SESI 2023/2024

(A231)

DTS3083(C) STRUKTUR DATA

TAJUK: HOSPITAL PATIENT MANAGEMENT SYSTEM

NAMA	**NOMBOR MATRIK**
MUHAMMAD HAIKAL BIN MOHD JAKI	D20221101806
MUHAMMAD HAIKAL BIN AZMAN	D20221101846
AMIRUL AZIM BIN APANDI	D20221101850
MUHAMAD ALI HANAFIAH BIN SABARUDIN	D20221101859

NAMA PENSYARAH: PUAN HASNATUL NAZUHA BINTI HASSAN

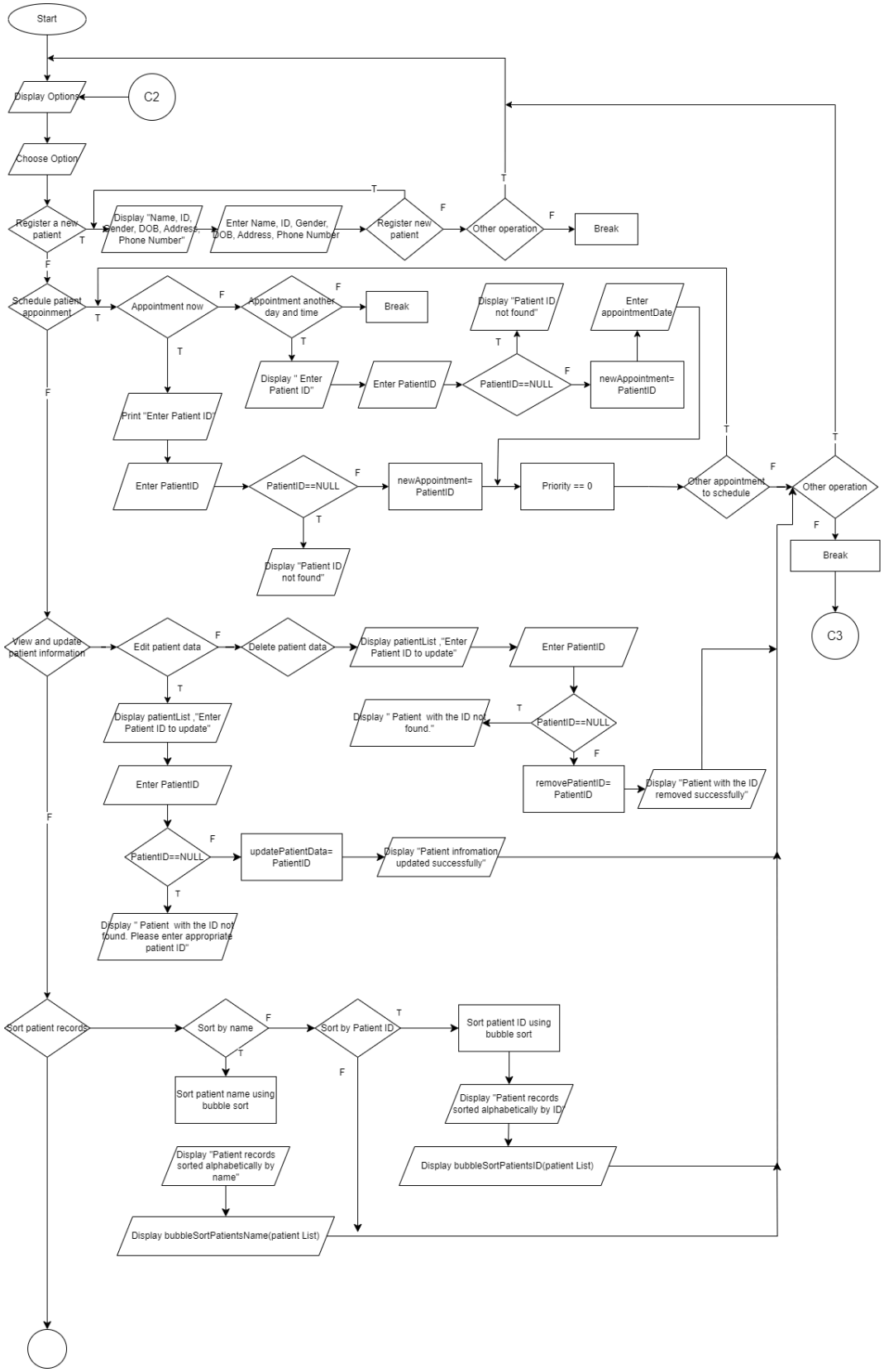
1. PROBLEM ANALYSIS

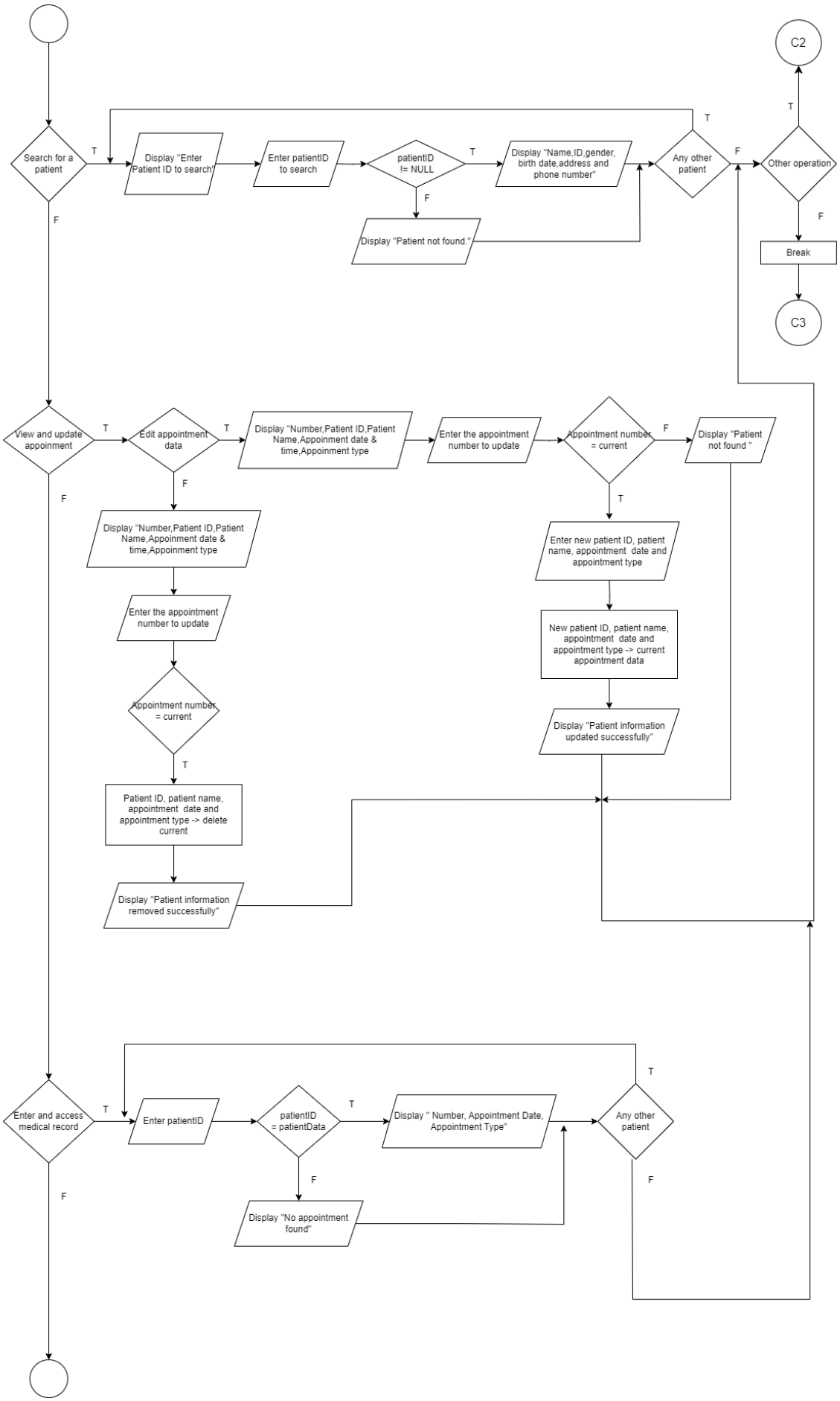
Case	Input	Process	Output
1. Register a new patient	Name, ID number, Gender, Date of Birth, Address and Phone Number	To register a new patient in patient data by the input	Patient information will be registered in patient data
2. Schedule a patient appointment	Number 1 or 2, Patient ID, appointment type, appointment date and time	1. To assign the appointment at exact time by the Patient ID 2. To assign the appointment by the date and time that entered by the user	1. The appointment assigned at the exact time and displayed the appointment ID 2. The appointment assigned by the date and time that entered by the user
3. View and update patient information	Number 1 or 2, Name, Patient ID, Gender, Date of Birth, Address and Phone Number	1. To edit patient data and assign new information for patient by input 2. To delete the patient information from patient data by Patient ID	1. New patient information assigned in patient data 2. Patient information deleted from patient data
4. Sort patient records	Number 1 or 2	1. To sort patient record by name	1. Patient record sorted by name

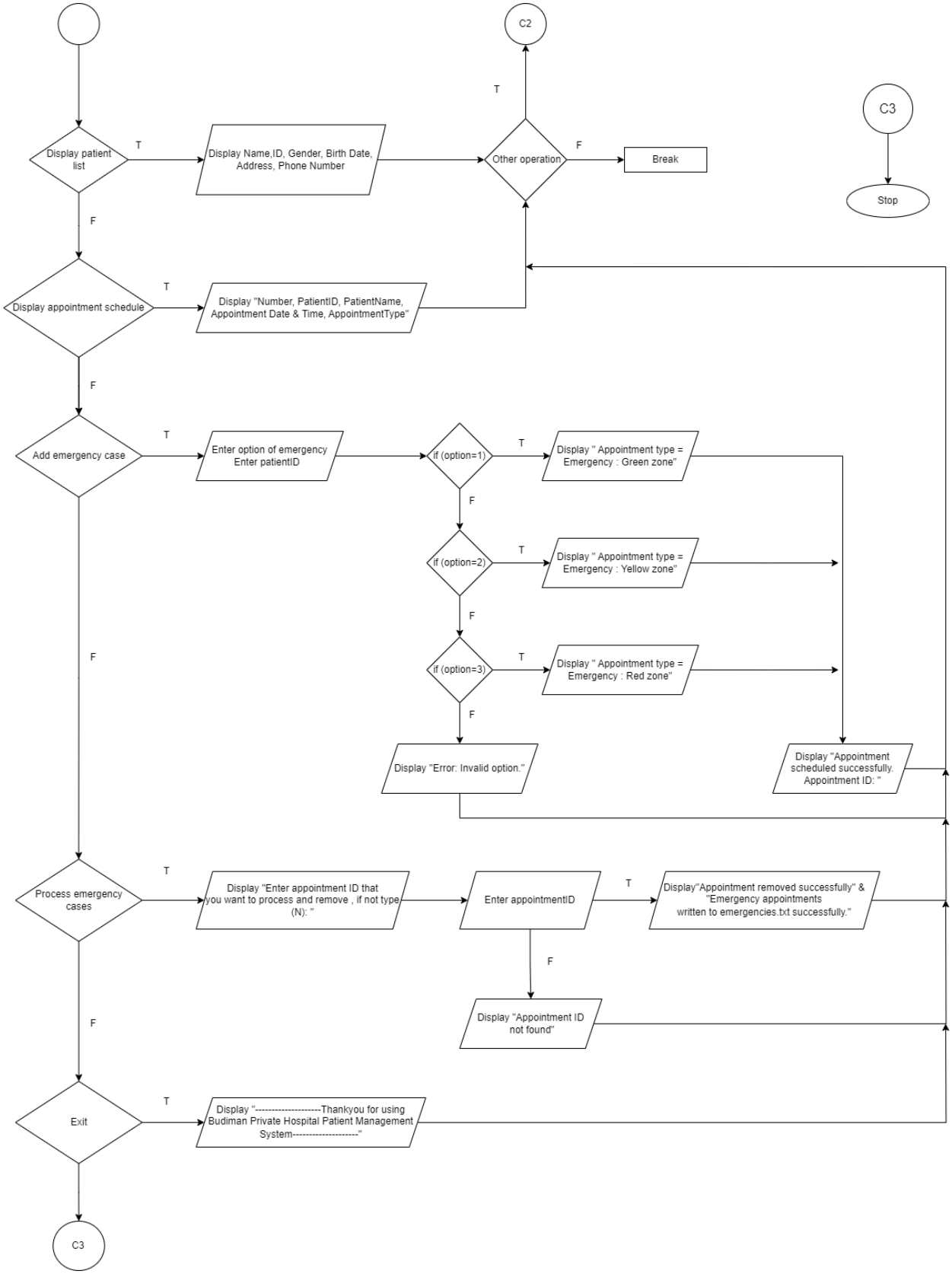
		2. To sort patient record by ID number	2. Patient record sorted by ID number
5. Search for a patient	Patient ID	To search patient by their ID number	Display the patient information
6. View and update appointment	Appointment number, Patient ID, Patient Name, Appointment date, Number 1 or 2	1. To edit appointment data and assign new data at the record by the input 2. To delete appointment data from the record by the Patient ID	1. New appointment data assigned at the record 2. The appointment data deleted from the record
7. Enter and access medical records	Patient ID	To access medical record by the patient ID	Display the patient medical record
8. Display patient list	No input	To display patient list recorded	Display patient list
9. Display appointment schedule	No input	To display patient appointment schedule recorded	Display the patient appointment schedule record
10. Add an emergency case	Number 1, 2 or 3 and patient ID	Add emergency appointment based	Appointment scheduled and

		on emergency zone (Green,Yellow or Red)	display appointment ID
11. Process emergency cases	Appointment ID	To display and delete the emergency record from the data by the appointment ID	Display emergency appointment record and delete the emergency appointment
12. Exit	No input	To terminate all the process	Display thank you for using Budiman Private Hospital Patient Management System

2. FLOW CHART


Start
Display Options
C2
Choose Option
Register a new patient
Display "Name, ID, Gender, DOB, Address, Phone Number"
Enter Name, ID, Gender, DOB, Address, Phone Number
Register new patient
Other operation
Break
Schedule patient appoinment
Appointment now
Appointment another day and time
Break
Display " Enter Patient ID"
Enter PatientID
PatientID==NULL
Display "Patient ID not found"
newAppointment= PatientID
Enter appointmentDate
Print "Enter Patient ID"
Enter PatientID
PatientID==NULL
newAppointment= PatientID
Priority == 0
Other appointment to schedule
Other operation
Break
C3
Display "Patient ID not found"
View and update patient information
Edit patient data
Delete patient data
Display patientList ,"Enter Patient ID to update"
Enter PatientID
PatientID==NULL
Display " Patient with the ID not found."
removePatientID= PatientID
Display "Patient with the ID removed successfully"
Display patientList ,"Enter Patient ID to update"
Enter PatientID
PatientID==NULL
updatePatientData= PatientID
Display "Patient infromation updated successfully"
Display " Patient with the ID not found. Please enter appropriate patient ID"
Sort patient records
Sort by name
Sort by Patient ID
Sort patient ID using bubble sort
Sort patient name using bubble sort
Display "Patient records sorted alphabetically by ID"
Display "Patient records sorted alphabetically by name"
Display bubbleSortPatientsID(patient List)
Display bubbleSortPatientsName(patient List)


C2
Search for a patient
T
Display "Enter Patient ID to search"
Enter patientID to search
patientID != NULL
T
Display "Name,ID,gender, birth date,address and phone number"
Any other patient
T
F
Other operation
T
F
Break
C3
F
Display "Patient not found."
View and update appoinment
T
Edit appointment data
T
Display "Number,Patient ID,Patient Name,Appoinment date & time,Appoinment type
Enter the appointment number to update
Appointment number = current
F
Display "Patient not found "
T
F
Display "Number,Patient ID,Patient Name,Appoinment date & time,Appoinment type
Enter the appointment number to update
Appointment number = current
T
Patient ID, patient name, appointment date and appointment type -> delete current
Display "Patient information removed successfully"
Enter new patient ID, patient name, appointment date and appointment type
New patient ID, patient name, appointment date and appointment type -> current appointment data
Display "Patient information updated successfully"
F
Enter and access medical record
T
Enter patientID
patientID = patientData
T
Display " Number, Appointment Date, Appointment Type"
Any other patient
T
F
F
Display "No appointment found"


C2
T
C3
Display patient list
T
Display Name,ID, Gender, Birth Date, Address, Phone Number
Other operation
F
Break
Stop
F
Display appointment schedule
T
Display "Number, PatientID, PatientName, Appointment Date & Time, AppointmentType"
F
Add emergency case
T
Enter option of emergency
Enter patientID
if (option=1)
T
Display " Appointment type = Emergency : Green zone"
F
if (option=2)
T
Display " Appointment type = Emergency : Yellow zone"
F
if (option=3)
T
Display " Appointment type = Emergency : Red zone"
F
Display "Error: Invalid option."
Display "Appointment scheduled successfully. Appointment ID: "
F
Process emergency cases
T
Display "Enter appointment ID that you want to process and remove , if not type (N): "
Enter appointmentID
T
Display"Appointment removed successfully" & "Emergency appointments written to emergencies.txt successfully."
F
Display "Appointment ID not found"
F
Exit
T
Display "----------Thankyou for using Budiman Private Hospital Patient Management System----------"
C3

3. COMPLETE C++ PROGRAM

```
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
#include <queue>
#include <ctime>
#include <vector>

using namespace std;

// Structure to represent patient data
struct Patient {
    string name, patientID, gender, DOB, address, phoneNo;
};

// Structure to represent appointment data
struct Appointment {
    string appointmentNum, patientID, patientName, appointmentDate,
appointmentType, priority; // Add a priority field for emergency
cases
};

// Structure for a linked list node to store patient data
struct patientNode {
    Patient patientData;
    patientNode* next; // Pointer to the next node in the linked
list
    patientNode* prev; // Add a pointer to the previous node for
doubly-linked list functionality
};

// Structure for a linked list node to store appointment data
struct appointmentNode {
    Appointment appointmentData;
    appointmentNode* next; // Pointer to the next node in the linked
list
};
```

```
int latestAppointmentNumber = 1000; // Initialize the variable to
keep track of the latest appointment number with a starting value of
1000
int appointmentPriority = 0; // Initialize the variable to represent
the priority of appointments with a starting value of 0



// PATIENT FUNCTION
// Function to insert a new patient into the linked list
patientNode* insertPatient(patientNode* head, const Patient&
newPatient) {
    patientNode* newNode = new patientNode{newPatient, nullptr,
nullptr};

    if (head == nullptr) {
        return newNode; // The list is empty, so the new node
becomes the head
    }

    patientNode* current = head;
    while (current->next != nullptr) {
        current = current->next; // Move to the last node
    }

    current->next = newNode; // Append the new node to the end
    return head;
}

// Function to display only patient names after sorting
void displaySortedPatientNames(const patientNode* head) {
    const patientNode* current = head;

    cout << "Sorted Patient Names:" << endl;
    cout << left << setw(20) << "Name" << setw(15) << "ID" <<
setw(8) << "Gender"
         << setw(12) << "Birth Date" << setw(50) << "Address" <<
setw(15) << "Phone Number" << endl;

    while (current != nullptr) {
```

```
        cout << left << setw(20) << current->patientData.name <<
setw(15) << current->patientData.patientID
             << setw(8) << current->patientData.gender << setw(12)
<< current->patientData.DOB
             << setw(50) << current->patientData.address << setw(15)
<< current->patientData.phoneNo << endl;

        current = current->next;
    }
}

// Function to perform bubble sort on patient data based on their
names
void bubbleSortPatientsName(patientNode* head) {
    if (head == nullptr || head->next == nullptr) {
        // No need to sort if the list is empty or has only one
element
        return;
    }

    bool swapped;
    patientNode* current;
    patientNode* lastSorted = nullptr;

    do {
        swapped = false;
        current = head;

        while (current->next != lastSorted) {
            // Compare adjacent names and swap if needed
            if (current->patientData.name >
current->next->patientData.name) {
                swap(current->patientData,
current->next->patientData);
                swapped = true;
            }

            current = current->next;
        }
```

```
        lastSorted = current; // Mark the last sorted node
    } while (swapped);
}

// Function to perform bubble sort on patient data based on their ID
void bubbleSortPatientsID(patientNode* head) {
    if (head == nullptr || head->next == nullptr) {
        // No need to sort if the list is empty or has only one
element
        return;
    }

    bool swapped;
    patientNode* current;
    patientNode* lastSorted = nullptr;

    do {
        swapped = false;
        current = head;

        while (current->next != lastSorted) {
            // Compare adjacent IDs and swap if needed
            if (current->patientData.patientID >
current->next->patientData.patientID) {
                swap(current->patientData,
current->next->patientData);
                swapped = true;
            }

            current = current->next;
        }

        lastSorted = current; // Mark the last sorted node
    } while (swapped);
}

// Function to display patient data in a formatted table
void displayPatientsFile(const patientNode* head) {
    const patientNode* current = head;
```

```
    // Display table header
    cout << left << setw(20) << "Name" << setw(15) << "ID" <<
setw(8) << "Gender"
         << setw(12) << "Birth Date" << setw(50) << "Address" <<
setw(15) << "Phone Number" << endl;

    while (current != nullptr) {
        // Display patient data in table rows
        cout << left << setw(20) << current->patientData.name <<
setw(15) << current->patientData.patientID
             << setw(8) << current->patientData.gender << setw(12)
<< current->patientData.DOB
             << setw(50) << current->patientData.address << setw(15)
<< current->patientData.phoneNo << endl;

        current = current->next;
    }
}

// Function to write and insert patient data into a file
(patients.txt)
void writePatientsFile(const patientNode* head, const string&
fileName) {
    ofstream outFile(fileName, ios::out | ios::app);  // Open file
for writing and append to existing content
    if (!outFile.is_open()) {
        cerr << "Error opening file: " << fileName << endl;
        return;
    }

    const patientNode* current = head;
    while (current != nullptr) {
        // Write patient data to the file and separate it with the
"|" symbol
        outFile << current->patientData.name << "|" <<
current->patientData.patientID << "|"
                << current->patientData.gender << "|" <<
current->patientData.DOB << "|"
                << current->patientData.address << "|" <<
current->patientData.phoneNo << "\n";
```

```
            current = current->next;
        }

        outFile.close();
        if (outFile.fail()) {
            cerr << "Error closing file: " << fileName << endl;
        }
    }

    // Function to read and load patient data from a file (patients.txt)
    patientNode* readPatientsFile(const string& filename) {
        ifstream inFile(filename);
        patientNode* head = nullptr;

        while (inFile) {
            Patient newPatient;

            getline(inFile, newPatient.name, '|');
            getline(inFile, newPatient.patientID, '|');
            getline(inFile, newPatient.gender, '|');
            getline(inFile, newPatient.DOB, '|');
            getline(inFile, newPatient.address, '|');
            getline(inFile, newPatient.phoneNo);

            if (!newPatient.name.empty()) {
                head = insertPatient(head, newPatient);
            }
        }

        inFile.close();
        return head;
    }

    // Function to prompt the user to enter data for a new patient
    void newPatient(patientNode*& patientList) {
        Patient newPatient;
        cin.ignore();  // Ignore newline character from previous input
        cout << "Please enter patient details as below:" << endl;
        cout << " Name: ";
        getline(cin, newPatient.name);
```

```
    cout << " ID (Axxxxx): ";
    getline(cin, newPatient.patientID);

    cout << " Gender (Male/Female): ";
    getline(cin, newPatient.gender);

    cout << " Date of Birth (DD/MM/YYYY): ";
    getline(cin, newPatient.DOB);

    cout << " Address: ";
    getline(cin, newPatient.address);

    cout << " Phone number (0123456789): ";
    getline(cin, newPatient.phoneNo);

    patientList = insertPatient(patientList, newPatient);

    // Append the new patient to the file without rewriting the
entire list
    ofstream outFile("patients.txt", ios::out | ios::app);
    if (!outFile.is_open()) {
        cerr << "Error opening file: " << "patients.txt" << endl;
        return;
    }

    outFile << newPatient.name << "|" << newPatient.patientID << "|"
            << newPatient.gender << "|" << newPatient.DOB << "|"
            << newPatient.address << "|" << newPatient.phoneNo <<
"\n";

    outFile.close();
    if (outFile.fail()) {
        cerr << "Error closing file: " << "patients.txt" << endl;
    }
}

// Function that lets the user edit patient information based on the
patient ID
void updatePatientData(patientNode* head) {
```

```
    string patientID;
    cout << "Enter the Patient ID to update: ";
    cin >> patientID;

    patientNode* current = head;

    while (current != nullptr) {
        if (current->patientData.patientID == patientID) {
            // Patient found, allow the user to update information
            cout << "Enter new details for the patient:" << endl;
            cout << " Name: ";
            cin.ignore(); // Ignore newline character from previous
input
            getline(cin, current->patientData.name);

            cout << " Gender (M/F): ";
            getline(cin, current->patientData.gender);

            cout << " Date of Birth: ";
            getline(cin, current->patientData.DOB);

            cout << " Address: ";
            getline(cin, current->patientData.address);

            cout << " Phone number: ";
            getline(cin, current->patientData.phoneNo);

            // Open the new file for writing
            ofstream outFileTemp("patients_temp.txt");
            if (!outFileTemp.is_open()) {
                cerr << "Error opening file: " <<
"patients_temp.txt" << endl;
                return;
            }

            // Write all patients to the new file
            patientNode* tempCurrent = head;
            while (tempCurrent != nullptr) {
                outFileTemp << tempCurrent->patientData.name << "|"
<< tempCurrent->patientData.patientID << "|"
```

```
                                << tempCurrent->patientData.gender <<
"|" << tempCurrent->patientData.DOB << "|"
                                << tempCurrent->patientData.address <<
"|" << tempCurrent->patientData.phoneNo << "\n";
                tempCurrent = tempCurrent->next;
            }

            outFileTemp.close();
            if (outFileTemp.fail()) {
                cerr << "Error closing file: " <<
"patients_temp.txt" << endl;
            }

            // Remove the old file
            remove("patients.txt");

            // Rename the new file to the original file name
            rename("patients_temp.txt", "patients.txt");

            cout << "Patient information updated successfully." <<
endl;
            return;
        }

        current = current->next;
    }

    // Patient not found
    cout << "Patient with ID " << patientID << " not found. Please
enter the appropriate patient ID." << endl;
}

// Function to search for a patient by using patient ID by
implementing linear search
void searchPatientData(patientNode* head) {
    string patientID;
    cout << "Enter the Patient ID to search: ";
    cin >> patientID;

    patientNode* current = head;
```

```
    // Search for the patient with the entered ID
    while (current != nullptr) {
        if (current->patientData.patientID == patientID) {
            // Display patient data in table rows
            cout << left << setw(20) << "Name" << setw(15) << "ID"
<< setw(8) << "Gender"
                 << setw(12) << "Birth Date" << setw(50) <<
"Address" << setw(15) << "Phone Number" << endl;

            cout << left << setw(20) << current->patientData.name <<
setw(15) << current->patientData.patientID
                 << setw(8) << current->patientData.gender <<
setw(12) << current->patientData.DOB
                 << setw(50) << current->patientData.address <<
setw(15) << current->patientData.phoneNo << endl;

            return; // Exit the function after finding and
displaying the patient
        }

        current = current->next;
    }

    // Patient with the entered ID not found
    cout << "Patient with ID " << patientID << " not found." <<
endl;
}

// Function to remove a patient from the linked list based on
patient ID
void removePatient(patientNode*& head, const string& patientID) {
    if (head == nullptr) {
        cout << "No patients to remove. The list is empty." << endl;
        return;
    }

    patientNode* current = head;
    patientNode* previous = nullptr;
```

```
    // Traverse the list to find the patient with the specified ID
    while (current != nullptr && current->patientData.patientID !=
patientID) {
        previous = current;
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Patient with ID " << patientID << " not found." <<
endl;
        return;
    }

    // Remove the patient node from the list
    if (previous == nullptr) {
        // The patient to be removed is the head of the list
        head = head->next;
    } else {
        previous->next = current->next;
    }

    // Remove the patient data from the file
    ofstream outFileTemp("patients_temp.txt");
    if (!outFileTemp.is_open()) {
        cerr << "Error opening file: " << "patients_temp.txt" <<
endl;
        return;
    }

    patientNode* tempCurrent = head;
    while (tempCurrent != nullptr) {
        outFileTemp << tempCurrent->patientData.name << "|" <<
tempCurrent->patientData.patientID << "|"
                    << tempCurrent->patientData.gender << "|" <<
tempCurrent->patientData.DOB << "|"
                    << tempCurrent->patientData.address << "|" <<
tempCurrent->patientData.phoneNo << "\n";
        tempCurrent = tempCurrent->next;
    }
```

```
    outFileTemp.close();
    if (outFileTemp.fail()) {
        cerr << "Error closing file: " << "patients_temp.txt" <<
endl;
    }

    // Remove the old file
    remove("patients.txt");

    // Rename the new file to the original file name
    rename("patients_temp.txt", "patients.txt");

    delete current;
    cout << "Patient with ID " << patientID << " removed
successfully." << endl;
}


// APPOINTMENT FUNCTION
// Function to insert a new appointment into the linked list
appointmentNode* insertAppointment(appointmentNode* head, const
Appointment& newAppointment) {
    appointmentNode* newNode = new appointmentNode{newAppointment,
nullptr};

    if (head == nullptr || newAppointment.appointmentDate <
head->appointmentData.appointmentDate) {
        // If the list is empty or the new appointment comes before
the head, insert at the beginning
        newNode->next = head;
        return newNode;
    }

    appointmentNode* current = head;

    while (current->next != nullptr &&
           newAppointment.appointmentDate >
current->next->appointmentData.appointmentDate) {
        current = current->next;
    }
```

```
    // Insert the new appointment after the current node
    newNode->next = current->next;
    current->next = newNode;

    return head;
}

// Function to display appointment data in a formatted table
void displayAppointmentSchedule(const appointmentNode* head) {
    const appointmentNode* current = head;

    cout << left << setw(10) << "Number" << setw(15) << "PatientID"
<< setw(20) << "PatientName"
         << setw(25) << "Appointment Date & Time" << setw(15) <<
"AppointmentType" << endl;

    while (current != nullptr) {
        cout << left << setw(10) <<
current->appointmentData.appointmentNum << setw(15) <<
current->appointmentData.patientID
             << setw(20) << current->appointmentData.patientName <<
setw(25) << current->appointmentData.appointmentDate
             << setw(15) << current->appointmentData.appointmentType
<< endl;

        current = current->next;
    }
}

// Function to find the latest appointment number inside the text
file
int findMaxAppointmentNumber(const appointmentNode* head) {
    int maxAppointmentNumber = 1000; // Default starting value

    const appointmentNode* current = head;
    while (current != nullptr) {
        int currentNumber =
stoi(current->appointmentData.appointmentNum);
```

```
        maxAppointmentNumber = max(maxAppointmentNumber,
currentNumber);
        current = current->next;
    }

    return maxAppointmentNumber;
}

// Function to sort appointments based on date and time
appointmentNode* sortAppointments(appointmentNode* head) {
    if (head == nullptr || head->next == nullptr) {
        // No need to sort if the list is empty or has only one
element
        return head;
    }

    appointmentNode* sortedList = nullptr;
    appointmentNode* current = head;

    while (current != nullptr) {
        appointmentNode* next = current->next;

        if (sortedList == nullptr ||
            current->appointmentData.appointmentDate <
sortedList->appointmentData.appointmentDate) {
            // If the sorted list is empty or the current
appointment should come before the sorted list's head,
            // insert the current appointment at the beginning
            current->next = sortedList;
            sortedList = current;
        } else {
            // Otherwise, find the correct position in the sorted
list
            appointmentNode* temp = sortedList;
            while (temp->next != nullptr &&
                   current->appointmentData.appointmentDate >
temp->next->appointmentData.appointmentDate) {
                temp = temp->next;
            }
```

```
            // Insert the current appointment after temp
            current->next = temp->next;
            temp->next = current;
        }

        current = next;
    }

    return sortedList;
}

// Function to insert a new appointment into the linked list
void writeAppointmentData(const appointmentNode* head, const string&
fileName) {
    ofstream outFile(fileName, ios::out | ios::app);  // Open file
for writing and appending
    if (!outFile.is_open()) {
        cerr << "Error opening file: " << fileName << endl;
        return;
    }

    const appointmentNode* current = head;
    while (current != nullptr) {
        outFile << current->appointmentData.appointmentNum << "|"
                << current->appointmentData.patientID << "|"
                << current->appointmentData.patientName << "|"
                << current->appointmentData.appointmentDate << "|"
                << current->appointmentData.appointmentType << "|"
                << current->appointmentData.priority << "\n";

        current = current->next;
    }

    outFile.close();
    if (outFile.fail()) {
        cerr << "Error closing file: " << fileName << endl;
    }
}

// Function to read appointment data from a file (appointments.txt)
```

```
appointmentNode* readAppointmentsFile(const string& filename) {
    ifstream inFile(filename);
    appointmentNode* head = nullptr;

    while (inFile) {
        Appointment newAppointment;

        getline(inFile, newAppointment.appointmentNum, '|');
        getline(inFile, newAppointment.patientID, '|');
        getline(inFile, newAppointment.patientName, '|');
        getline(inFile, newAppointment.appointmentDate, '|');
        getline(inFile, newAppointment.appointmentType,'|');
        getline(inFile, newAppointment.priority);

        if (!newAppointment.appointmentNum.empty()) {
            // Insert the new appointment into the linked list
            head = insertAppointment(head, newAppointment);
        }
    }

    inFile.close();
    return head;
}

// Function to get the current date and time as a string
string getCurrentDateTime() {
    time_t now = time(0);
    struct tm* timeInfo;
    char buffer[80];
    time(&now);
    timeInfo = localtime(&now);
    strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M", timeInfo);
    return string(buffer);
}

// Function to ask the user to schedule an appointment
void scheduleAppointment(patientNode* patientList, appointmentNode*&
appointmentList, queue<Appointment>& appointmentQueue) {
    Appointment newAppointment;
    cin.ignore();
```

```
    // Prompt user to enter patient ID
    string patientID;
    cout << "Enter patient ID: ";
    cin >> newAppointment.patientID;

    // Search for the patient with the entered ID
    patientNode* currentPatient = patientList;
    while (currentPatient != nullptr) {
        if (currentPatient->patientData.patientID ==
newAppointment.patientID) {
            // Patient found, set appointment details
            newAppointment.patientName =
currentPatient->patientData.name;

            cout << "Please enter the schedule for patient: " <<
newAppointment.patientName << endl;
            cin.ignore();
            cout << "Enter appointment date and time (YYYY-MM-DD
00:00): ";
            getline(cin, newAppointment.appointmentDate);

            cout << "Enter appointment type: ";
            getline(cin, newAppointment.appointmentType);

            // Auto-increment appointment ID
            latestAppointmentNumber++;
            newAppointment.appointmentNum =
to_string(latestAppointmentNumber);

            // Set priority to 0 so that it will appear below
emergency cases in the appointment list
            newAppointment.priority = appointmentPriority;
            newAppointment.priority =
to_string(appointmentPriority);

            // Add the new appointment to the queue
            appointmentQueue.push(newAppointment);

            // Insert the new appointment into the linked list
```

```
            appointmentList = insertAppointment(appointmentList,
newAppointment);

            // Append the new appointment to the file without
rewriting the entire list
            ofstream outFile("appointment.txt", ios::out |
ios::app);
            if (!outFile.is_open()) {
                cerr << "Error opening file: " << "appointment.txt"
<< endl;
                return;
            }

            outFile << newAppointment.appointmentNum << "|" <<
newAppointment.patientID << "|"
                    << newAppointment.patientName << "|" <<
newAppointment.appointmentDate << "|"
                    << newAppointment.appointmentType <<"|"<<
newAppointment.priority << "\n";

            outFile.close();
            if (outFile.fail()) {
                cerr << "Error closing file: " << "appointment.txt"
<< endl;
            }

            cout << "Appointment scheduled successfully." << endl;
            return;
        }

        currentPatient = currentPatient->next;
    }

    // Patient with the entered ID not found
    cout << "Patient with ID " << patientID << " not found." <<
endl;
}

// Function to schedule a new appointment
```

```
void currentAppointment(patientNode* patientList, appointmentNode*&
appointmentList, queue<Appointment>& appointmentQueue) {
    Appointment newAppointment;
    cin.ignore();

    cout << "Enter Patient ID for the appointment: ";
    cin >> newAppointment.patientID;

    // Search for the patient with the entered ID
    patientNode* currentPatient = patientList;
    while (currentPatient != nullptr) {
        if (currentPatient->patientData.patientID ==
newAppointment.patientID) {
            // Patient found, set appointment details
            newAppointment.patientName =
currentPatient->patientData.name;

            // Check if the appointment already exists
            appointmentNode* currentAppointment = appointmentList;
            while (currentAppointment != nullptr) {
                if (currentAppointment->appointmentData.patientID ==
newAppointment.patientID &&

currentAppointment->appointmentData.appointmentDate ==
getCurrentDateTime()) {
                    cout << "Appointment for this patient already
exists for today." << endl;
                    return;
                }
                currentAppointment = currentAppointment->next;
            }

            // Set appointmentDateTime to current date and time
            newAppointment.appointmentDate = getCurrentDateTime();
            // Auto-increment appointment ID
            latestAppointmentNumber++;
            newAppointment.appointmentNum =
to_string(latestAppointmentNumber);
```

```
            // Set priority to 0 so that it will appear below
emergency cases in the appointment list
            newAppointment.priority = appointmentPriority;
            newAppointment.priority =
to_string(appointmentPriority);

            cout << "Please enter the schedule for patient: " <<
newAppointment.patientName << endl;
            cout << "Enter Appointment Type: ";
            cin.ignore();
            getline(cin, newAppointment.appointmentType);

            // Add the new appointment to the queue
            appointmentQueue.push(newAppointment);

            // Insert the new appointment into the linked list
            appointmentList = insertAppointment(appointmentList,
newAppointment);

            // Append the new patient to the file without rewriting
the entire list
            ofstream outFile("appointment.txt", ios::out |
ios::app);
            if (!outFile.is_open()) {
                cerr << "Error opening file: " << "appointment.txt"
<< endl;
                return;
            }

            outFile << newAppointment.appointmentNum << "|" <<
newAppointment.patientID << "|"
                    << newAppointment.patientName << "|" <<
newAppointment.appointmentDate << "|"
                    << newAppointment.appointmentType <<"|"<<
newAppointment.priority << "\n";

            outFile.close();
            if (outFile.fail()) {
                cerr << "Error closing file: " << "appointment.txt"
<< endl;
```

```
            }

            cout << "Appointment scheduled successfully. Appointment
ID: " << newAppointment.appointmentNum << endl;
            return;
        }
        currentPatient = currentPatient->next;
    }

    // Patient with the entered ID not found
    cout << "Patient with the entered ID not found." << endl;
}

// Function to remove appointment data
void updateAppointmentData(appointmentNode* head) {
    string appointmentNum;
    cout << "Enter the appointment number to update: ";
    cin >> appointmentNum;

    appointmentNode* current = head;

    while (current != nullptr) {
        if (current->appointmentData.appointmentNum ==
appointmentNum) {
            // Appointment found, allow the user to update
information
            cout << "Enter new details for the appointment:" <<
endl;
            cout << " Patient ID: ";
            cin.ignore(); // Ignore newline character from previous
input
            getline(cin, current->appointmentData.patientID);

            cout << " Patient Name: ";
            getline(cin, current->appointmentData.patientName);

            cout << " Appointment date: ";
            getline(cin, current->appointmentData.appointmentDate);

            // Ask the user if it's an emergency appointment
```

```
            cout << " Is it an emergency appointment? (1 for Yes, 0
for No): ";
            int isEmergency;
            cin >> isEmergency;

            if (isEmergency == 1) {
                // Prompt the user to select the level of emergency
                cout << " Select the level of emergency (1 for Green
Zone, 2 for Yellow Zone, 3 for Red Zone): ";
                int emergencyLevel;
                cin >> emergencyLevel;

                appointmentPriority += emergencyLevel;
                current->appointmentData.priority =
to_string(appointmentPriority);

                // Set the appointment type based on the selected
level
                if (emergencyLevel == 1) {
                    current->appointmentData.appointmentType =
"Emergency : Green Zone";
                } else if (emergencyLevel == 2) {
                    current->appointmentData.appointmentType =
"Emergency : Yellow Zone";
                } else if (emergencyLevel == 3) {
                    current->appointmentData.appointmentType =
"Emergency : Red Zone";
                } else {
                    cout << "Invalid emergency level. Setting the
appointment as a regular appointment." << endl;
                    current->appointmentData.appointmentType =
"Regular";
                    current->appointmentData.priority = "0";
                }
            } else {
                // Non-emergency appointment
                current->appointmentData.appointmentType =
"Regular";
                current->appointmentData.priority = "0";
            }
```

```
            // Open the new file for writing
            ofstream outFileTemp("appointment_temp.txt");
            if (!outFileTemp.is_open()) {
                cerr << "Error opening file: " <<
"appointment_temp.txt" << endl;
                return;
            }

            // Write all appointments to the new file
            appointmentNode* tempCurrent = head;
            while (tempCurrent != nullptr) {
                outFileTemp <<
tempCurrent->appointmentData.appointmentNum << "|" <<
tempCurrent->appointmentData.patientID << "|"
                            <<
tempCurrent->appointmentData.patientName << "|" <<
tempCurrent->appointmentData.appointmentDate << "|"
                            <<
tempCurrent->appointmentData.appointmentType << "|" <<
tempCurrent->appointmentData.priority << "\n";
                tempCurrent = tempCurrent->next;
            }

            outFileTemp.close();
            if (outFileTemp.fail()) {
                cerr << "Error closing file: " <<
"appointment_temp.txt" << endl;
            }

            // Remove the old file
            remove("appointment.txt");

            // Rename the new file to the original file name
            rename("appointment_temp.txt", "appointment.txt");

            cout << "Appointment information updated successfully."
<< endl;
            return;
        }
```

```
        current = current->next;
    }

    // Appointment not found
    cout << "Appointment with number " << appointmentNum << " not
found. Please enter the appropriate appointment number." << endl;
}

// Function that lets the user edit appointment information based on
the appointment ID
void removeAppointment(appointmentNode*& head, const string&
appointmentNum) {
    if (head == nullptr) {
        cout << "No appointment to remove. The list is empty." <<
endl;
        return;
    }

    appointmentNode* current = head;
    appointmentNode* previous = nullptr;

    // Traverse the list to find the appointment with the specified
ID
    while (current != nullptr &&
current->appointmentData.appointmentNum != appointmentNum) {
        previous = current;
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Appointment with number " << appointmentNum << "
not found." << endl;
        return;
    }

    // Remove the appointment node from the list
    if (previous == nullptr) {
        // The appointment to be removed is the head of the list
        head = head->next;
```

```
    } else {
        previous->next = current->next;
    }

    // Remove the appointment data from the file
    ofstream outFileTemp("appointment_temp.txt");
    if (!outFileTemp.is_open()) {
        cerr << "Error opening file: " << "appointment_temp.txt" <<
endl;
        return;
    }

    appointmentNode* tempCurrent = head;
    while (tempCurrent != nullptr) {
        outFileTemp << tempCurrent->appointmentData.appointmentNum
<< "|" << tempCurrent->appointmentData.patientID << "|"
                    << tempCurrent->appointmentData.patientName <<
"|" << tempCurrent->appointmentData.appointmentDate << "|"
                    << tempCurrent->appointmentData.appointmentType
<< "|" << tempCurrent->appointmentData.priority << "\n";
        tempCurrent = tempCurrent->next;
    }

    outFileTemp.close();
    if (outFileTemp.fail()) {
        cerr << "Error closing file: " << "appointment_temp.txt" <<
endl;
    }

    // Remove the old file
    remove("appointment.txt");

    // Rename the new file to the original file name
    rename("appointment_temp.txt", "appointment.txt");

    delete current;
    cout << "Appointment with number " << appointmentNum << "
removed successfully." << endl;
}
```

```
//EMERGENCY FUNCTION

// Function to sort emergency appointments based on emergency
priority and date
appointmentNode* sortEmergencyAppointments(appointmentNode* head) {
    // Check if the list is empty or has only one element
    if (head == nullptr || head->next == nullptr) {
        // No need to sort if the list is empty or has only one
element
        return head;
    }

    // Initialize the sorted list for emergency appointments
    appointmentNode* emergencySortedList = nullptr;
    // Pointer to traverse the original list
    appointmentNode* current = head;

    while (current != nullptr) {
        // Store the next node to prevent losing the connection
during sorting
        appointmentNode* next = current->next;

        if (emergencySortedList == nullptr ||
            current->appointmentData.priority >
emergencySortedList->appointmentData.priority) {
            // If the sorted list is empty or the current
appointment has higher priority,
            // insert the current appointment at the beginning
            current->next = emergencySortedList;
            emergencySortedList = current;
        } else {
            // Otherwise, find the correct position in the sorted
list
            appointmentNode* temp = emergencySortedList;
            while (temp->next != nullptr &&
                   current->appointmentData.priority <
temp->next->appointmentData.priority) {
                temp = temp->next;
            }
```

```
            // If priorities are the same, consider the date
            if (temp->next != nullptr &&
current->appointmentData.priority ==
temp->next->appointmentData.priority) {
                while (temp->next != nullptr &&
                       current->appointmentData.appointmentDate >
temp->next->appointmentData.appointmentDate) {
                    temp = temp->next;
                }
            }

            // Insert the current appointment after temp
            current->next = temp->next;
            temp->next = current;
        }

        // Move to the next node in the original list
        current = next;
    }

    // Return the sorted list of emergency appointments
    return emergencySortedList;
}

// Function to display emergency appointments in a formatted table
void displayEmergencyAppointmentSchedule(const appointmentNode*
head) {
    // Pointer to traverse the list
    const appointmentNode* current = head;

    // Display column headers
    cout << left << setw(10) << "Number" << setw(15) << "PatientID"
<< setw(20) << "PatientName"
         << setw(25) << "Appointment Date & Time" << setw(15) <<
"AppointmentType" << endl;

    // Iterate through the list and display emergency appointments
    while (current != nullptr && current->appointmentData.priority
!= "0") {
```

```
        cout << left << setw(10) <<
current->appointmentData.appointmentNum << setw(15) <<
current->appointmentData.patientID
             << setw(20) << current->appointmentData.patientName <<
setw(25) << current->appointmentData.appointmentDate
             << setw(15) << current->appointmentData.appointmentType
<< endl;

        // Move to the next node in the list
        current = current->next;
    }
}

// Function to schedule emergency appointment
void scheduleEmergencyAppointment(patientNode* patientList,
appointmentNode*& appointmentList, queue<Appointment>&
appointmentQueue) {
    // Create a new appointment object
    Appointment newAppointment;
    // Clear any existing input buffer
    cin.ignore();
    // Variable to store the user's option for emergency type
    int option;
    // Display options for emergency type
    cout << "Please choose which type of emergency :" << endl;
    cout << "1. Green Zone" << endl;
    cout << "2. Yellow Zone" << endl;
    cout << "3. Red Zone" << endl;
    cout << "Number : ";
    // Read user's option
    cin >> option;
    // Prompt user to enter patient ID for the emergency appointment
    cout << "Enter Patient ID for the appointment: ";
    // Read patient ID
    cin >> newAppointment.patientID;

    // Search for the patient with the entered ID
    patientNode* currentPatient = patientList;
    while (currentPatient != nullptr) {
```

```
        if (currentPatient->patientData.patientID ==
newAppointment.patientID) {
            // Patient found, set appointment details
            newAppointment.patientName =
currentPatient->patientData.name;

            // Check if the appointment already exists
            appointmentNode* currentAppointment = appointmentList;
            while (currentAppointment != nullptr) {
                if (currentAppointment->appointmentData.patientID ==
newAppointment.patientID &&

currentAppointment->appointmentData.appointmentDate ==
getCurrentDateTime()) {
                    cout << "Appointment for this patient already
exists for today." << endl;
                    return;
                }
                currentAppointment = currentAppointment->next;
            }

            // Set appointmentDateTime to current date and time
            newAppointment.appointmentDate = getCurrentDateTime();
            // Auto-increment appointment ID
            latestAppointmentNumber++;
            newAppointment.appointmentNum =
to_string(latestAppointmentNumber);

            // Set priority to 1 because it is an emergency case, so
that it will appear on top of the list
            if (option == 1) {
                // Increment the overall appointment priority based
on the selected emergency zone
                appointmentPriority += option;
                // Update the priority field in the appointment
                newAppointment.priority =
to_string(appointmentPriority);
                // Reset the overall appointment priority
                appointmentPriority = 0;
```

```
                cout << "Appointment type = Emergency : Green Zone"
<< endl;
                newAppointment.appointmentType = "Emergency : Green
Zone";
            } else if (option == 2) {
                appointmentPriority += option;
                newAppointment.priority =
to_string(appointmentPriority);
                appointmentPriority = 0;
                cout << "Appointment type = Emergency : Yellow Zone"
<< endl;
                newAppointment.appointmentType = "Emergency : Yellow
Zone";
            } else if (option == 3) {
                appointmentPriority += option;
                newAppointment.priority =
to_string(appointmentPriority);
                appointmentPriority = 0;
                cout << "Appointment type = Emergency : Red Zone" <<
endl;
                newAppointment.appointmentType = "Emergency : Red
Zone";
            } else {
                // Handle invalid option, for example, print an
error message
                cerr << "Error: Invalid option." << endl;
            }

            // Add the new appointment to the queue
            appointmentQueue.push(newAppointment);

            // Insert the new appointment into the linked list
            appointmentList = insertAppointment(appointmentList,
newAppointment);

            // Append the new patient to the file without rewriting
the entire list
            ofstream outFile("appointment.txt", ios::out |
ios::app);
            if (!outFile.is_open()) {
```

```
                cerr << "Error opening file: " << "appointment.txt"
<< endl;
                return;
            }

            // Write the new appointment details to the file
            outFile << newAppointment.appointmentNum << "|" <<
newAppointment.patientID << "|"
                    << newAppointment.patientName << "|" <<
newAppointment.appointmentDate << "|"
                    << newAppointment.appointmentType << "|" <<
newAppointment.priority << "\n";

            // Close the file
            outFile.close();
            // Check if the file closing was successful
            if (outFile.fail()) {
                cerr << "Error closing file: " << "appointment.txt"
<< endl;
            }

            // Display a success message
            cout << "Appointment scheduled successfully. Appointment
ID: " << newAppointment.appointmentNum << endl;
            // Return from the function after scheduling the
appointment
            return;
        }
        // Move to the next patient in the list
        currentPatient = currentPatient->next;
    }

    // Patient with the entered ID not found
    cout << "Patient with the entered ID not found." << endl;
}

// Function to write emergency appointments to a file
void writeEmergencyAppointments(const appointmentNode* head) {
    // Open the file in truncation mode (clear existing content)
    ofstream outFile("emergencies.txt", ios::out | ios::trunc);
```

```
    // Check if the file is successfully opened
    if (!outFile.is_open()) {
        cerr << "Error opening file: emergencies.txt" << endl;
        return;
    }

    // Pointer to traverse the list
    const appointmentNode* current = head;

    // Open the original appointment file for reading
    ifstream inFile("appointment.txt");
    // Check if the file is successfully opened
    if (!inFile.is_open()) {
        cerr << "Error opening file: appointment.txt" << endl;
        // Close the emergencies file
        outFile.close();
        return;
    }

    // Variable to store each line of the file
    string line;
    // Iterate through each line in the original appointment file
    while (getline(inFile, line)) {
        // Assuming the data in the file is stored in the format:
"num|ID|Name|Date|Type|Priority"
        // Use stringstream to extract individual fields from the
line
        stringstream ss(line);
        string num, ID, name, date, type, priority;
        getline(ss, num, '|');
        getline(ss, ID, '|');
        getline(ss, name, '|');
        getline(ss, date, '|');
        getline(ss, type, '|');
        getline(ss, priority, '|');

        // Check if the appointment is an emergency (priority 1, 2,
or 3)
        if (priority == "1" || priority == "2" || priority == "3") {
            // Write only emergency appointments to the new file
```

```
            outFile << num << "|" << ID << "|" << name << "|" <<
date << "|" << type << "|" << priority << "\n";
        }
    }

    // Close the input file
    inFile.close();

    // Close the output file
    outFile.close();
    // Check if the file closing was successful
    if (outFile.fail()) {
        cerr << "Error closing file: emergencies.txt" << endl;
    } else {
        // Display a success message
        cout << "Emergency appointments written to emergencies.txt
successfully." << endl;
    }
}

// Function to display medical record for each patient based on
patientID
void medicalRecord(const appointmentNode* head) {
    // Prompt user to enter patient ID
    cout << "Enter patient ID: ";
    // Variable to store the entered patient ID
    string patientID;
    // Read patient ID from the user
    cin >> patientID;

    // Pointer to traverse the list
    const appointmentNode* current = head;
    // Flag to indicate whether appointments were found for the
specified patient
    bool found = false;

    // Collect appointments for the specified patient
    vector<Appointment> patientAppointments;
```

```
    // Iterate through the list and collect appointments for the
specified patient
    while (current != nullptr) {
        if (current->appointmentData.patientID == patientID) {
            // Add the appointment to the vector
            patientAppointments.push_back(current->appointmentData);
            // Set the flag to true, indicating that appointments
were found
            found = true;
        }

        // Move to the next node in the list
        current = current->next;
    }

    // Display the medical record in a table
    if (found) {
        cout << "Medical Record for Patient ID: " << patientID <<
endl;
        cout << left << setw(15) << "Number"
             << setw(20) << "Appointment Date"
             << setw(15) << "Appointment Type" << endl;

        // Iterate through the collected appointments and display
them in a table
        for (const auto& appointment : patientAppointments) {
            cout << left << setw(15) << appointment.appointmentNum
                 << setw(20) << appointment.appointmentDate
                 << setw(15) << appointment.appointmentType << endl;
        }
    } else {
        // Display a message if no appointments were found for the
specified patient
        cout << "No appointments found for patient ID: " <<
patientID << endl;
    }
}


// Display usermenu
```

```
void userMenu(){
    cout << "--------------------Welcome to Budiman Private Hospital
Patient Management System--------------------" << endl;
    cout << "Please select your number: " << endl;
    cout << "1. Register a new patient." << endl; //dah
    cout << "2. Schedule a patient appointment." << endl; //dah
    cout << "3. View and update patient information." << endl; //dah
    cout << "4. Sort patient records." << endl; //dah
    cout << "5. Search for a patient." << endl; //dah
    cout << "6. View and update appointments." << endl;//dah
    cout << "7. Enter and access medical records." << endl;
    cout << "8. Display patient list." << endl; //dah
    cout << "9. Display appointment schedule." << endl; //dah
    cout << "10. Add an emergency case." << endl;//dah
    cout << "11. Process emergency cases." << endl;//dah
    cout << "12. Exit" << endl; //dah
}

//Main function
int main() {
    // Initialize patient list and read data from the patients file
    patientNode* patientList = nullptr;
    patientList = readPatientsFile("patients.txt");

    // Initialize appointment list and read data from the
appointments file
    appointmentNode* appointmentList = nullptr;
    appointmentList = readAppointmentsFile("appointment.txt");

    // Find the latest appointment number
    latestAppointmentNumber =
findMaxAppointmentNumber(appointmentList);

    // Initialize appointment queue
    queue<Appointment> appointmentQueue;

    char decision;
    int number;
    string removeAppointmentNums;
```

```
    // Main loop for user interaction
    do {
        // Display user menu
        userMenu();
        cout << "Number: ";
        cin >> number;

        // Switch statement to handle user input
        switch (number) {
            case 1:
                // Register new patients
                do {
                    newPatient(patientList);
                    cout << "Do you have any other patient to
register? (Y/N): ";
                    cin >> decision;
                } while (decision == 'Y' || decision == 'y');
                break;

            case 2:
                // Schedule appointments
                do {
                    cout << "Choose which appointment you want: " <<
endl;
                    cout << "1. Appointment for now" << endl;
                    cout << "2. Schedule Appointment for another day
and time" << endl;
                    cout << "Number : ";
                    int choose;
                    cin >> choose;

                    if (choose == 1) {
                        currentAppointment(patientList,
appointmentList, appointmentQueue);
                        cout << "Do you have any other appointment
to schedule (Y/N): ";
                        cin >> decision;
                    } else if (choose == 2) {
                        scheduleAppointment(patientList,
appointmentList, appointmentQueue);
```

```
                        cout << "Do you have any other appointment
to schedule (Y/N): ";
                        cin >> decision;
                    }
                } while (decision == 'Y' || decision == 'y');
                latestAppointmentNumber =
findMaxAppointmentNumber(appointmentList);
                break;
            case 3:
                //View and update patient
                int num;
                cout << "Choose which option you want to do :
"<<endl;
                cout << "1. Edit patient data"<<endl;
                cout << "2. Delete patient data"<<endl;
                cout << "Number : ";
                cin >> num;
                if (num==1){
                    displayPatientsFile(patientList);
                    updatePatientData(patientList);
                    }
                else if(num==2){
                    string removePatientID;
                    displayPatientsFile(patientList);
                    cout <<"Enter the patient ID that you want to
remove : ";
                    cin >> removePatientID;
                    removePatient(patientList, removePatientID);
                    }
                break;
            case 4:
                //Sort patient record by name or id
                int choose;
                cout <<"1. Sort by name" <<endl;
                cout <<"2. Sort by ID" <<endl;
                cout <<"Please choose : ";
                cin >> choose;

                if (choose == 1){
                bubbleSortPatientsName(patientList);
```

```
                cout << "Patient records sorted alphabetically by
name." << endl;
                displaySortedPatientNames(patientList);
                } else if(choose == 2){
                bubbleSortPatientsID(patientList);
                cout << "Patient records sorted alphabetically by
ID." << endl;
                displaySortedPatientNames(patientList);
                }
                    break;
            case 5:
                // Searching for a patient
                do {
                    searchPatientData(patientList);
                    cout << "Do you have any other patient to
search? (Y/N): ";
                    cin >> decision;
                } while (decision == 'Y' || decision == 'y');
                break;
            case 6:
                //View and update appointment
                cout << "Choose which option you want to do :
"<<endl;
                cout << "1. Edit appointment data"<<endl;
                cout << "2. Delete appointment data"<<endl;
                cout <<"Number : ";
                cin >> num;
                if (num==1){
                    displayAppointmentSchedule(appointmentList);
                    updateAppointmentData(appointmentList);
                    }
                else if(num==2){
                    string removeAppointmentNum;
                    displayAppointmentSchedule(appointmentList);
                    cout <<"Enter the appointment number that you
want to remove : ";
                    cin >> removeAppointmentNum;
                    removeAppointment(appointmentList,
removeAppointmentNum);
                    }
```

```
                break;
            case 7:
                //Enter and access medical records
                do{
                medicalRecord(appointmentList);

                cout << "Do you want to check another patient's
medical record? (y/n): ";
                cin >> decision;
                } while (decision == 'y' || decision == 'Y');
             break;
            case 8:
                // Displaying all registered patients
                cout << "Registered Patients:" << endl;
                displayPatientsFile(patientList);
                break;
            case 9:
                appointmentList = sortAppointments(appointmentList);
                displayAppointmentSchedule(appointmentList);
                // Implement the functionality for displaying
appointment schedule
                break;
            case 10:
                //Add an emergency case
                do {

                    scheduleEmergencyAppointment(patientList,
appointmentList, appointmentQueue);

                    cout <<"Do you have any other appointment to
schedule (Y/N): ";
                    cin >> decision;
                } while (decision == 'Y' || decision == 'y');
                latestAppointmentNumber =
findMaxAppointmentNumber(appointmentList);
                writeEmergencyAppointments(appointmentList);

                break;
            case 11:
                //Process and delete emergency case
```

```
                appointmentList =
sortEmergencyAppointments(appointmentList);

displayEmergencyAppointmentSchedule(appointmentList);
                cout <<"Enter appointment ID that you want to
process and remove , if not type (N): ";
                cin >> removeAppointmentNums;
                removeAppointment(appointmentList,
removeAppointmentNums);
                writeEmergencyAppointments(appointmentList);

                break;
            case 12:
                // Exiting the program
                while (patientList != nullptr) {
                    patientNode* temp = patientList;
                    patientList = patientList->next;
                    delete temp;
                }
                cout << "--------------------Thankyou for using
Budiman Private Hospital Patient Management
System--------------------" << endl;
                exit(0);
            default:
                cout << "Invalid option. Please try again." << endl;
        }

        cout << "Do you want to perform another operation? (Y/N): ";
        cin >> decision;

    } while (decision == 'Y' || decision == 'y');

    cout << "--------------------Thankyou for using Budiman Private
Hospital Patient Management System--------------------" << endl;
    // Clean up the memory used by the linked list
    while (patientList != nullptr) {
        patientNode* temp = patientList;
        patientList = patientList->next;
        delete temp;
    }
```

```
    while (appointmentList != nullptr) {
        appointmentNode* temp = appointmentList;
        appointmentList = appointmentList->next;
        delete temp;
    }

    return 0;

}
```

4. EXAMPLE OF OUTPUT

4.1 Main Menu

```
--------------------Welcome to Budiman Private Hospital Patient Management System--------------------
Please select your number:
1. Register a new patient.
2. Schedule a patient appointment.
3. View and update patient information.
4. Sort patient records.
5. Search for a patient.
6. View and update appointments.
7. Enter and access medical records.
8. Display patient list.
9. Display appointment schedule.
10. Add an emergency case.
11. Process emergency cases.
12. Exit
Number:
```

4.2 Register New Patient

```
Number: 1
Please enter patient details as below:
 Name: Amirul Azim
 ID (Axxxxx): A00076
 Gender (Male/Female): Male
 Date of Birth (DD/MM/YYYY): 06/04/2002
 Address: 97, Jalan Seri Menanti, Muar
 Phone number (0123456789): 01112106050
Do you have any other patient to register? (Y/N): n
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

(patients.txt)

Final Project > patients.txt

```
1  Arron Adam|A00025|Male|29/04/1997|45, Star Garden, Tanjung  Malim|0159992341
2  Chung Yee Maa|A01095|Female|07/09/2001|88, Taman Intan Berlian, Hulu Selangor|0174563210
3  Dean Michael|A00910|Male|11/11/2001|10, Taman Cahaya, Tanjung Malim|0165643291
4  Saravanan Arumugan|A00083|Male|05/08/1997|Lot 15, Pinggiran Sungai, Tanjung Malim|0129878765
5  Aizad Adzraus|A00012|Male|05/04/2000|81, Taman Suka Ria, Cyberjaya|0193240287
6  Haikal Jaki|A00096|Male|03/12/2003|6 , Jalan Harmoni , Johor Bahru|0197169627
7  Nur Anis|A00099|Female|15/06/2001|80, Jalan Sabariah , Kota Bharu|0132831360
8  Amirul Azim|A00076|Male|06/04/2002|97, Jalan Seri Menanti, Muar|01112106050
```

4.3 Schedule a Patient Appointment

4.3.1 Appointment for current time

```
Number: 2
Choose which appointment you want:
1. Appointment for now
2. Schedule Appointment for another day and time
Number : 1
Enter Patient ID for the appointment: A00076
Please enter schedule for patient: Amirul Azim
Enter Appointment Type: Medical checkup
Appointment scheduled successfully. Appointment ID: 1006
Do you have any other appointment to schedule (Y/N): n
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.3.2 Appointment for Another Day and Time

```
Number: 2
Choose which appointment you want:
1. Appointment for now
2. Schedule Appointment for another day and time
Number : 2
Enter patient ID: A00076
Please enter schedule for patient: Amirul Azim
Enter appointment date and time (YYYY-MM-DD 00:00): 2024-03-25 14:00
Enter appointment type: Dentist
Appointment scheduled successfully.
Do you have any other appointment to schedule (Y/N): n
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

(appointment.txt)

appointment.txt

Final Project > appointment.txt

```
1004|A00910|Dean Michael|2024-02-02 20:00|Emergency : Red Zone|3
1005|A00096|Haikal Jaki|2024-02-02 20:00|Emergency : Yellow Zone|2
1003|A01095|Chung Yee Maa|2024-02-02 20:00|Emergency : Green Zone|1
1001|A00012|Aizad Adzraus|2024-02-02 19:59|Medical checkup|0
1002|A00025|Arron Adam|2024-05-10 12:00|Dentist|0
1006|A00076|Amirul Azim|2024-02-03 16:41|Medical checkup|0
1007|A00076|Amirul Azim|2024-03-25 14:00|Dentist|0
```

4.4 View and Update Patient Data

4.4.1 Edit Patient Data

```
Number: 3
Choose which option you want to do :
1. Edit patient data
2. Delete patient data
Number : 1
Name                ID          Gender  Birth Date  Address                                          Phone Number
Arron Adam          A00025      Male    29/04/1997  45, Star Garden, Tanjung  Malim                  0159992341
Chung Yee Maa       A01095      Female  07/09/2001  88, Taman Intan Berlian, Hulu Selangor           0174563210
Dean Michael        A00910      Male    11/11/2001  10, Taman Cahaya, Tanjung Malim                  0165643291
Saravanan Arumugan  A00083      Male    05/08/1997  Lot 15, Pinggiran Sungai, Tanjung Malim          0129878765
Aizad Adzraus       A00012      Male    05/04/2000  81, Taman Suka Ria, Cyberjaya                    0193240287
Haikal Jaki         A00096      Male    03/12/2003  6 , Jalan Harmoni , Johor Bahru                  0197169627
Nur Anis            A00099      Female  15/06/2001  80, Jalan Sabariah , Kota Bharu                  0132831360
Amirul Azim         A00076      Male    06/04/2002  97, Jalan Seri Menanti, Muar                     01112106050
Enter the Patient ID to update: A00076
Enter new details for the patient:
 Name: Amirul Adzim
 Gender(M/F): Male
 Date of Birth: 06/04/2002
 Address: 97, Jalan Seri Menanti, Muar
 Phone number: 01112106050
Patient information updated successfully.
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

```
Enter the Patient ID to update: A00023
Patient with ID A00023 not found. Please enter the appropriate patient ID.
```

4.4.2 Delete Patient Data

```
Number : 2
Name                ID          Gender  Birth Date  Address                                          Phone Number
Arron Adam          A00025      Male    29/04/1997  45, Star Garden, Tanjung  Malim                  0159992341
Chung Yee Maa       A01095      Female  07/09/2001  88, Taman Intan Berlian, Hulu Selangor           0174563210
Dean Michael        A00910      Male    11/11/2001  10, Taman Cahaya, Tanjung Malim                  0165643291
Saravanan Arumugan  A00083      Male    05/08/1997  Lot 15, Pinggiran Sungai, Tanjung Malim          0129878765
Aizad Adzraus       A00012      Male    05/04/2000  81, Taman Suka Ria, Cyberjaya                    0193240287
Haikal Jaki         A00096      Male    03/12/2003  6 , Jalan Harmoni , Johor Bahru                  0197169627
Nur Anis            A00099      Female  15/06/2001  80, Jalan Sabariah , Kota Bharu                  0132831360
Amirul Adzim        A00076      Male    06/04/2002  97, Jalan Seri Menanti, Muar                     01112106050
Enter the patient ID that you want to remove : A00099
Patient with ID A00099 removed successfully.
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

```
Enter the patient ID that you want to remove : A00022
Patient with ID A00022 not found.
```

(patients.txt) before

Final Project > patients.txt

```
1  Arron Adam|A00025|Male|29/04/1997|45, Star Garden, Tanjung  Malim|0159992341
2  Chung Yee Maa|A01095|Female|07/09/2001|88, Taman Intan Berlian, Hulu Selangor|0174563210
3  Dean Michael|A00910|Male|11/11/2001|10, Taman Cahaya, Tanjung Malim|0165643291
4  Saravanan Arumugan|A00083|Male|05/08/1997|Lot 15, Pinggiran Sungai, Tanjung Malim|0129878765
5  Aizad Adzraus|A00012|Male|05/04/2000|81, Taman Suka Ria, Cyberjaya|0193240287
6  Haikal Jaki|A00096|Male|03/12/2003|6 , Jalan Harmoni , Johor Bahru|0197169627
7  Nur Anis|A00099|Female|15/06/2001|80, Jalan Sabariah , Kota Bharu|0132831360
8  Amirul Adzim|A00076|Male|06/04/2002|97, Jalan Seri Menanti, Muar|01112106050
```

(patients.txt) after

Final Project > patients.txt

```
1  Arron Adam|A00025|Male|29/04/1997|45, Star Garden, Tanjung  Malim|0159992341
2  Chung Yee Maa|A01095|Female|07/09/2001|88, Taman Intan Berlian, Hulu Selangor|0174563210
3  Dean Michael|A00910|Male|11/11/2001|10, Taman Cahaya, Tanjung Malim|0165643291
4  Saravanan Arumugan|A00083|Male|05/08/1997|Lot 15, Pinggiran Sungai, Tanjung Malim|0129878765
5  Aizad Adzraus|A00012|Male|05/04/2000|81, Taman Suka Ria, Cyberjaya|0193240287
6  Haikal Jaki|A00096|Male|03/12/2003|6 , Jalan Harmoni , Johor Bahru|0197169627
7  Amirul Adzim|A00076|Male|06/04/2002|97, Jalan Seri Menanti, Muar|01112106050
8
```

4.5 Sort Patient Record

4.5.1 Sort by ID

```
Number: 4
1. Sort by name
2. Sort by ID
Please choose : 1
Patient records sorted alphabetically by name.
Sorted Patient Names:
Name                ID          Gender  Birth Date  Address                                    Phone Number
Aizad Adzraus       A00012      Male    05/04/2000  81, Taman Suka Ria, Cyberjaya              0193240287
Amirul Adzim        A00076      Male    06/04/2002  97, Jalan Seri Menanti, Muar               01112106050
Arron Adam          A00025      Male    29/04/1997  45, Star Garden, Tanjung  Malim           0159992341
Chung Yee Maa       A01095      Female  07/09/2001  88, Taman Intan Berlian, Hulu Selangor     0174563210
Dean Michael        A00910      Male    11/11/2001  10, Taman Cahaya, Tanjung Malim            0165643291
Haikal Jaki         A00096      Male    03/12/2003  6 , Jalan Harmoni , Johor Bahru            0197169627
Nur Anis            A00099      Female  15/06/2001  80, Jalan Sabariah , Kota Bharu            0132831360
Saravanan Arumugan  A00083      Male    05/08/1997  Lot 15, Pinggiran Sungai, Tanjung Malim    0129878765
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.5.2 Sort by Name

```
Number: 4
1. Sort by name
2. Sort by ID
Please choose : 2
Patient records sorted alphabetically by ID.
Sorted Patient Names:
Name                ID          Gender  Birth Date  Address                                    Phone Number
Aizad Adzraus       A00012      Male    05/04/2000  81, Taman Suka Ria, Cyberjaya              0193240287
Arron Adam          A00025      Male    29/04/1997  45, Star Garden, Tanjung  Malim           0159992341
Amirul Adzim        A00076      Male    06/04/2002  97, Jalan Seri Menanti, Muar               01112106050
Saravanan Arumugan  A00083      Male    05/08/1997  Lot 15, Pinggiran Sungai, Tanjung Malim    0129878765
Haikal Jaki         A00096      Male    03/12/2003  6 , Jalan Harmoni , Johor Bahru            0197169627
Nur Anis            A00099      Female  15/06/2001  80, Jalan Sabariah , Kota Bharu            0132831360
Dean Michael        A00910      Male    11/11/2001  10, Taman Cahaya, Tanjung Malim            0165643291
Chung Yee Maa       A01095      Female  07/09/2001  88, Taman Intan Berlian, Hulu Selangor     0174563210
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.6 Search patient by ID

```
Number: 5
Enter the Patient ID to search: A00076
Name                ID              Gender  Birth Date  Address                                                 Phone Number
Amirul Adzim        A00076          Male    06/04/2002  97, Jalan Seri Menanti, Muar                            01112106050
Do you have any other patient to search? (Y/N): n
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

```
Enter the Patient ID to search: A00011
Patient with ID A00011 not found.
```

4.7 Update Appointment by Appointment Number

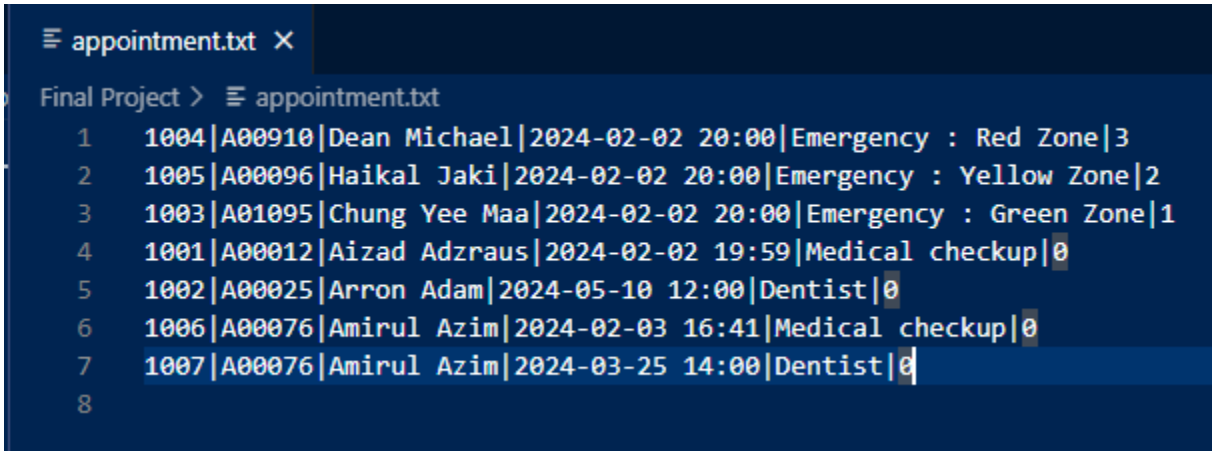
4.7.1 Edit Appointment Data

```
Number: 6
Choose which option you want to do :
1. Edit appointment data
2. Delete appointment data
Number : 1
Number    PatientID      PatientName         Appointment Date & Time  AppointmentType
1001      A00012         Aizad Adzraus       2024-02-02 19:59         Medical checkup
1004      A00910         Dean Michael        2024-02-02 20:00         Emergency : Red Zone
1003      A01095         Chung Yee Maa       2024-02-02 20:00         Emergency : Green Zone
1005      A00096         Haikal Jaki         2024-02-02 20:00         Emergency : Yellow Zone
1006      A00076         Amirul Azim         2024-02-03 16:41         Medical checkup
1007      A00076         Amirul Azim         2024-03-25 14:00         Dentist
1002      A00025         Arron Adam          2024-05-10 12:00         Dentist
Enter the appointment number to update: 1006
Enter new details for the appointment:
 Patient ID: A00076
 Patient Name: Amirul Adzim
 Appointment date: 2024-02-03 16:41
 Is it an emergency appointment? (1 for Yes, 0 for No): 0
Appointment information updated successfully.
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

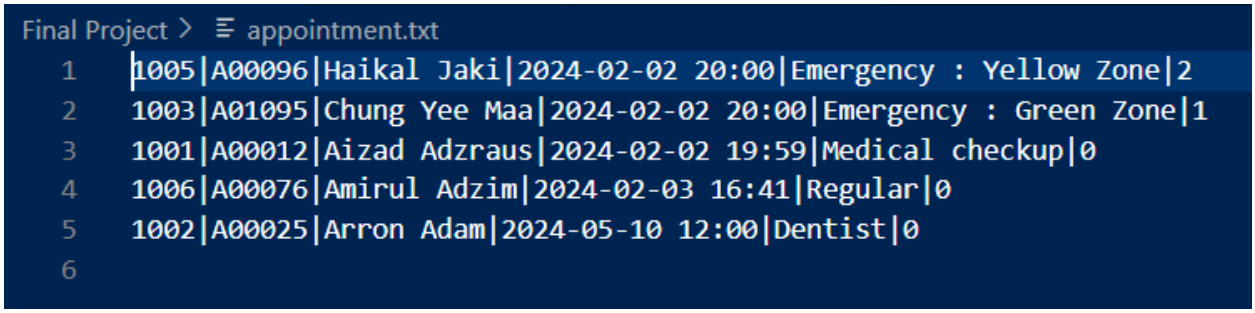
4.7.2 Delete Appointment Data

```
Number: 6
Choose which option you want to do :
1. Edit appointment data
2. Delete appointment data
Number : 2
Number    PatientID      PatientName         Appointment Date & Time  AppointmentType
1001      A00012         Aizad Adzraus       2024-02-02 19:59         Medical checkup
1005      A00096         Haikal Jaki         2024-02-02 20:00         Emergency : Yellow Zone
1003      A01095         Chung Yee Maa       2024-02-02 20:00         Emergency : Green Zone
1004      A00910         Dean Michael        2024-02-02 20:00         Emergency : Red Zone
1006      A00076         Amirul Adzim        2024-02-03 16:41         Regular
1007      A00076         Amirul Azim         2024-03-25 14:00         Dentist
1002      A00025         Arron Adam          2024-05-10 12:00         Dentist
Enter the appointment number that you want to remove : 1007
Appointment with number 1007 removed successfully.
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

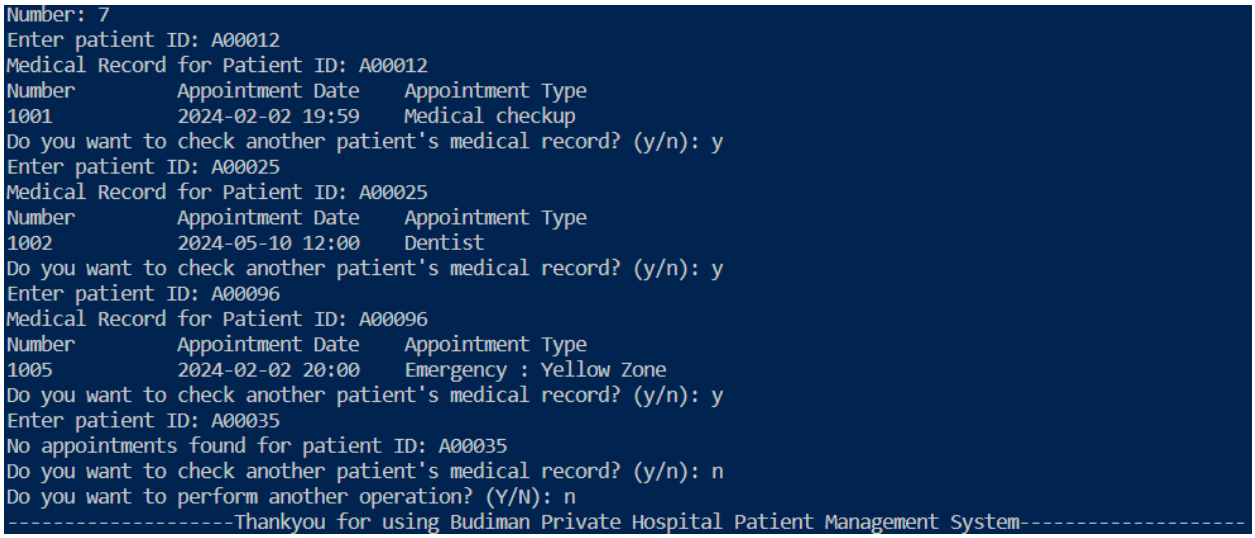
(patients.txt) before



(patients.txt) after



4.8 Enter and Access Medical Records



4.9 Display Patients List

```
Number: 8
Registered Patients:
Name                ID          Gender  Birth Date  Address                                          Phone Number
Arron Adam          A00025      Male    29/04/1997  45, Star Garden, Tanjung  Malim                  0159992341
Chung Yee Maa       A01095      Female  07/09/2001  88, Taman Intan Berlian, Hulu Selangor           0174563210
Dean Michael        A00910      Male    11/11/2001  10, Taman Cahaya, Tanjung Malim                  0165643291
Saravanan Arumugan  A00083      Male    05/08/1997  Lot 15, Pinggiran Sungai, Tanjung Malim          0129878765
Aizad Adzraus       A00012      Male    05/04/2000  81, Taman Suka Ria, Cyberjaya                    0193240287
Haikal Jaki         A00096      Male    03/12/2003  6 , Jalan Harmoni , Johor Bahru                  0197169627
Nur Anis            A00099      Female  15/06/2001  80, Jalan Sabariah , Kota Bharu                  0132831360
Amirul Adzim        A00076      Male    06/04/2002  97, Jalan Seri Menanti, Muar                     01112106050
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.10 Display Appointment List

```
Number: 9
Number    PatientID     PatientName         Appointment Date & Time  AppointmentType
1001      A00012        Aizad Adzraus       2024-02-02 19:59         Medical checkup
1005      A00096        Haikal Jaki         2024-02-02 20:00         Emergency : Yellow Zone
1003      A01095        Chung Yee Maa       2024-02-02 20:00         Emergency : Green Zone
1004      A00910        Dean Michael        2024-02-02 20:00         Emergency : Red Zone
1006      A00076        Amirul Adzim        2024-02-03 16:41         Regular
1002      A00025        Arron Adam          2024-05-10 12:00         Dentist
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.11 Add an Emergency Case

```
Number: 10
Please choose which type of emergency :
1. Green Zone
2. Yellow Zone
3. Red Zone
Number : 3
Enter Patient ID for the appointment: A00076
Appointment type = Emergency : Red Zone
Appointment scheduled successfully. Appointment ID: 1007
Do you have any other appointment to schedule (Y/N): n
Emergency appointments written to emergencies.txt successfully.
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.12 Process Emergency case

```
Number: 11
Number    PatientID     PatientName         Appointment Date & Time  AppointmentType
1004      A00910        Dean Michael        2024-02-02 20:00         Emergency : Red Zone
1007      A00076        Amirul Adzim        2024-02-04 09:00         Emergency : Red Zone
1005      A00096        Haikal Jaki         2024-02-02 20:00         Emergency : Yellow Zone
1003      A01095        Chung Yee Maa       2024-02-02 20:00         Emergency : Green Zone
Enter appointment ID that you want to process and remove , if not type (N): 1004
Appointment with number 1004 removed successfully.
Emergency appointments written to emergencies.txt successfully.
Do you want to perform another operation? (Y/N): n
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```

4.13 Exit

```
Number: 12
--------------------Thankyou for using Budiman Private Hospital Patient Management System--------------------
```