Hospital Managment System Code

```
#include <iostream>
#include <string>
using namespace std;

// Structure to store staff details
struct Staff
{
   int id;
   string name;
   string role;
};

// Functions
void addStaff(Staff staffList[], int &count)
{
   cout << "Enter Staff ID: ";
   cin >> staffList[count].id;
   cin.ignore(); // To clear newline from input buffer
   cout << "Enter Staff Name: ";
   getline(cin, staffList[count].name);
   cout << "Enter Staff Role: ";
   getline(cin, staffList[count].role);
   count++;
```

```
    cout << "Staff added successfully.\n";
}

void showStaff(Staff staffList[], int count)
{
    if (count == 0)
    {
        cout << "No staff available.\n";
    }
    else
    {
        for (int i = 0; i < count; i++)
        {
            cout << "ID: " << staffList[i].id
                 << " Name: " << staffList[i].name
                 << " Role: " << staffList[i].role << endl;
        }
    }
}

void searchStaff(Staff staffList[], int count)
{
    int searchId;
    cout << "Enter Staff ID to search: ";
    cin >> searchId;
```

```
    bool found = false;
    for (int i = 0; i < count; i++)
    {
        if (staffList[i].id == searchId)
        {
            cout << "Staff Found - ID: " << staffList[i].id
                 << " Name: " << staffList[i].name
                 << " Role: " << staffList[i].role << endl;
            found = true;
            break;
        }
    }
    if (!found)
    {
        cout << "Staff with ID " << searchId << " not found.\n";
    }
}

void displayMenuOfStaff()
{

    Staff staffList[50];
    int count = 0, choice;

    do
    {
```

```
// Menu
cout << "\nHospital Staff Management\n";
cout << "1. Add Staff\n";
cout << "2. Show All Staff\n";
cout << "3. Search Staff\n";
cout << "4. Exit\n";
cout << "Enter your choice: ";
cin >> choice;

// Switch case for menu options
switch (choice)
{
case 1:
  addStaff(staffList, count);
  break;
case 2:
  showStaff(staffList, count);
  break;
case 3:
  searchStaff(staffList, count);
  break;
case 4:
  cout << "Exiting program. Goodbye!\n";
  break;
default:
  cout << "Invalid choice. Try again.\n";
```

```
    }
  } while (choice != 4);
}

// Maximum number of patients and appointments (for simplicity)
const int MAX_PATIENTS = 100;
const int MAX_APPOINTMENTS = 10;

// Structure to hold appointment details
struct Appointment
{
  string doctorName;
  string appointmentDate;
  string appointmentTime;
};

// Structure to hold patient details
struct Patient
{
  int id; // Unique Patient ID
  string name;
  int age;
  string gender;
  string contact;
  string medicalHistory;
```

```
  Appointment appointments[MAX_APPOINTMENTS]; // List of appointments for this patient
  int appointmentCount;                // Count of appointments for this patient
};

// Array of patients (up to MAX_PATIENTS)
Patient patients[MAX_PATIENTS];
int currentPatientCount = 0; // Track the number of patients

// Function to register a new patient
void registerPatient()
{
  if (currentPatientCount >= MAX_PATIENTS)
  {
    cout << "Patient list is full." << endl;
    return;
  }

  Patient newPatient;

  cout << "Enter unique Patient ID: ";
  cin >> newPatient.id;
  cin.ignore(); // Clear input buffer
  for (int i = 0; i < currentPatientCount; i++)
  {
    if (patients[i].id == newPatient.id)
```

```
        {
            cout << "Patient ID already exists. Please enter a unique ID.\n";
            return;
        }
    }

    cout << "Enter patient's name: ";
    getline(cin, newPatient.name);
    cout << "Enter patient's age: ";
    cin >> newPatient.age;
    cout << "Enter patient's gender: ";
    cin >> newPatient.gender;
    cout << "Enter patient's contact info: ";
    cin >> newPatient.contact;
    cin.ignore(); // Clear input buffer
    cout << "Enter patient's medical history: ";
    getline(cin, newPatient.medicalHistory);

    newPatient.appointmentCount = 0; // No appointments yet

    patients[currentPatientCount] = newPatient;
    currentPatientCount++;

    cout << "Patient registered successfully.\n";
}
```

```
// Function to schedule an appointment for a patient
void scheduleAppointment()
{
   if (currentPatientCount == 0)
   {
      cout << "No patients registered yet.\n";
      return;
   }

   int patientId;
   cout << "Enter Patient ID: ";
   cin >> patientId;

   int index = -1;
   for (int i = 0; i < currentPatientCount; i++)
   {
      if (patients[i].id == patientId)
      {
         index = i;
         break;
      }
   }

   if (index == -1)
   {
      cout << "Patient ID not found.\n";
```

```
    return;
  }

  if (patients[index].appointmentCount >= MAX_APPOINTMENTS)
  {
    cout << "This patient has reached the maximum number of appointments.\n";
    return;
  }

  Appointment newAppointment;

  cout << "Enter doctor's name: ";
  cin >> newAppointment.doctorName;
  cout << "Enter appointment date (DD/MM/YYYY): ";
  cin >> newAppointment.appointmentDate;
  cout << "Enter appointment time (HH:MM): ";
  cin >> newAppointment.appointmentTime;

  // Add the new appointment to the patient's list
  patients[index].appointments[patients[index].appointmentCount++] =
newAppointment;

  cout << "Appointment scheduled successfully.\n";
}

// Function to display medical history and treatment records for a patient
```

```
void viewMedicalHistory()
{
   if (currentPatientCount == 0)
   {
      cout << "No patients registered yet.\n";
      return;
   }

   int patientId;
   cout << "Enter Patient ID: ";
   cin >> patientId;

   int index = -1;
   for (int i = 0; i < currentPatientCount; i++)
   {
      if (patients[i].id == patientId)
      {
         index = i;
         break;
      }
   }

   if (index == -1)
   {
      cout << "Patient ID not found.\n";
      return;
```

```
    }

    Patient patient = patients[index];

    cout << "\nPatient ID: " << patient.id << endl;
    cout << "Name: " << patient.name << endl;
    cout << "Age: " << patient.age << endl;
    cout << "Gender: " << patient.gender << endl;
    cout << "Contact: " << patient.contact << endl;
    cout << "Medical History: " << patient.medicalHistory << endl;

    // Display patient's appointments
    cout << "\nAppointments:\n";
    for (int i = 0; i < patient.appointmentCount; ++i)
    {
        cout << "Doctor: " << patient.appointments[i].doctorName << endl;
        cout << "Date: " << patient.appointments[i].appointmentDate << endl;
        cout << "Time: " << patient.appointments[i].appointmentTime << endl;
        cout << "--------------------------\n";
    }
}

// Function to search for a patient by name
void searchPatientByName()
{
    string name;
```

```
    cout << "Enter the name of the patient you want to search: ";
    cin.ignore();
    getline(cin, name);

    bool found = false;
    for (int i = 0; i < currentPatientCount; i++)
    {
        if (patients[i].name == name)
        {
            cout << "\nPatient ID: " << patients[i].id << endl;
            cout << "Name: " << patients[i].name << endl;
            cout << "Age: " << patients[i].age << endl;
            cout << "Gender: " << patients[i].gender << endl;
            cout << "Contact: " << patients[i].contact << endl;
            cout << "Medical History: " << patients[i].medicalHistory << endl;
            found = true;
        }
    }

    if (!found)
    {
        cout << "Patient not found.\n";
    }
}

// Main menu function
```

```
void displayMenuOfPatients()
{
  int choice;
  do
  {
    cout << "\nHospital Patient Management\n";
    cout << "1. Register new patient\n";
    cout << "2. Schedule appointment\n";
    cout << "3. View medical history and treatment records\n";
    cout << "4. Search patient by name\n";
    cout << "5. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice)
    {
    case 1:
      registerPatient();
      break;
    case 2:
      scheduleAppointment();
      break;
    case 3:
      viewMedicalHistory();
      break;
    case 4:
```

```
            searchPatientByName();
            break;
        case 5:
            cout << "Exiting the program.\n";
            break;
        default:
            cout << "Invalid choice, please try again.\n";
        }
    } while (choice != 5);
}

const int max_items = 100;    // Maximum number of items
string item_names[max_items];   // Array to store item names
int item_quantities[max_items]; // Array to store item quantities
float item_prices[max_items];   // Array to store item prices
int item_count = 0;          // Current number of items

// Function to add an item
void add_item()
{
    if (item_count >= max_items)
    {
        cout << "Inventory is full. Cannot add more items.\n";
        return;
    }
    cout << "Enter item name: ";
```

```
    cin >> item_names[item_count];
    cout << "Enter quantity: ";
    cin >> item_quantities[item_count];
    cout << "Enter price: ";
    cin >> item_prices[item_count];
    item_count++;
    cout << "Item added successfully.\n";
}

// Function to view all items
void view_items()
{
    if (item_count == 0)
    {
        cout << "No items to display.\n";
        return;
    }
    cout << "Inventory Items:\n";
    for (int i = 0; i < item_count; i++)
    {
        cout << i + 1 << ". Name: " << item_names[i]
             << ", Quantity: " << item_quantities[i]
             << ", Price: " << item_prices[i] << endl;
    }
}
```

```
// Function to update an item
void update_item()
{
   if (item_count == 0)
   {
      cout << "No items to update.\n";
      return;
   }
   string search_name;
   cout << "Enter the name of the item to update: ";
   cin >> search_name;

   // Search for the item
   for (int i = 0; i < item_count; i++)
   {
      if (item_names[i] == search_name)
      {
         int sub_choice;
         cout << "1. Update Quantity\n";
         cout << "2. Update Price\n";
         cout << "Enter your choice: ";
         cin >> sub_choice;

         if (sub_choice == 1)
         {
            cout << "Enter new quantity: ";
```

```
            cin >> item_quantities[i];
            cout << "Quantity updated successfully.\n";
        }
        else if (sub_choice == 2)
        {
            cout << "Enter new price: ";
            cin >> item_prices[i];
            cout << "Price updated successfully.\n";
        }
        else
        {
            cout << "Invalid choice.\n";
        }
        return;
      }
   }
   cout << "Item not found.\n";
}

void menu()
{

   int choice;

   do
   {
```

```
// Display menu
cout << "\nInventory Menu:\n";
cout << "1. Add Item\n";
cout << "2. View Items\n";
cout << "3. Update Item\n";
cout << "4. Exit\n";
cout << "Enter your choice: ";
cin >> choice;

// Handle menu choices using functions
switch (choice)
{
case 1:
  add_item();
  break;
case 2:
  view_items();
  break;
case 3:
  update_item();
  break;
case 4:
  cout << "Exiting program.\n";
  break;
default:
  cout << "Invalid choice. Please try again.\n";
```

```
    }
  } while (choice != 4);
}

const int NUM_WARDS = 3, NUM_BEDS = 6;

// Function to display the ward status
void displayStatus(string wards[NUM_WARDS][NUM_BEDS])
{
  cout << "\nWard Status:\n";
  for (int i = 0; i < NUM_WARDS; i++)
  {
    cout << "Ward " << i + 1 << ": ";
    for (int j = 0; j < NUM_BEDS; j++)
    {
      if (wards[i][j].empty())
      {
        cout << "[Empty] ";
      }
      else
      {
        cout << "[" << wards[i][j] << "] ";
      }
    }
    cout << endl;
  }
```

```
}

// Function to manage bed allocation and discharge
void manageBed(string wards[NUM_WARDS][NUM_BEDS], bool allocate)
{
  int ward, bed;
  cout << "Enter ward (1-" << NUM_WARDS << ") and bed (1-" << NUM_BEDS << "): ";
  cin >> ward >> bed;

  if (ward < 1 || ward > NUM_WARDS || bed < 1 || bed > NUM_BEDS)
  {
    cout << "Invalid input. Try again.\n";
  }

  string &current = wards[ward - 1][bed - 1];
  if (allocate)
  {
    if (!current.empty())
    {
      cout << "Bed occupied. Try another.\n";
    }
    else
    {
      cout << "Enter patient name: ";
      cin.ignore();
      getline(cin, current);
```

```
            cout << "Bed allocated to " << current << ".\n";
        }
    }
    else
    {
        if (current.empty())
        {
            cout << "Bed already empty.\n";
        }
        else
        {
            cout << "Discharging " << current << "...\n";
        }
    }
}

void displayMenuOfWards()
{
    string wards[NUM_WARDS][NUM_BEDS];
    int choice;

    do
    {
        cout << "\nHospital Ward and Bed Management System\n";
        cout << "1. Display Ward Status\n";
        cout << "2. Allocate Bed\n";
```

```
        cout << "3. Discharge Bed\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
          displayStatus(wards);
          break;
        case 2:
          manageBed(wards, true);
          break;
        case 3:
          manageBed(wards, false);
          break;
        case 4:
          cout << "Exiting program. Goodbye!\n";
          break;
        default:
          cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);
}

int main()
```

```
{

  int choice;
  do
  {
    cout << "\n -----HOSPITAL MANAGEMENT SYSTEM----\n";
    cout << "1. Staff Management\n";
    cout << "2. Patient Management\n";
    cout << "3. Inventory Management\n";
    cout << "4. Wards and beds Management\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice)
    {
    case 1:
      displayMenuOfStaff();
      break;
    case 2:
      displayMenuOfPatients();
      break;
    case 3:
      menu();
      break;
    case 4:
      displayMenuOfWards();
```

```
            break;
        default:
            cout << "Invalid choice. Try again\n";
        }
    } while (choice != 4);

    return 0;
}
```