void addDoctor()**
  + **Purpose:** Adds a new doctor to the system by collecting doctor ID, name, and specialization from the user.
  + **Details:** Ensures the doctor ID is unique and that name and specialization are not empty. The new doctor is saved to the file (doctors.txt), and the doctor count is updated.
* **void assignDoctorToPatient()**
  + **Purpose:** Assigns a doctor to a patient and increments the consultation count for the assigned doctor.
  + **Details:** It asks for the doctor ID, finds the corresponding doctor, and increments their consultation count. If the doctor is not found, an error message is displayed.
* **void displayConsultationStatistics()**
  + **Purpose:** Displays a list of all doctors, their specialization, and the number of consultations they have performed.
  + **Details:** Lists all the doctors in the system with their consultation statistics. If no doctors exist, an error message is shown.
* **void saveDoctors()**
  + **Purpose:** Saves all doctor information to the file (doctors.txt).
  + **Details:** Writes doctor data, including ID, name, specialization, and consultation count, to the file. If the file cannot be opened, an error message is displayed.
* **void loadDoctors()**
  + **Purpose:** Loads doctor data from the file (doctors.txt).
  + **Details:** Reads doctor data from the file and loads it into the class arrays. If the file does not exist, a fresh start is initiated with no doctor data.

**File Handling:** The Doctor class uses file handling for persistence:

* **Saving Doctors:** The saveDoctors() method writes all doctor information (ID, name, specialization, consultation count) to the file doctors.txt.
* **Loading Doctors:** The loadDoctors() method reads doctor information from doctors.txt and loads it into the respective arrays. If the file is missing, it notifies the user and starts fresh.

**Example Usage:**

cpp

Doctor doctorManager; // Create an instance of the Doctor class

doctorManager.addDoctor(); // Add a new doctor to the system

doctorManager.assignDoctorToPatient(); // Assign a doctor to a patient

doctorManager.displayConsultationStatistics(); // Display all doctors' consultation statistics

doctorManager.saveDoctors(); // Save the doctors' data to a file

doctorManager.loadDoctors(); // Load doctors' data from a file

**Error Handling:**

* The addDoctor() method checks whether the doctor ID already exists and ensures that the name and specialization are not empty.
* The assignDoctorToPatient() method checks if there are any doctors available and whether the entered doctor ID exists in the system.
* The displayConsultationStatistics() method ensures that there are doctors available before attempting to display their statistics.
* The saveDoctors() and loadDoctors() methods handle file access errors and provide error messages if the file cannot be accessed.

**Limitations:**

* **Storage Limit:** The system can store a maximum of 100 doctors due to the fixed array size.
* **File Handling:** The file doctors.txt must be in the same directory as the program. If the file is missing, the system will load with no doctors.

**Patient Class Documentation**

**Overview:** The Patient class manages patient records within a hospital management system. It provides functionality for adding new patients, searching for a patient by ID, displaying all patient information, and saving/loading patient data to/from a file. The class keeps track of basic patient details, including ID, name, age, and diagnosis.

**Class Members:**

**Private Members:**

* int patientID[100]: An array to store the patient IDs.
* string patientName[100]: An array to store the names of the patients.
* int patientAge[100]: An array to store the ages of the patients.
* string diagnosis[100]: An array to store diagnoses for each patient.
* int patientCount: A variable to track the number of patients in the system.

**Public Methods:**

* **Patient()**
  + **Purpose:** Constructor to initialize patientCount to 0 and load any existing patient data from the file (patients.txt).
  + **Details:** The constructor ensures that the system starts with no patients or loads existing patient records from the file.
* **void addPatient()**
  + **Purpose:** Adds a new patient to the system by accepting patient ID, name, age, and diagnosis from the user.
  + **Details:** This method checks that the patient ID is unique, the name and diagnosis are non-empty, and the age is a positive number. It then saves the new patient record and updates the patient count.
* **void searchPatient()**
  + **Purpose:** Searches for a patient by their unique ID and displays the corresponding details (ID, name, age, diagnosis).
  + **Details:** If a patient with the given ID is found, their information is displayed; otherwise, an error message is shown.
* **void displayPatients()**
  + **Purpose:** Displays the list of all patients currently in the system.
  + **Details:** If there are no patients in the system, an error message is shown. Otherwise, all patients' IDs, names, ages, and diagnoses are displayed.
* **void savePatients()**
  + **Purpose:** Saves all patient records to the file (patients.txt).
  + **Details:** Writes patient data (ID, name, age, diagnosis) to the file. If the file cannot be opened, an error message is shown.
* **void loadPatients()**
  + **Purpose:** Loads patient records from the file (patients.txt).
  + **Details:** Reads patient data from the file and populates the class arrays. If the file does not exist, the system starts with no patients.

**File Handling:** The Patient class uses file handling for persistence:

* **Saving Patients:** The savePatients() method writes patient data (ID, name, age, diagnosis) to the patients.txt file.
* **Loading Patients:** The loadPatients() method reads patient data from the patients.txt file and populates the class arrays. If the file is missing, it starts fresh with no patients.

**Example Usage:**

Cpp

Patient patientManager; // Create an instance of the Patient class

patientManager.addPatient(); // Add a new patient

patientManager.searchPatient(); // Search for a patient by ID

patientManager.displayPatients(); // Display all patients' information

patientManager.savePatients(); // Save patients' data to the file

patientManager.loadPatients(); // Load patients' data from the file

**Error Handling:**

* The addPatient() method checks if the patient ID already exists, ensures the name and diagnosis are not empty, and that the age is a positive number.
* The searchPatient() method checks whether the entered patient ID exists in the system.
* The displayPatients() method ensures that there are patients to display before proceeding.
* The savePatients() and loadPatients() methods handle file access errors, including when the file cannot be opened or doesn't exist.

**Limitations:**

* **Storage Limit:** The system can store a maximum of 100 patients due to the fixed array size.
* **File Handling:** The file patients.txt must be in the same directory as the program. If the file is missing, the system will start with no patient data.