#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <ctype.h>

/\* MAIN HEALTHCARE MANAGEMENT SYSTEM \*/

// Main menu function

void displayMainMenu() {

printf("\nHealthcare Management System\n");

printf("1. User Registration\n");

printf("2. Doctor Appointment Scheduling\n");

printf("3. Treatment Management\n");

printf("4. Blood Bank Inventory\n");

printf("5. Medical Store Management\n");

printf("6. Billing/Payment System\n");

printf("7. Exit\n");

printf("Enter your choice: ");

}

/\* 1. USER REGISTRATION MODULE \*/

struct Patient {

int id;

char name[50];

int age;

char gender[10];

char contact[15];

};

void addPatient() {

struct Patient p;

FILE \*fp = fopen("patients.txt", "a");

if (fp == NULL) {

printf("Error opening file! ᓚ₍⑅^..^₎♡\nPlease check the file permissions.\n");

return;

}

printf("\n [ ADD NEW PATIENT ]\n");

printf("--Enter Patient ID (example:101): ");

scanf("%d", &p.id);

getchar();

printf("--Enter Patient Name (example: Eren Yeager): ");

fgets(p.name, sizeof(p.name), stdin);

p.name[strcspn(p.name, "\n")] = 0;

printf("--Enter Patient Age (example: 25): ");

scanf("%d", &p.age);

getchar();

printf("--Enter Patient Gender (example: Male/Female/Other): ");

fgets(p.gender, sizeof(p.gender), stdin);

p.gender[strcspn(p.gender, "\n")] = 0;

printf("--Enter Patient Contact (example: 01712345678): ");

fgets(p.contact, sizeof(p.contact), stdin);

p.contact[strcspn(p.contact, "\n")] = 0;

fprintf(fp, "%d %s %d %s %s\n", p.id, p.name, p.age, p.gender, p.contact);

fclose(fp);

printf("\nPatient registered successfully!ᓚ₍⑅^..^₎♡\n");

}

void viewPatients() {

struct Patient p;

FILE \*fp = fopen("patients.txt", "r");

if (fp == NULL) {

printf("\nNo patients registered yet! ᓚ₍⑅^..^₎♡Please add patients first.\n");

return;

}

printf("\n [ REGISTERED PATIENTS ]\n");

while (fscanf(fp, "%d %s %d %s %s", &p.id, p.name, &p.age, p.gender, p.contact) != EOF) {

printf("\nID: %d, Name: %s, Age: %d, Gender: %s, Contact: %s\n", p.id, p.name, p.age, p.gender, p.contact);

}

fclose(fp);

printf("\nEnd of patient listᓚ₍⑅^..^₎♡.\n");

}

void searchPatient() {

int searchId;

printf("\n [ SEARCH PATIENT ]\n");

printf("--Enter Patient ID to search (example:101): ");

scanf("%d", &searchId);

getchar();

struct Patient p;

FILE \*fp = fopen("patients.txt", "r");

if (fp == NULL) {

printf("\nNo patients registered yet!ᓚ₍⑅^..^₎♡ Please add patients first.\n");

return;

}

int found = 0;

while (fscanf(fp, "%d %s %d %s %s", &p.id, p.name, &p.age, p.gender, p.contact) != EOF) {

if (p.id == searchId) {

printf("\nPatient Found!(^. .^)⟆\\n");

printf("~ID: %d\n,\n~Name: %s,\n~Age: %d,\n~Gender: %s,\n~Contact: %s\n", p.id, p.name, p.age, p.gender, p.contact);

found = 1;

break;

}

}

fclose(fp);

if (!found) {

printf("\nPatient with ID %d not found!ᓚ₍⑅^..^₎♡\n Please check the ID and try again.\n", searchId);

}

}

void updatePatient() {

int updateId;

printf("\n [ UPDATE PATIENT ]\n");

printf("--Enter Patient ID to update (example:101): ");

scanf("%d", &updateId);

getchar();

struct Patient p;

FILE \*fp = fopen("patients.txt", "r");

FILE \*temp = fopen("temp.txt", "w");

if (fp == NULL || temp == NULL) {

printf("Error! Please check the file permissionsᓚ₍⑅^..^₎♡.\n");

return;

}

int found = 0;

while (fscanf(fp, "%d %s %d %s %s", &p.id, p.name, &p.age, p.gender, p.contact) != EOF) {

if (p.id == updateId) {

found = 1;

printf("--Enter new details for Patient ID %d (example:101):\n", p.id);

printf("--Enter Name (example: Zeke Yeager): ");

fgets(p.name, sizeof(p.name), stdin);

p.name[strcspn(p.name, "\n")] = 0;

printf("--Enter Age (example: 25) : ");

scanf("%d", &p.age);

getchar();

printf("--Enter Gender (example: Male/Female/Other): ");

fgets(p.gender, sizeof(p.gender), stdin);

p.gender[strcspn(p.gender, "\n")] = 0;

printf("--Enter Contact (example: 01712345678): ");

fgets(p.contact, sizeof(p.contact), stdin);

p.contact[strcspn(p.contact, "\n")] = 0;

}

fprintf(temp, "%d %s %d %s %s\n", p.id, p.name, p.age, p.gender, p.contact);

}

fclose(fp);

fclose(temp);

if (found) {

remove("patients.txt");

rename("temp.txt", "patients.txt");

printf("\nPatient details updated successfullyᓚ₍⑅^..^₎♡!\n");

} else {

printf("\nPatient with ID %d not found! Please check the ID and try again.\n", updateId);

remove("temp.txt");

}

}

void deletePatient() {

int deleteId;

printf("\n [ DELETE PATIENT ]\n");

printf("--Enter Patient ID to delete (example:101): ");

scanf("%d", &deleteId);

getchar();

struct Patient p;

FILE \*fp = fopen("patients.txt", "r");

FILE \*temp = fopen("temp.txt", "w");

if (fp == NULL || temp == NULL) {

printf("\nError opening file!ᓚ₍⑅^..^₎♡\n Please check the file permissions.\n");

return;

}

int found = 0;

while (fscanf(fp, "%d %s %d %s %s", &p.id, p.name, &p.age, p.gender, p.contact) != EOF) {

if (p.id == deleteId) {

found = 1;

printf("\nPatient with ID %d deleted successfully!ᓚ₍⑅^..^₎♡\n", deleteId);

} else {

fprintf(temp, "%d %s %d %s %s\n", p.id, p.name, p.age, p.gender, p.contact);

}

}

fclose(fp);

fclose(temp);

if (found) {

remove("patients.txt");

rename("temp.txt", "patients.txt");

} else {

printf("\nPatient with ID %d not found!ᓚ₍⑅^..^₎♡ Please check the ID and try again.\n", deleteId);

remove("temp.txt");

}

}

void userRegistrationSystem() {

int choice;

do {

printf("\n~~~ User Registration For A PATIENT ~~~\n");

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

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

printf(" 3. Search Patient\n");

printf(" 4. Update Patient\n");

printf(" 5. Delete Patient\n");

printf(" 6. Back to Main Menu\n");

printf("\n--Enter your choice (1-6): ");

scanf("%d", &choice);

getchar();

switch (choice) {

case 1: addPatient(); break;

case 2: viewPatients(); break;

case 3: searchPatient(); break;

case 4: updatePatient(); break;

case 5: deletePatient(); break;

case 6: printf("\nReturning to main menu... ᓚ₍⑅^..^₎♡\n"); break;

default: printf("\nInvalid choice!ᓚ₍⑅^..^₎♡ Please enter a number between 1 and 6.\n");

}

} while (choice != 6);

}

// 2. Doctor Appointment Scheduling Module

#define MAX\_PATIENTS 100

#define MAX\_DOCTORS 25

#define MAX\_NAME\_LEN 50

#define MAX\_MOBILE\_LEN 15

#define FILENAME "appointments.txt"

typedef struct {

int doctorID;

char name[MAX\_NAME\_LEN];

char department[MAX\_NAME\_LEN];

float fee;

} Doctor;

typedef struct {

int patientID;

char name[MAX\_NAME\_LEN];

char mobileNumber[MAX\_MOBILE\_LEN];

char backupMobile[MAX\_MOBILE\_LEN];

int doctorID;

int timeSlot;

float paymentAmount;

int missed;

int emergency;

} Appointment;

Doctor doctors[MAX\_DOCTORS] = {

{0, "Kajol Kanti Das", "Cardiology", 1000},

{1, "Bidhan Roy", "Cardiology", 1000},

{2, "Farooq Ahmed", "Cardiology", 1000},

{3, "Rakhi Hasan", "Cardiology", 1000},

{4, "Rupashree Bisshas", "Cardiology", 1000},

{5, "Sumit Roy", "Neurology", 1500},

{6, "Sakhi Ali Majumdar", "Neurology", 1500},

{7, "Mohiuddin Issa", "Neurology", 1500},

{8, "Badal Ahmed", "Neurology", 1500},

{9, "Kulsum Khana", "Neurology", 1500},

{10, "Munir Khanum Chowdhury", "Orthopaedics", 1500},

{11, "Arors Ahmed", "Orthopaedics", 1500},

{12, "Mr JK Khan", "Orthopaedics", 1500},

{13, "Prottoy Das", "Orthopaedics", 1500},

{14, "Affan Anwar", "Orthopaedics", 1500},

{15, "Muhammad Mehrab Hussain", "Dermatology", 800},

{16, "Shahida Akhtar", "Dermatology", 800},

{17, "Zakir Hussain", "Dermatology", 800},

{18, "Mohammed Masud", "Dermatology", 800},

{19, "Liton Das", "Dermatology", 800},

{20, "Iqbal Choudhary", "Pediatrics", 1000},

{21, "Sukumar Kanti Das", "Pediatrics", 1000},

{22, "Saila Hussain", "Pediatrics", 1000},

{23, "Sharmin Akter", "Pediatrics", 1000},

{24, "Turna Barua", "Pediatrics", 1000}

};

Appointment appointments[MAX\_PATIENTS];

void displayDoctors() {

printf("\nAvailable Doctors:\n");

for (int i = 0; i < MAX\_DOCTORS; i++) {

printf("%d. Dr. %s - %s (Fee: %.2f)\n", i + 1, doctors[i].name, doctors[i].department, doctors[i].fee);

}

}

void bookAppointment(int \*appointmentCount) {

Appointment newAppointment;

int doctorID, timeSlot;

printf("\nEnter patient ID (Example: 1001): ");

scanf("%d", &newAppointment.patientID);

printf("Enter patient name (Example: Saima Hossain): ");

getchar();

fgets(newAppointment.name, MAX\_NAME\_LEN, stdin);

newAppointment.name[strcspn(newAppointment.name, "\n")] = 0;

printf("Enter mobile number (Example: 018973700987): ");

scanf("%s", newAppointment.mobileNumber);

printf("Enter backup mobile number (Example: 01890079539): ");

scanf("%s", newAppointment.backupMobile);

displayDoctors();

printf("Select doctor by number (1-%d): ", MAX\_DOCTORS);

scanf("%d", &doctorID);

doctorID--;

newAppointment.doctorID = doctorID;

printf("\nChoose time slot:\n");

printf("1. 9:00-11:00 AM\n");

printf("2. 3:00-6:00 PM\n");

printf("3. 8:00-11:00 PM\n");

printf("Enter time slot (1-3): ");

scanf("%d", &timeSlot);

newAppointment.timeSlot = timeSlot - 1;

newAppointment.paymentAmount = doctors[doctorID].fee \* 0.30;

newAppointment.missed = 0;

newAppointment.emergency = 0;

appointments[\*appointmentCount] = newAppointment;

(\*appointmentCount)++;

printf("\nAppointment booked successfully!\n");

printf("Payment of %.2f made (30%% of doctor's fee).\n", newAppointment.paymentAmount);

}

void displayAppointments(int appointmentCount) {

printf("\nList of Appointments:\n");

for (int i = 0; i < appointmentCount; i++) {

printf("Patient ID: %d, Name: %s, Doctor: %s, Time Slot: %d, Payment: %.2f, Missed: %s, Emergency: %s\n",

appointments[i].patientID,

appointments[i].name,

doctors[appointments[i].doctorID].name,

appointments[i].timeSlot + 1,

appointments[i].paymentAmount,

appointments[i].missed ? "Yes" : "No",

appointments[i].emergency ? "Yes" : "No");

}

}

void cancelAppointment(int \*appointmentCount) {

int patientID;

printf("\nEnter patient ID to cancel the appointment (Example: 1001): ");

scanf("%d", &patientID);

int found = 0;

for (int i = 0; i < \*appointmentCount; i++) {

if (appointments[i].patientID == patientID) {

for (int j = i; j < \*appointmentCount - 1; j++) {

appointments[j] = appointments[j + 1];

}

(\*appointmentCount)--;

found = 1;

printf("\nAppointment canceled successfully!\n");

break;

}

}

if (!found) {

printf("Appointment not found for Patient ID: %d\n", patientID);

}

}

void missedAppointment(int \*appointmentCount) {

int patientID;

printf("\nEnter patient ID to mark the appointment as missed (Example: 1001): ");

scanf("%d", &patientID);

int found = 0;

for (int i = 0; i < \*appointmentCount; i++) {

if (appointments[i].patientID == patientID) {

appointments[i].missed = 1;

found = 1;

printf("\nAppointment marked as missed.\n");

break;

}

}

if (!found) {

printf("Appointment not found for Patient ID: %d\n", patientID);

}

}

void emergencyAppointment(int \*appointmentCount) {

Appointment newAppointment;

int doctorID, timeSlot;

printf("\nEnter patient ID (Example: 1001): ");

scanf("%d", &newAppointment.patientID);

printf("Enter patient name (Example: Saima Hossain): ");

getchar();

fgets(newAppointment.name, MAX\_NAME\_LEN, stdin);

newAppointment.name[strcspn(newAppointment.name, "\n")] = 0;

printf("Enter mobile number (Example: 018973700987): ");

scanf("%s", newAppointment.mobileNumber);

printf("Enter backup mobile number (Example: 01890079539): ");

scanf("%s", newAppointment.backupMobile);

displayDoctors();

printf("Select doctor by number (1-%d): ", MAX\_DOCTORS);

scanf("%d", &doctorID);

doctorID--;

newAppointment.doctorID = doctorID;

printf("\nChoose time slot:\n");

printf("1. 9:00-11:00 AM\n");

printf("2. 3:00-6:00 PM\n");

printf("3. 8:00-11:00 PM\n");

printf("Enter time slot (1-3): ");

scanf("%d", &timeSlot);

newAppointment.timeSlot = timeSlot - 1;

newAppointment.paymentAmount = doctors[doctorID].fee \* 0.30;

newAppointment.emergency = 1;

newAppointment.missed = 0;

appointments[\*appointmentCount] = newAppointment;

(\*appointmentCount)++;

printf("\nEmergency appointment booked successfully!\n");

printf("Payment of %.2f made (30%% of doctor's fee).\n", newAppointment.paymentAmount);

}

float getDoctorFee(int doctorID) {

return doctors[doctorID].fee;

}

void helpline() {

printf("\nHelpline Numbers:\n");

printf("9990930, 89000979\n");

}

void saveAppointments(int appointmentCount) {

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

if (file == NULL) {

printf("Error opening file for saving appointments.\n");

return;

}

fwrite(&appointments, sizeof(Appointment), appointmentCount, file);

fclose(file);

}

void loadAppointments(int \*appointmentCount) {

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

if (file == NULL) {

printf("No previous appointments found.\n");

return;

}

\*appointmentCount = fread(&appointments, sizeof(Appointment), MAX\_PATIENTS, file);

fclose(file);

}

void doctorAppointmentSystem() {

int appointmentCount = 0;

int choice;

loadAppointments(&appointmentCount);

while(1) {

printf("\nDoctors Appointment System\n");

printf("1. Book an Appointment\n");

printf("2. View All Appointments\n");

printf("3. Cancel Appointment\n");

printf("4. Mark Missed Appointment\n");

printf("5. Book Emergency Appointment\n");

printf("6. Helpline Number\n");

printf("7. Back to Main Menu\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1:

bookAppointment(&appointmentCount);

break;

case 2:

displayAppointments(appointmentCount);

break;

case 3:

cancelAppointment(&appointmentCount);

break;

case 4:

missedAppointment(&appointmentCount);

break;

case 5:

emergencyAppointment(&appointmentCount);

break;

case 6:

helpline();

break;

case 7:

saveAppointments(appointmentCount);

printf("Returning to main menu...\n");

return;

default:

printf("Invalid choice. Please try again.\n");

}

}

}

// 3. Treatment Management Module

struct Treatment {

int patientID;

char name[50];

int age;

char mobile[15];

char treatmentType[50];

char date[15];

char result[100];

struct Treatment\* next;

struct Treatment\* prev;

};

struct Treatment\* head = NULL;

const char\* availableTreatments[] = {

"X-Ray", "MRI", "Blood\_Test", "COVID-19\_Test", "CT\_Scan",

"ECG", "Endoscopy", "Ultrasound", "Dialysis", "Chemotherapy",

"Physiotherapy", "Vaccination", "Surgery", "Eye\_Test", "Dental\_Checkup"

};

int numTreatments = sizeof(availableTreatments) / sizeof(availableTreatments[0]);

void viewAvailableTreatments() {

printf("\nAvailable Treatments:\n");

for (int i = 0; i < numTreatments; i++) {

printf("%d. %s\n", i + 1, availableTreatments[i]);

}

}

void addTreatment() {

struct Treatment\* newTreatment = (struct Treatment\*)malloc(sizeof(struct Treatment));

printf("Enter Patient ID: ");

scanf("%d", &newTreatment->patientID);

printf("Enter Name: ");

scanf("%s", newTreatment->name);

printf("Enter Age: ");

scanf("%d", &newTreatment->age);

printf("Enter Mobile Number: ");

scanf("%s", newTreatment->mobile);

viewAvailableTreatments();

int choice;

printf("Select Treatment Type (Enter Number): ");

scanf("%d", &choice);

if (choice < 1 || choice > numTreatments) {

printf("Invalid choice! Defaulting to 'General Checkup'.\n");

strcpy(newTreatment->treatmentType, "General Checkup");

} else {

strcpy(newTreatment->treatmentType, availableTreatments[choice - 1]);

}

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

scanf("%s", newTreatment->date);

strcpy(newTreatment->result, "Pending");

newTreatment->next = head;

newTreatment->prev = NULL;

if (head != NULL) {

head->prev = newTreatment;

}

head = newTreatment;

printf("\nTreatment record added successfully!\n");

}

void viewTreatments() {

struct Treatment\* temp = head;

if (temp == NULL) {

printf("No records found!\n");

return;

}

printf("\n%-10s %-20s %-5s %-15s %-20s %-15s %-10s\n",

"ID", "Name", "Age", "Mobile", "Treatment", "Date", "Result");

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

while (temp != NULL) {

printf("%-10d %-20s %-5d %-15s %-20s %-15s %-10s\n",

temp->patientID, temp->name, temp->age,

temp->mobile, temp->treatmentType, temp->date, temp->result);

temp = temp->next;

}

printf("\n");

}

void sortTreatmentsByPatientID() {

if (head == NULL || head->next == NULL) {

printf("\nNo sorting needed.\n");

return;

}

int swapped;

struct Treatment \*ptr1;

struct Treatment \*lptr = NULL;

do {

swapped = 0;

ptr1 = head;

while (ptr1->next != lptr) {

if (ptr1->patientID > ptr1->next->patientID) {

// Swap the data instead of nodes to maintain list structure

int tempID = ptr1->patientID;

ptr1->patientID = ptr1->next->patientID;

ptr1->next->patientID = tempID;

char tempName[50];

strcpy(tempName, ptr1->name);

strcpy(ptr1->name, ptr1->next->name);

strcpy(ptr1->next->name, tempName);

int tempAge = ptr1->age;

ptr1->age = ptr1->next->age;

ptr1->next->age = tempAge;

char tempMobile[15];

strcpy(tempMobile, ptr1->mobile);

strcpy(ptr1->mobile, ptr1->next->mobile);

strcpy(ptr1->next->mobile, tempMobile);

char tempType[50];

strcpy(tempType, ptr1->treatmentType);

strcpy(ptr1->treatmentType, ptr1->next->treatmentType);

strcpy(ptr1->next->treatmentType, tempType);

char tempDate[15];

strcpy(tempDate, ptr1->date);

strcpy(ptr1->date, ptr1->next->date);

strcpy(ptr1->next->date, tempDate);

char tempResult[100];

strcpy(tempResult, ptr1->result);

strcpy(ptr1->result, ptr1->next->result);

strcpy(ptr1->next->result, tempResult);

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

} while (swapped);

printf("\nTreatments sorted by Patient ID.\n");

}

void generatePatientInvoice(int patientID) {

struct Treatment\* temp = head;

int found = 0;

printf("\n\n\t\t\t The Healthcare Management System\n");

printf("\t\t\t 24/7 Hotline: 017-12345678\n");

printf("\t\t\t Email: info@healthcare-system.com\n");

printf("================================================================================\n");

printf("\n\t\t\t\t PATIENT INVOICE\n\n");

// Find all treatments for this patient

while (temp != NULL) {

if (temp->patientID == patientID) {

found = 1;

// Patient info (from first treatment record)

printf(" Patient ID: %-40d\n", temp->patientID);

printf(" Name: %-47s\n", temp->name);

printf(" Age: %-3d\n", temp->age);

printf(" Mobile: %-40s\n", temp->mobile);

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

printf(" Treatment History:\n");

printf(" +---------------------+--------------+---------------+\n");

printf(" | Treatment Type | Date | Result |\n");

printf(" +---------------------+--------------+---------------+\n");

break;

}

temp = temp->next;

}

if (!found) {

printf(" No records found for Patient ID: %d\n", patientID);

return;

}

// Print all treatments for this patient

temp = head;

while (temp != NULL) {

if (temp->patientID == patientID) {

printf(" | %-19s | %-12s | %-13s |\n",

temp->treatmentType, temp->date, temp->result);

}

temp = temp->next;

}

printf(" +---------------------+--------------+---------------+\n");

printf("\n================================================================================\n");

printf(" Important Notes:\n");

printf(" 1. Please bring this invoice for any future references\n");

printf(" 2. Contact our hotline for any emergency or query\n");

printf(" 3. Payment should be made within 7 days of treatment\n");

printf(" 4. Keep your patient ID safe for future appointments\n");

printf("================================================================================\n\n");

}

void searchPatientInTreatment() {

char name[50];

int id;

printf("Enter Patient ID or Name to search: ");

if (scanf("%d", &id) == 1) {

struct Treatment\* temp = head;

while (temp != NULL) {

if (temp->patientID == id) {

printf("Patient Found: %s\n", temp->name);

return;

}

temp = temp->next;

}

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

} else {

scanf("%s", name);

struct Treatment\* temp = head;

while (temp != NULL) {

if (strcmp(temp->name, name) == 0) {

printf("Patient Found: ID: %d, Age: %d, Mobile: %s\n", temp->patientID, temp->age, temp->mobile);

return;

}

temp = temp->next;

}

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

}

}

void updateResult() {

int id;

printf("Enter Patient ID to update result: ");

scanf("%d", &id);

struct Treatment\* temp = head;

while (temp != NULL) {

if (temp->patientID == id) {

printf("Enter new result: ");

scanf("%s", temp->result);

printf("\nResult updated successfully!\n");

return;

}

temp = temp->next;

}

printf("\nPatient ID not found!\n");

}

void deleteTreatment() {

int id;

printf("Enter Patient ID to delete record: ");

scanf("%d", &id);

struct Treatment \*temp = head, \*prev = NULL;

while (temp != NULL && temp->patientID != id) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("\nPatient ID not found!\n");

return;

}

if (prev == NULL) {

head = temp->next;

} else {

prev->next = temp->next;

}

if (temp->next != NULL) {

temp->next->prev = prev;

}

free(temp);

printf("\nRecord deleted successfully!\n");

}

void saveToFile() {

FILE\* file = fopen("treatments.txt", "w");

if (!file) {

printf("Error opening file for writing!\n");

return;

}

struct Treatment\* temp = head;

while (temp != NULL) {

fprintf(file, "%d %s %d %s %s %s %s\n",

temp->patientID, temp->name, temp->age,

temp->mobile, temp->treatmentType, temp->date, temp->result);

temp = temp->next;

}

fclose(file);

printf("\nData saved successfully!\n");

}

void loadFromFile() {

FILE\* file = fopen("treatments.txt", "r");

if (!file) {

printf("No previous treatment records found.\n");

return;

}

int patientID, age;

char name[50], mobile[15], treatmentType[50], date[15], result[100];

// Clear existing list

while (head != NULL) {

struct Treatment\* temp = head;

head = head->next;

free(temp);

}

while (fscanf(file, "%d %s %d %s %s %s %s",

&patientID, name, &age, mobile, treatmentType, date, result) == 7) {

struct Treatment\* newTreatment = (struct Treatment\*)malloc(sizeof(struct Treatment));

newTreatment->patientID = patientID;

strcpy(newTreatment->name, name);

newTreatment->age = age;

strcpy(newTreatment->mobile, mobile);

strcpy(newTreatment->treatmentType, treatmentType);

strcpy(newTreatment->date, date);

strcpy(newTreatment->result, result);

newTreatment->next = head;

newTreatment->prev = NULL;

if (head != NULL) {

head->prev = newTreatment;

}

head = newTreatment;

}

fclose(file);

}

void treatmentManagementSystem() {

loadFromFile();

int choice;

while (1) {

printf("\n\nHealthcare Management System - Treatment Management\n");

printf("1. View Available Treatments\n");

printf("2. Add Treatment Record\n");

printf("3. View All Treatment Records\n");

printf("4. Update Test Result\n");

printf("5. Delete Treatment Record\n");

printf("6. Search Patient\n");

printf("7. Sort Treatment Records by Patient ID\n");

printf("8. Generate Patient Invoice\n");

printf("9. Save & Back to Main Menu\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: viewAvailableTreatments(); break;

case 2: addTreatment(); break;

case 3: viewTreatments(); break;

case 4: updateResult(); break;

case 5: deleteTreatment(); break;

case 6: searchPatientInTreatment(); break;

case 7: sortTreatmentsByPatientID(); break;

case 8: {

int id;

printf("Enter Patient ID for Invoice: ");

scanf("%d", &id);

generatePatientInvoice(id);

break;

}

case 9: saveToFile(); return;

default: printf("Invalid choice!\n");

}

}

}

// 4. Blood Bank Inventory Module

#define MAX\_BLOOD\_TYPES 4

#define MAX\_DONORS 100

#define MIN\_BLOOD\_STOCK\_ALERT 10

#define MAX\_DONOR\_NAME\_LENGTH 100

#define MAX\_CONTACT\_LENGTH 15

#define BLOOD\_DONORS\_FILE "blood\_donors.txt"

#define BLOOD\_STOCK\_FILE "blood\_stock.txt"

typedef enum {

A\_POSITIVE,

B\_POSITIVE,

AB\_POSITIVE,

O\_POSITIVE

} BloodType;

typedef struct {

char name[MAX\_DONOR\_NAME\_LENGTH];

int age;

char contact[MAX\_CONTACT\_LENGTH];

BloodType bloodType;

float quantityDonated;

char donationDate[20];

} Donor;

typedef struct {

BloodType bloodType;

float stock;

char expiryDate[20];

} BloodStock;

Donor donors[MAX\_DONORS];

BloodStock bloodStock[MAX\_BLOOD\_TYPES];

int totalDonors = 0;

// File handling functions

void loadDonorsFromFile() {

FILE \*file = fopen(BLOOD\_DONORS\_FILE, "r");

if (file) {

totalDonors = 0;

while (fscanf(file, "%[^|]|%d|%[^|]|%d|%f|%[^\n]\n",

donors[totalDonors].name,

&donors[totalDonors].age,

donors[totalDonors].contact,

(int\*)&donors[totalDonors].bloodType,

&donors[totalDonors].quantityDonated,

donors[totalDonors].donationDate) == 6) {

totalDonors++;

if (totalDonors >= MAX\_DONORS) break;

}

fclose(file);

}

}

void saveDonorsToFile() {

FILE \*file = fopen(BLOOD\_DONORS\_FILE, "w");

if (file) {

for (int i = 0; i < totalDonors; i++) {

fprintf(file, "%s|%d|%s|%d|%.2f|%s\n",

donors[i].name,

donors[i].age,

donors[i].contact,

donors[i].bloodType,

donors[i].quantityDonated,

donors[i].donationDate);

}

fclose(file);

}

}

void loadBloodStockFromFile() {

FILE \*file = fopen(BLOOD\_STOCK\_FILE, "r");

if (file) {

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

if (fscanf(file, "%d|%f|%[^\n]\n",

(int\*)&bloodStock[i].bloodType,

&bloodStock[i].stock,

bloodStock[i].expiryDate) != 3) {

// Initialize if reading fails

bloodStock[i].bloodType = i;

bloodStock[i].stock = 0.0;

strcpy(bloodStock[i].expiryDate, "0000-00-00");

}

}

fclose(file);

} else {

// Initialize if file doesn't exist

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

bloodStock[i].bloodType = i;

bloodStock[i].stock = 0.0;

strcpy(bloodStock[i].expiryDate, "0000-00-00");

}

}

}

void saveBloodStockToFile() {

FILE \*file = fopen(BLOOD\_STOCK\_FILE, "w");

if (file) {

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

fprintf(file, "%d|%.2f|%s\n",

bloodStock[i].bloodType,

bloodStock[i].stock,

bloodStock[i].expiryDate);

}

fclose(file);

}

}

void addDonor() {

if (totalDonors < MAX\_DONORS) {

Donor newDonor;

int type;

printf("Enter donor name: ");

scanf(" %[^\n]", newDonor.name);

printf("Enter donor age: ");

scanf("%d", &newDonor.age);

printf("Enter donor contact number: ");

scanf("%s", newDonor.contact);

do {

printf("Enter blood type (1: A+, 2: B+, 3: AB+, 4: O+): ");

scanf("%d", &type);

if (type < 1 || type > 4) {

printf("Invalid blood type. Please enter a valid blood type.\n");

}

} while (type < 1 || type > 4);

newDonor.bloodType = (BloodType)(type - 1);

printf("Enter quantity donated (in liters): ");

scanf("%f", &newDonor.quantityDonated);

time\_t t = time(NULL);

struct tm tm = \*localtime(&t);

sprintf(newDonor.donationDate, "%d-%02d-%02d", tm.tm\_year + 1900, tm.tm\_mon + 1, tm.tm\_mday);

donors[totalDonors++] = newDonor;

printf("Donor registered successfully.\n");

} else {

printf("Donor list is full.\n");

}

}

void updateBloodStock() {

int type;

do {

printf("Enter blood type to update stock (1: A+, 2: B+, 3: AB+, 4: O+): ");

scanf("%d", &type);

if (type < 1 || type > 4) {

printf("Invalid blood type. Please enter a valid blood type.\n");

}

} while (type < 1 || type > 4);

type--;

printf("Enter blood stock quantity (in liters): ");

scanf("%f", &bloodStock[type].stock);

printf("Enter expiry date of the blood stock (YYYY-MM-DD): ");

scanf("%s", bloodStock[type].expiryDate);

printf("Blood stock updated successfully.\n");

}

void checkBloodStock() {

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

if (bloodStock[i].stock < MIN\_BLOOD\_STOCK\_ALERT) {

printf("ALERT: Blood type %d is low on stock!\n", i + 1);

} else {

printf("Blood type %d has sufficient stock.\n", i + 1);

}

}

}

void ensureSufficientBlood() {

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

if (bloodStock[i].stock < MIN\_BLOOD\_STOCK\_ALERT) {

printf("Please consider donating more blood for blood type %d.\n", i + 1);

}

}

}

void displayDonationHistory() {

if (totalDonors == 0) {

printf("No donors registered yet.\n");

return;

}

printf("\nDonation History:\n");

printf("Name\tAge\tContact\t\tBlood Type\tQuantity (L)\tDonation Date\n");

for (int i = 0; i < totalDonors; i++) {

char \*bloodType;

switch (donors[i].bloodType) {

case A\_POSITIVE: bloodType = "A+"; break;

case B\_POSITIVE: bloodType = "B+"; break;

case AB\_POSITIVE: bloodType = "AB+"; break;

case O\_POSITIVE: bloodType = "O+"; break;

}

printf("%s\t%d\t%s\t%s\t%.2f\t\t%s\n", donors[i].name, donors[i].age, donors[i].contact, bloodType, donors[i].quantityDonated, donors[i].donationDate);

}

}

void displayBloodStock() {

printf("\nBlood Stock:\n");

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

char \*bloodType;

switch (i) {

case A\_POSITIVE: bloodType = "A+"; break;

case B\_POSITIVE: bloodType = "B+"; break;

case AB\_POSITIVE: bloodType = "AB+"; break;

case O\_POSITIVE: bloodType = "O+"; break;

}

printf("Blood Type: %s, Stock: %.2f liters, Expiry Date: %s\n", bloodType, bloodStock[i].stock, bloodStock[i].expiryDate);

}

}

void removeExpiredStock() {

time\_t t = time(NULL);

struct tm tm = \*localtime(&t);

char currentDate[20];

sprintf(currentDate, "%d-%02d-%02d", tm.tm\_year + 1900, tm.tm\_mon + 1, tm.tm\_mday);

for (int i = 0; i < MAX\_BLOOD\_TYPES; i++) {

if (strcmp(bloodStock[i].expiryDate, currentDate) < 0) {

printf("Blood type %d has expired. Removing from stock.\n", i + 1);

bloodStock[i].stock = 0;

}

}

}

void bloodBankSystem() {

// Initialize and load data

loadDonorsFromFile();

loadBloodStockFromFile();

int choice;

while (1) {

printf("\nBlood Bank Management System\n");

printf("1. Register Blood Donor\n");

printf("2. Update Blood Stock\n");

printf("3. Check Blood Stock\n");

printf("4. Ensure Sufficient Blood\n");

printf("5. View Donation History\n");

printf("6. View Blood Stock\n");

printf("7. Remove Expired Blood Stock\n");

printf("8. Back to Main Menu\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

addDonor();

break;

case 2:

updateBloodStock();

break;

case 3:

checkBloodStock();

break;

case 4:

ensureSufficientBlood();

break;

case 5:

displayDonationHistory();

break;

case 6:

displayBloodStock();

break;

case 7:

removeExpiredStock();

break;

case 8:

saveDonorsToFile();

saveBloodStockToFile();

printf("Returning to main menu...\n");

return;

default:

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

}

}

}

/\* 5. Medical Store Management Module \*/

#define MAX\_MEDICINES 100

#define MEDICAL\_FILENAME "medical.txt"

#define LOW\_STOCK\_THRESHOLD 5

typedef struct {

int id;

char name[50];

char category[30];

int quantity;

float price;

char mfg\_date[11]; // DD-MM-YYYY format

char exp\_date[11]; // DD-MM-YYYY format

} Medicine;

Medicine inventory[MAX\_MEDICINES];

int medicine\_count = 0;

void load\_inventory() {

FILE \*file = fopen(MEDICAL\_FILENAME, "r");

if(file) {

medicine\_count = 0;

while (fscanf(file, "%d|%[^|]|%[^|]|%d|%f|%[^|]|%[^\n]\n",

&inventory[medicine\_count].id,

inventory[medicine\_count].name,

inventory[medicine\_count].category,

&inventory[medicine\_count].quantity,

&inventory[medicine\_count].price,

inventory[medicine\_count].mfg\_date,

inventory[medicine\_count].exp\_date) == 7) {

medicine\_count++;

if (medicine\_count >= MAX\_MEDICINES) break;

}

fclose(file);

} else {

printf("No existing inventory found. Starting fresh.\n");

}

}

void save\_inventory() {

FILE \*file = fopen(MEDICAL\_FILENAME, "w");

if(file) {

for(int i = 0; i < medicine\_count; i++) {

fprintf(file, "%d|%s|%s|%d|%.2f|%s|%s\n",

inventory[i].id,

inventory[i].name,

inventory[i].category,

inventory[i].quantity,

inventory[i].price,

inventory[i].mfg\_date,

inventory[i].exp\_date);

}

fclose(file);

}

}

int is\_unique\_id(int id) {

for(int i = 0; i < medicine\_count; i++) {

if(inventory[i].id == id) {

printf("ID already exists! ");

return 0;

}

}

return 1;

}

int validate\_date(const char \*date) {

if(strlen(date) != 10 || date[2] != '-' || date[5] != '-') {

printf("Date must be in DD-MM-YYYY format with 2 digits for day and month\n");

return 0;

}

for(int i = 0; i < 10; i++) {

if(i == 2 || i == 5) continue;

if(!isdigit(date[i])) {

printf("Date must contain only digits except for separators\n");

return 0;

}

}

int day, month, year;

if(sscanf(date, "%d-%d-%d", &day, &month, &year) != 3) {

printf("Invalid date format\n");

return 0;

}

if(year < 1900 || year > 2100) {

printf("Year must be between 1900 and 2100\n");

return 0;

}

if(month < 1 || month > 12) {

printf("Month must be between 01 and 12\n");

return 0;

}

int days\_in\_month[] = {31,28,31,30,31,30,31,31,30,31,30,31};

if((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))

days\_in\_month[1] = 29;

if(day < 1 || day > days\_in\_month[month-1]) {

printf("Invalid day for month %d\n", month);

return 0;

}

return 1;

}

int is\_expired(const char \*exp\_date) {

int exp\_day, exp\_month, exp\_year;

if(sscanf(exp\_date, "%d-%d-%d", &exp\_day, &exp\_month, &exp\_year) != 3) {

return 0;

}

time\_t now = time(NULL);

struct tm \*tm = localtime(&now);

int current\_year = tm->tm\_year + 1900;

int current\_month = tm->tm\_mon + 1;

int current\_day = tm->tm\_mday;

if(exp\_year < current\_year) return 1;

if(exp\_year > current\_year) return 0;

if(exp\_month < current\_month) return 1;

if(exp\_month > current\_month) return 0;

return (exp\_day < current\_day);

}

void add\_medicine() {

if(medicine\_count >= MAX\_MEDICINES) {

printf("Inventory full! Cannot add more medicines.\n");

return;

}

Medicine new\_med;

do {

printf("Enter Medicine ID: ");

if(scanf("%d", &new\_med.id) != 1) {

while(getchar() != '\n');

printf("Invalid ID! ");

continue;

}

} while(!is\_unique\_id(new\_med.id));

while(getchar() != '\n');

printf("Enter Medicine Name: ");

fgets(new\_med.name, 50, stdin);

new\_med.name[strcspn(new\_med.name, "\n")] = '\0';

printf("Enter Medicine Category: ");

fgets(new\_med.category, 30, stdin);

new\_med.category[strcspn(new\_med.category, "\n")] = '\0';

do {

printf("Enter Quantity: ");

if(scanf("%d", &new\_med.quantity) != 1) {

while(getchar() != '\n');

printf("Invalid quantity! ");

new\_med.quantity = -1;

}

} while(new\_med.quantity < 0);

do {

printf("Enter Price: ");

if(scanf("%f", &new\_med.price) != 1) {

while(getchar() != '\n');

printf("Invalid price! ");

new\_med.price = -1;

}

} while(new\_med.price <= 0);

do {

printf("Enter Manufacturing Date (DD-MM-YYYY): ");

scanf("%10s", new\_med.mfg\_date);

while(getchar() != '\n');

if(!validate\_date(new\_med.mfg\_date)) {

printf("Invalid manufacturing date. Please try again.\n");

}

} while(!validate\_date(new\_med.mfg\_date));

do {

printf("Enter Expiry Date (DD-MM-YYYY): ");

scanf("%10s", new\_med.exp\_date);

while(getchar() != '\n');

if(!validate\_date(new\_med.exp\_date)) {

printf("Invalid expiry date. Please try again.\n");

}

} while(!validate\_date(new\_med.exp\_date));

if(is\_expired(new\_med.exp\_date)) {

printf("Warning: This medicine has already expired!\n");

printf("Do you still want to add it? (1=Yes, 0=No): ");

int confirm;

scanf("%d", &confirm);

while(getchar() != '\n');

if(confirm != 1) {

printf("Medicine not added.\n");

return;

}

}

inventory[medicine\_count++] = new\_med;

printf("Medicine added successfully!\n");

}

void update\_medicine() {

int id;

printf("Enter Medicine ID to update: ");

if(scanf("%d", &id) != 1) {

while(getchar() != '\n');

printf("Invalid ID!\n");

return;

}

while(getchar() != '\n');

for(int i = 0; i < medicine\_count; i++) {

if(inventory[i].id == id) {

printf("Current Details:\n");

printf("Name: %s\nCategory: %s\nQuantity: %d\nPrice: %.2f\nExpiry: %s\n",

inventory[i].name, inventory[i].category,

inventory[i].quantity, inventory[i].price, inventory[i].exp\_date);

int new\_quantity;

do {

printf("Enter new quantity (>= 0): ");

if(scanf("%d", &new\_quantity) != 1) {

while(getchar() != '\n');

printf("Invalid input! Please enter a number.\n");

new\_quantity = -1;

} else if(new\_quantity < 0) {

printf("Quantity cannot be negative!\n");

}

} while(new\_quantity < 0);

inventory[i].quantity = new\_quantity;

float new\_price;

do {

printf("Enter new price (> 0): ");

if(scanf("%f", &new\_price) != 1) {

while(getchar() != '\n');

printf("Invalid input! Please enter a number.\n");

new\_price = -1;

} else if(new\_price <= 0) {

printf("Price must be positive!\n");

}

} while(new\_price <= 0);

inventory[i].price = new\_price;

printf("Medicine updated successfully!\n");

return;

}

}

printf("Medicine with ID %d not found!\n", id);

}

void search\_medicine() {

int choice;

printf("Search by:\n1. Name\n2. Category\n3. ID\nChoice: ");

scanf("%d", &choice);

while(getchar() != '\n');

char search\_term[50];

int found = 0;

switch(choice) {

case 1:

printf("Enter medicine name: ");

fgets(search\_term, 50, stdin);

search\_term[strcspn(search\_term, "\n")] = '\0';

for(int i = 0; i < medicine\_count; i++) {

if(strcasecmp(inventory[i].name, search\_term) == 0) {

printf("\nID: %d\nName: %s\nCategory: %s\nQuantity: %d\nPrice: %.2f\nExpiry: %s\n",

inventory[i].id, inventory[i].name, inventory[i].category,

inventory[i].quantity, inventory[i].price, inventory[i].exp\_date);

found = 1;

}

}

break;

case 2:

printf("Enter category: ");

fgets(search\_term, 30, stdin);

search\_term[strcspn(search\_term, "\n")] = '\0';

for(int i = 0; i < medicine\_count; i++) {

if(strcasecmp(inventory[i].category, search\_term) == 0) {

printf("\nID: %d\nName: %s\nQuantity: %d\nPrice: %.2f\n",

inventory[i].id, inventory[i].name,

inventory[i].quantity, inventory[i].price);

found = 1;

}

}

break;

case 3:

printf("Enter ID: ");

int search\_id;

scanf("%d", &search\_id);

while(getchar() != '\n');

for(int i = 0; i < medicine\_count; i++) {

if(inventory[i].id == search\_id) {

printf("\nName: %s\nCategory: %s\nQuantity: %d\nPrice: %.2f\nExpiry: %s\n",

inventory[i].name, inventory[i].category,

inventory[i].quantity, inventory[i].price,

inventory[i].exp\_date);

found = 1;

break;

}

}

break;

default:

printf("Invalid choice!\n");

return;

}

if(!found) printf("\nNo matching medicines found!\n");

}

void display\_inventory() {

if(medicine\_count == 0) {

printf("Inventory is empty!\n");

return;

}

printf("\n%-5s %-20s %-15s %-10s %-10s %-12s\n",

"ID", "Name", "Category", "Quantity", "Price", "Expiry");

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

for(int i = 0; i < medicine\_count; i++) {

printf("%-5d %-20s %-15s %-10d $%-9.2f %-12s\n",

inventory[i].id,

inventory[i].name,

inventory[i].category,

inventory[i].quantity,

inventory[i].price,

inventory[i].exp\_date);

}

}

void delete\_medicine() {

int id;

printf("Enter Medicine ID to delete: ");

scanf("%d", &id);

while(getchar() != '\n');

for(int i = 0; i < medicine\_count; i++) {

if(inventory[i].id == id) {

for(int j = i; j < medicine\_count - 1; j++) {

inventory[j] = inventory[j + 1];

}

medicine\_count--;

printf("Medicine deleted successfully!\n");

return;

}

}

printf("Medicine with ID %d not found!\n", id);

}

void check\_stock\_alerts() {

int alert\_count = 0;

printf("\nStock Alerts:\n");

for(int i = 0; i < medicine\_count; i++) {

if(inventory[i].quantity < LOW\_STOCK\_THRESHOLD) {

printf(" [LOW STOCK] %s (ID: %d) - %d remaining\n",

inventory[i].name, inventory[i].id, inventory[i].quantity);

alert\_count++;

}

if(is\_expired(inventory[i].exp\_date)) {

printf(" [EXPIRED] %s (ID: %d) - Expired on %s\n",

inventory[i].name, inventory[i].id, inventory[i].exp\_date);

alert\_count++;

}

}

if(alert\_count == 0) printf("No stock alerts!\n");

}

void generate\_report() {

float total\_value = 0;

int expired\_count = 0;

int low\_stock\_count = 0;

printf("\nInventory Valuation Report:\n");

printf("%-20s %-10s %-10s %-12s %-10s\n",

"Medicine", "Quantity", "Unit Price", "Total Value", "Status");

for(int i = 0; i < medicine\_count; i++) {

float value = inventory[i].quantity \* inventory[i].price;

total\_value += value;

char status[20] = "OK";

if(is\_expired(inventory[i].exp\_date)) {

strcpy(status, "EXPIRED");

expired\_count++;

} else if(inventory[i].quantity < LOW\_STOCK\_THRESHOLD) {

strcpy(status, "LOW STOCK");

low\_stock\_count++;

}

printf("%-20s %-10d $%-9.2f $%-10.2f %-10s\n",

inventory[i].name,

inventory[i].quantity,

inventory[i].price,

value,

status);

}

printf("\nSummary:\n");

printf("Total Inventory Value: $%.2f\n", total\_value);

printf("Expired Items: %d\n", expired\_count);

printf("Low Stock Items: %d\n", low\_stock\_count);

}

void medicalStoreSystem() {

load\_inventory();

int choice;

do {

printf("\n=== Medical Store Management System ===\n");

printf("1. Add Medicine\n");

printf("2. Update Medicine\n");

printf("3. Search Medicine\n");

printf("4. Display Inventory\n");

printf("5. Delete Medicine\n");

printf("6. Stock Alerts\n");

printf("7. Generate Report\n");

printf("8. Back to Main Menu\n");

printf("Enter your choice: ");

if(scanf("%d", &choice) != 1) {

while(getchar() != '\n');

printf("Invalid input! Please enter a number.\n");

continue;

}

while(getchar() != '\n');

switch(choice) {

case 1: add\_medicine(); break;

case 2: update\_medicine(); break;

case 3: search\_medicine(); break;

case 4: display\_inventory(); break;

case 5: delete\_medicine(); break;

case 6: check\_stock\_alerts(); break;

case 7: generate\_report(); break;

case 8: save\_inventory(); printf("Returning to main menu...\n"); break;

default: printf("Invalid choice! Please try again.\n");

}

} while(choice != 8);

}

// 6. Billing/Payment System Module

#define MAX\_BILLS 100

#define MAX\_NAME\_LEN 50

#define MAX\_ADDRESS\_LEN 100

#define MAX\_PHONE\_LEN 15

#define MAX\_GENDER\_LEN 10

typedef struct {

int billId;

int patientId;

char patientName[MAX\_NAME\_LEN];

char patientAddress[MAX\_ADDRESS\_LEN];

int patientAge;

char patientPhone[MAX\_PHONE\_LEN];

char patientGender[MAX\_GENDER\_LEN];

float consultationFee;

float treatmentCharges;

float testCharges;

float medicineCharges;

float discount;

float tax;

float totalAmount;

float paidAmount;

float dueAmount;

char paymentMethod[20];

} Bill;

Bill bills[MAX\_BILLS];

int billCount = 0;

void generateBill() {

if (billCount >= MAX\_BILLS) {

printf("Billing storage full!\n");

return;

}

Bill newBill;

newBill.billId = billCount + 1;

printf("Enter Patient ID: ");

scanf("%d", &newBill.patientId);

getchar();

printf("Enter Patient Name: ");

fgets(newBill.patientName, MAX\_NAME\_LEN, stdin);

newBill.patientName[strcspn(newBill.patientName, "\n")] = 0;

printf("Enter Patient Address: ");

fgets(newBill.patientAddress, MAX\_ADDRESS\_LEN, stdin);

newBill.patientAddress[strcspn(newBill.patientAddress, "\n")] = 0;

printf("Enter Patient Age: ");

scanf("%d", &newBill.patientAge);

getchar();

printf("Enter Patient Phone Number: ");

fgets(newBill.patientPhone, MAX\_PHONE\_LEN, stdin);

newBill.patientPhone[strcspn(newBill.patientPhone, "\n")] = 0;

printf("Enter Patient Gender: ");

fgets(newBill.patientGender, MAX\_GENDER\_LEN, stdin);

newBill.patientGender[strcspn(newBill.patientGender, "\n")] = 0;

printf("Enter Consultation Fee: ");

scanf("%f", &newBill.consultationFee);

printf("Enter Treatment Charges: ");

scanf("%f", &newBill.treatmentCharges);

printf("Enter Test Charges: ");

scanf("%f", &newBill.testCharges);

printf("Enter Medicine Charges: ");

scanf("%f", &newBill.medicineCharges);

printf("Enter Discount Amount: ");

scanf("%f", &newBill.discount);

printf("Enter Tax Percentage: ");

scanf("%f", &newBill.tax);

float taxAmount = (newBill.consultationFee + newBill.treatmentCharges +

newBill.testCharges + newBill.medicineCharges - newBill.discount) \*

(newBill.tax / 100);

newBill.totalAmount = newBill.consultationFee + newBill.treatmentCharges +

newBill.testCharges + newBill.medicineCharges +

taxAmount - newBill.discount;

newBill.paidAmount = 0;

newBill.dueAmount = newBill.totalAmount;

strcpy(newBill.paymentMethod, "Pending");

bills[billCount++] = newBill;

printf("\nBill generated successfully!\n");

}

void viewBills() {

if (billCount == 0) {

printf("No bills found!\n");

return;

}

printf("\n--- Billing Records ---\n");

for (int i = 0; i < billCount; i++) {

printf("\nBill ID: %d\n", bills[i].billId);

printf("Patient ID: %d\n", bills[i].patientId);

printf("Total Amount: %.2f\n", bills[i].totalAmount);

printf("Paid Amount: %.2f\n", bills[i].paidAmount);

printf("Due Amount: %.2f\n", bills[i].dueAmount);

printf("Payment Method: %s\n", bills[i].paymentMethod);

}

}

void processPayment() {

int billId, found = -1;

float payment;

char method[20];

printf("Enter Bill ID for payment: ");

scanf("%d", &billId);

for (int i = 0; i < billCount; i++) {

if (bills[i].billId == billId) {

found = i;

break;

}

}

if (found == -1) {

printf("Bill not found!\n");

return;

}

printf("Enter Payment Amount: ");

scanf("%f", &payment);

if (payment > bills[found].dueAmount) {

printf("Error: Payment exceeds due amount!\n");

return;

}

printf("Enter Payment Method (Cash/Card/Insurance): ");

scanf("%s", method);

bills[found].paidAmount += payment;

bills[found].dueAmount -= payment;

strcpy(bills[found].paymentMethod, method);

printf("Payment processed successfully!\n");

}

void saveBillsToFile() {

FILE \*file = fopen("bills.txt", "w");

if (!file) {

printf("Error saving bills!\n");

return;

}

for (int i = 0; i < billCount; i++) {

fprintf(file, "%d %d %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %s %s %s %d %s %s\n",

bills[i].billId, bills[i].patientId,

bills[i].consultationFee, bills[i].treatmentCharges,

bills[i].testCharges, bills[i].medicineCharges,

bills[i].discount, bills[i].tax,

bills[i].totalAmount, bills[i].paidAmount,

bills[i].paymentMethod, bills[i].patientName,

bills[i].patientAddress, bills[i].patientAge,

bills[i].patientPhone, bills[i].patientGender);

}

fclose(file);

printf("Bills saved successfully!\n");

}

void loadBillsFromFile() {

FILE \*file = fopen("bills.txt", "r");

if (!file) {

printf("No previous billing records found!\n");

return;

}

billCount = 0;

while (fscanf(file, "%d %d %f %f %f %f %f %f %f %f %s %s %s %d %s %s",

&bills[billCount].billId, &bills[billCount].patientId,

&bills[billCount].consultationFee, &bills[billCount].treatmentCharges,

&bills[billCount].testCharges, &bills[billCount].medicineCharges,

&bills[billCount].discount, &bills[billCount].tax,

&bills[billCount].totalAmount, &bills[billCount].paidAmount,

bills[billCount].paymentMethod, bills[billCount].patientName,

bills[billCount].patientAddress, &bills[billCount].patientAge,

bills[billCount].patientPhone, bills[billCount].patientGender) != EOF) {

bills[billCount].dueAmount = bills[billCount].totalAmount - bills[billCount].paidAmount;

billCount++;

}

fclose(file);

printf("Bills loaded successfully!\n");

}

void printInvoice(Bill bill) {

printf("\n--- Invoice ---\n");

printf("Bill ID: %d\n", bill.billId);

printf("Patient ID: %d\n", bill.patientId);

printf("Patient Name: %s\n", bill.patientName);

printf("Patient Address: %s\n", bill.patientAddress);

printf("Patient Age: %d\n", bill.patientAge);

printf("Patient Phone: %s\n", bill.patientPhone);

printf("Patient Gender: %s\n", bill.patientGender);

printf("Consultation Fee: %.2f\n", bill.consultationFee);

printf("Treatment Charges: %.2f\n", bill.treatmentCharges);

printf("Test Charges: %.2f\n", bill.testCharges);

printf("Medicine Charges: %.2f\n", bill.medicineCharges);

printf("Discount: %.2f\n", bill.discount);

printf("Tax: %.2f%%\n", bill.tax);

printf("Total Amount: %.2f\n", bill.totalAmount);

printf("Paid Amount: %.2f\n", bill.paidAmount);

printf("Due Amount: %.2f\n", bill.dueAmount);

printf("Payment Method: %s\n", bill.paymentMethod);

}

void generateTotalRevenueReport() {

float totalRevenue = 0;

for (int i = 0; i < billCount; i++) {

totalRevenue += bills[i].paidAmount;

}

printf("\n--- Total Revenue Report ---\n");

printf("Total Revenue: %.2f\n", totalRevenue);

}

void searchBillByPatientId() {

int patientId, found = 0;

printf("Enter Patient ID to search: ");

scanf("%d", &patientId);

for (int i = 0; i < billCount; i++) {

if (bills[i].patientId == patientId) {

printInvoice(bills[i]);

found = 1;

break;

}

}

if (!found) {

printf("No bill found for Patient ID %d.\n", patientId);

}

}

void searchBillByPatientName() {

char patientName[MAX\_NAME\_LEN];

int found = 0;

printf("Enter Patient Name to search: ");

getchar();

fgets(patientName, MAX\_NAME\_LEN, stdin);

patientName[strcspn(patientName, "\n")] = 0;

for (int i = 0; i < billCount; i++) {

if (strcasecmp(bills[i].patientName, patientName) == 0) {

printInvoice(bills[i]);

found = 1;

}

}

if (!found) {

printf("No bill found for Patient Name: %s.\n", patientName);

}

}

void billingSystem() {

loadBillsFromFile();

int choice;

while (1) {

printf("\n--- Billing System ---\n");

printf("1. Generate Bill\n");

printf("2. View Bills\n");

printf("3. Process Payment\n");

printf("4. Save Bills to File\n");

printf("5. Generate Total Revenue Report\n");

printf("6. Search Bill by Patient ID\n");

printf("7. Search Bill by Patient Name\n");

printf("8. Back to Main Menu\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

generateBill();

break;

case 2:

viewBills();

break;

case 3:

processPayment();

break;

case 4:

saveBillsToFile();

break;

case 5:

generateTotalRevenueReport();

break;

case 6:

searchBillByPatientId();

break;

case 7:

searchBillByPatientName();

break;

case 8:

saveBillsToFile();

printf("Returning to main menu...\n");

return;

default:

printf("Invalid choice! Please try again.\n");

}

}

}

// Main function

int main() {

int choice;

while(1) {

displayMainMenu();

scanf("%d", &choice);

switch(choice) {

case 1:

userRegistrationSystem();

break;

case 2:

doctorAppointmentSystem();

break;

case 3:

treatmentManagementSystem();

break;

case 4:

bloodBankSystem();

break;

case 5:

medicalStoreSystem();

break;

case 6:

billingSystem();

break;

case 7:

printf("Exiting Healthcare Management System...\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}