**Problem 1: Patient Information Management System**

**Description:** Create a menu-driven program to manage patient information, including basic details, medical history, and current medications.

**Menu Options:**

1. Add New Patient
2. View Patient Details
3. Update Patient Information
4. Delete Patient Record
5. List All Patients
6. Exit

**Requirements:**

1. Use variables to store patient details.
2. Utilize static and const for immutable data such as hospital name.
3. Implement switch case for menu selection.
4. Employ loops for iterative tasks like listing patients.
5. Use pointers for dynamic memory allocation.
6. Implement functions for CRUD operations.
7. Utilize arrays for storing multiple patient records.
8. Use structures for organizing patient data.
9. Apply nested structures for detailed medical history.
10. Use unions for optional data fields.
11. Employ nested unions for multi-type data entries.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #include <string.h>
15. #define MAX\_PATIENTS 100
16. // Immutable data
17. static const char hospitalName[] = "City General Hospital";
18. // Union for optional medications data
19. typedef union {
20. char medicationName[50];
21. int dosage;
22. } MedicationData;
23. // Nested structure
24. typedef struct {
25. char disease[50];
26. char treatment[50];
27. MedicationData medication;
28. int hasMedication;
29. } MedicalHistory;
30. // Structure
31. typedef struct {
32. int id;
33. char name[50];
34. int age;
35. char gender[10];
36. MedicalHistory history;
37. } Patient;
38. // Array to store the patients and a count of total patients
39. Patient patients[MAX\_PATIENTS];
40. int totalPatients = 0;
41. // Function prototypes
42. void addPatient();
43. void viewPatient();
44. void updatePatient();
45. void deletePatient();
46. void listPatients();
47. int findPatientIndex(int id);
48. int main() {
49. int choice;
50. while (1) {
51. printf("\n--- Patient Information Management System ---\n");
52. printf("Hospital: %s\n", hospitalName);
53. printf("1. Add New Patient\n");
54. printf("2. View Patient Details\n");
55. printf("3. Update Patient Information\n");
56. printf("4. Delete Patient Record\n");
57. printf("5. List All Patients\n");
58. printf("6. Exit\n");
59. printf("Enter your choice: ");
60. scanf("%d", &choice);
61. switch (choice) {
62. case 1:
63. addPatient();
64. break;
65. case 2:
66. viewPatient();
67. break;
68. case 3:
69. updatePatient();
70. break;
71. case 4:
72. deletePatient();
73. break;
74. case 5:
75. listPatients();
76. break;
77. case 6:
78. printf("Exiting...\n");
79. return 0;
80. default:
81. printf("Invalid choice. Please try again.\n");
82. }
83. }
84. }
85. // Function to find patient by ID and return index
86. int findPatientIndex(int id) {
87. for (int i = 0; i < totalPatients; i++) {
88. if (patients[i].id == id) {
89. return i;
90. }
91. }
92. return -1; // Patient not found
93. }
94. // Function to add a new patient
95. void addPatient() {
96. if (totalPatients >= MAX\_PATIENTS) {
97. printf("Cannot add more patients. Maximum limit reached.\n");
98. return;
99. }
100. Patient \*p = &patients[totalPatients];
101. printf("Enter patient ID: ");
102. scanf("%d", &p->id);
103. if (findPatientIndex(p->id) != -1) {
104. printf("Error: Patient ID already exists. Try again.\n");
105. return;
106. }
107. printf("Enter name: ");
108. scanf("%s", p->name);
109. printf("Enter age: ");
110. scanf("%d", &p->age);
111. printf("Enter gender: ");
112. scanf("%s", p->gender);
113. // Adding medical history
114. printf("Enter disease: ");
115. scanf("%s", p->history.disease);
116. printf("Enter treatment: ");
117. scanf("%s", p->history.treatment);
118. printf("Is the patient on medication? (1 for Yes, 0 for No): ");
119. scanf("%d", &p->history.hasMedication);
120. if (p->history.hasMedication == 1) {
121. printf("Enter medication name: ");
122. scanf("%s", p->history.medication.medicationName);
123. printf("Enter dosage (in mg): ");
124. scanf("%d", &p->history.medication.dosage);
125. }
126. totalPatients++;
127. printf("Patient added successfully!\n");
128. }
129. // Function to view a patient's details
130. void viewPatient() {
131. int id;
132. printf("Enter Patient ID: ");
133. scanf("%d", &id);
134. int index = findPatientIndex(id);
135. if (index != -1) {
136. Patient \*p = &patients[index];
137. printf("ID: %d, Name: %s, Age: %d, Gender: %s\n",
138. p->id, p->name, p->age, p->gender);
139. printf("Disease: %s\n", p->history.disease);
140. printf("Treatment: %s\n", p->history.treatment);
141. // Display medication details if available
142. if (p->history.hasMedication == 1) {
143. printf("Medication: %s\n", p->history.medication.medicationName);
144. printf("Dosage: %d mg\n", p->history.medication.dosage);
145. } else {
146. printf("No current medication.\n");
147. }
148. } else {
149. printf("Patient not found.\n");
150. }
151. }
152. // Function to update a patient's information
153. void updatePatient() {
154. int id;
155. printf("Enter Patient ID: ");
156. scanf("%d", &id);
157. int index = findPatientIndex(id);
158. if (index != -1) {
159. Patient \*p = &patients[index];
160. printf("Enter new name: ");
161. scanf("%s", p->name);
162. printf("Enter new age: ");
163. scanf("%d", &p->age);
164. printf("Enter new gender: ");
165. scanf("%s", p->gender);
166. // Update medical history
167. printf("Enter new disease: ");
168. scanf("%s", p->history.disease);
169. printf("Enter new treatment: ");
170. scanf("%s", p->history.treatment);
171. printf("Is the patient on medication? (1 for Yes, 0 for No): ");
172. scanf("%d", &p->history.hasMedication);
173. if (p->history.hasMedication == 1) {
174. printf("Enter new medication name: ");
175. scanf("%s", p->history.medication.medicationName);
176. printf("Enter new dosage (in mg): ");
177. scanf("%d", &p->history.medication.dosage);
178. }
179. printf("Patient information updated successfully!\n");
180. } else {
181. printf("Patient not found.\n");
182. }
183. }
184. // Function to delete a patient record
185. void deletePatient() {
186. int id;
187. printf("Enter Patient ID: ");
188. scanf("%d", &id);
189. int index = findPatientIndex(id);
190. if (index != -1) {
191. // Shift the array to delete the patient
192. for (int i = index; i < totalPatients - 1; i++) {
193. patients[i] = patients[i + 1];
194. }
195. totalPatients--;
196. printf("Patient record deleted successfully!\n");
197. } else {
198. printf("Patient not found.\n");
199. }
200. }
201. // Function to list all patients
202. void listPatients() {
203. if (totalPatients == 0) {
204. printf("No patients to display.\n");
205. return;
206. }
207. for (int i = 0; i < totalPatients; i++) {
208. printf("ID: %d, Name: %s, Age: %d, Gender: %s\n",
209. patients[i].id, patients[i].name, patients[i].age, patients[i].gender);
210. printf("Disease: %s\n", patients[i].history.disease);
211. printf("Treatment: %s\n", patients[i].history.treatment);
212. if (patients[i].history.hasMedication == 1) {
213. printf("Medication: %s\n", patients[i].history.medication.medicationName);
214. printf("Dosage: %d mg\n", patients[i].history.medication.dosage);
215. } else {
216. printf("No current medication.\n");
217. }
218. }
219. }

**Problem 2: Hospital Inventory Management**

**Description:** Design a system to manage the inventory of medical supplies.

**Menu Options:**

1. Add Inventory Item
2. View Inventory Item
3. Update Inventory Item
4. Delete Inventory Item
5. List All Inventory Items
6. Exit

**Requirements:**

1. Declare variables for inventory details.
2. Use static and const for fixed supply details.
3. Implement switch case for different operations like adding, deleting, and viewing inventory.
4. Utilize loops for repetitive inventory checks.
5. Use pointers to handle inventory records.
6. Create functions for managing inventory.
7. Use arrays to store inventory items.
8. Define structures for each supply item.
9. Use nested structures for detailed item specifications.
10. Employ unions for variable item attributes.
11. Implement nested unions for complex item data types.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #include <string.h>
15. #define MAX\_ITEMS 100
16. // Structure for detailed item specifications
17. typedef struct {
18. char manufacturer[50];
19. char expiryDate[15];
20. } ItemSpecifications;
21. // Union for variable item attributes
22. typedef union {
23. int quantity;
24. float weight;
25. } ItemAttributes;
26. // Nested union for complex item data types
27. typedef union {
28. char serialNumber[20];
29. char lotNumber[20];
30. } ComplexItemData;
31. // Structure for inventory item
32. typedef struct {
33. int id;
34. char name[50];
35. char category[30];
36. ItemSpecifications specs;
37. ItemAttributes attributes;
38. ComplexItemData complexData;
39. } InventoryItem;
40. // Array to store inventory items
41. InventoryItem \*inventory[MAX\_ITEMS];
42. int itemCount = 0;
43. // Function prototypes
44. void addInventoryItem();
45. void viewInventoryItem();
46. void updateInventoryItem();
47. void deleteInventoryItem();
48. void listAllInventoryItems();
49. int main() {
50. int choice;
51. printf("-------------Hospital Inventory Management--------------\n");
52. do {
53. printf("\nMenu:\n");
54. printf("1. Add Inventory Item\n");
55. printf("2. View Inventory Item\n");
56. printf("3. Update Inventory Item\n");
57. printf("4. Delete Inventory Item\n");
58. printf("5. List All Inventory Items\n");
59. printf("6. Exit\n");
60. printf("Enter your choice: ");
61. scanf("%d", &choice);
62. switch (choice) {
63. case 1:
64. addInventoryItem();
65. break;
66. case 2:
67. viewInventoryItem();
68. break;
69. case 3:
70. updateInventoryItem();
71. break;
72. case 4:
73. deleteInventoryItem();
74. break;
75. case 5:
76. listAllInventoryItems();
77. break;
78. case 6:
79. printf("Exiting the program. Goodbye!\n");
80. break;
81. default:
82. printf("Invalid choice! Please try again.\n");
83. }
84. } while (choice != 6);
85. return 0;
86. }
87. void addInventoryItem() {
88. if (itemCount >= MAX\_ITEMS) {
89. printf("Inventory capacity reached! Cannot add more items.\n");
90. return;
91. }
92. InventoryItem \*newItem = (InventoryItem \*)malloc(sizeof(InventoryItem));
93. printf("Enter Item ID: ");
94. scanf("%d", &newItem->id);
95. printf("Enter Item Name: ");
96. scanf("%s", newItem->name);
97. printf("Enter Item Category: ");
98. scanf("%s", newItem->category);
99. printf("Enter Manufacturer: ");
100. scanf("%s", newItem->specs.manufacturer);
101. printf("Enter Expiry Date: ");
102. scanf("%s", newItem->specs.expiryDate);
103. printf("Enter Quantity: ");
104. scanf("%d", &newItem->attributes.quantity);
105. printf("Enter Serial Number: ");
106. scanf("%s", newItem->complexData.serialNumber);
107. inventory[itemCount++] = newItem;
108. printf("Inventory item added successfully!\n");
109. }
110. void viewInventoryItem() {
111. int id;
112. printf("Enter Item ID to view details: ");
113. scanf("%d", &id);
114. for (int i = 0; i < itemCount; i++) {
115. if (inventory[i]->id == id) {
116. printf("Item ID: %d\n", inventory[i]->id);
117. printf("Name: %s\n", inventory[i]->name);
118. printf("Category: %s\n", inventory[i]->category);
119. printf("Manufacturer: %s\n", inventory[i]->specs.manufacturer);
120. printf("Expiry Date: %s\n", inventory[i]->specs.expiryDate);
121. printf("Quantity: %d\n", inventory[i]->attributes.quantity);
122. printf("Serial Number: %s\n", inventory[i]->complexData.serialNumber);
123. return;
124. }
125. }
126. printf("Item not found!\n");
127. }
128. void updateInventoryItem() {
129. int id;
130. printf("Enter Item ID to update: ");
131. scanf("%d", &id);
132. for (int i = 0; i < itemCount; i++) {
133. if (inventory[i]->id == id) {
134. printf("Enter New Name: ");
135. scanf("%s", inventory[i]->name);
136. printf("Enter New Category: ");
137. scanf("%s", inventory[i]->category);
138. printf("Enter New Quantity: ");
139. scanf("%d", &inventory[i]->attributes.quantity);
140. printf("Enter New Serial Number: ");
141. scanf("%s", inventory[i]->complexData.serialNumber);
142. printf("Inventory item updated successfully!\n");
143. return;
144. }
145. }
146. printf("Item not found!\n");
147. }
148. void deleteInventoryItem() {
149. int id;
150. printf("Enter Item ID to delete: ");
151. scanf("%d", &id);
152. for (int i = 0; i < itemCount; i++) {
153. if (inventory[i]->id == id) {
154. free(inventory[i]);
155. for (int j = i; j < itemCount - 1; j++) {
156. inventory[j] = inventory[j + 1];
157. }
158. itemCount--;
159. printf("Inventory item deleted successfully!\n");
160. return;
161. }
162. }
163. printf("Item not found!\n");
164. }
165. void listAllInventoryItems() {
166. if (itemCount == 0) {
167. printf("No inventory items found!\n");
168. return;
169. }
170. printf("\nList of Inventory Items:\n");
171. for (int i = 0; i < itemCount; i++) {
172. printf("ID: %d, Name: %s, Category: %s, Manufacturer: %s\n",
173. inventory[i]->id, inventory[i]->name, inventory[i]->category, inventory[i]->specs.manufacturer);
174. }
175. }

**Problem 3: Medical Appointment Scheduling System**

**Description:** Develop a system to manage patient appointments.

**Menu Options:**

1. Schedule Appointment
2. View Appointment
3. Update Appointment
4. Cancel Appointment
5. List All Appointments
6. Exit

**Requirements:**

1. Use variables for appointment details.
2. Apply static and const for non-changing data like clinic hours.
3. Implement switch case for appointment operations.
4. Utilize loops for scheduling.
5. Use pointers for dynamic data manipulation.
6. Create functions for appointment handling.
7. Use arrays for storing appointments.
8. Define structures for appointment details.
9. Employ nested structures for detailed doctor and patient information.
10. Utilize unions for optional appointment data.
11. Apply nested unions for complex appointment data.
12. #include <stdio.h>
13. #include <string.h>
14. #define MAX\_APPOINTMENTS 100
15. const char clinicHours[] = "9:00 AM to 5:00 PM";
16. // Nested structures for detailed doctor and patient information
17. struct Doctor {
18. char name[50];
19. char specialization[50];
20. };
21. struct Patient {
22. char name[50];
23. int age;
24. };
25. // Union for optional fields
26. union OptionalData {
27. char followUpDate[15];
28. char notes[100];
29. };
30. // Appointment structure with nested structures and union
31. struct Appointment {
32. int id;
33. struct Patient patient;
34. struct Doctor doctor;
35. char date[15];
36. char time[10];
37. union OptionalData optional;
38. char optionalType[10];
39. };
40. // Array to store appointments
41. struct Appointment appointments[MAX\_APPOINTMENTS];
42. static int totalAppointments = 0;
43. // Function prototypes
44. void scheduleAppointment();
45. void viewAppointment();
46. void updateAppointment();
47. void cancelAppointment();
48. void listAppointments();
49. int main() {
50. int choice;
51. printf("Clinic Hours: %s\n", clinicHours);
52. while (1) {
53. printf("\nMedical Appointment Scheduling System\n");
54. printf("1. Schedule Appointment\n");
55. printf("2. View Appointment\n");
56. printf("3. Update Appointment\n");
57. printf("4. Cancel Appointment\n");
58. printf("5. List All Appointments\n");
59. printf("6. Exit\n");
60. printf("Enter your choice: ");
61. scanf("%d", &choice);
62. switch (choice) {
63. case 1:
64. scheduleAppointment();
65. break;
66. case 2:
67. viewAppointment();
68. break;
69. case 3:
70. updateAppointment();
71. break;
72. case 4:
73. cancelAppointment();
74. break;
75. case 5:
76. listAppointments();
77. break;
78. case 6:
79. printf("Exiting...\n");
80. return 0;
81. default:
82. printf("Invalid choice. Please try again.\n");
83. }
84. }
85. }
86. // Function to schedule a new appointment
87. void scheduleAppointment() {
88. if (totalAppointments >= MAX\_APPOINTMENTS) {
89. printf("Cannot schedule more appointments. Maximum limit reached.\n");
90. return;
91. }
92. struct Appointment \*appointment = &appointments[totalAppointments];
93. appointment->id = ++totalAppointments;
94. printf("Enter patient name: ");
95. scanf(" %[^\n]%\*c", appointment->patient.name);
96. printf("Enter patient age: ");
97. scanf("%d", &appointment->patient.age);
98. printf("Enter doctor name: ");
99. scanf(" %[^\n]%\*c", appointment->doctor.name);
100. printf("Enter doctor specialization: ");
101. scanf(" %[^\n]%\*c", appointment->doctor.specialization);
102. printf("Enter appointment date (dd/mm/yyyy): ");
103. scanf("%s", appointment->date);
104. printf("Enter appointment time (hh:mm AM/PM): ");
105. scanf("%s", appointment->time);
106. printf("Optional: Enter 'followUp' or 'notes': ");
107. scanf("%s", appointment->optionalType);
108. if (strcmp(appointment->optionalType, "followUp") == 0) {
109. printf("Enter follow-up date (dd/mm/yyyy): ");
110. scanf("%s", appointment->optional.followUpDate);
111. } else if (strcmp(appointment->optionalType, "notes") == 0) {
112. printf("Enter notes: ");
113. scanf(" %[^\n]%\*c", appointment->optional.notes);
114. } else {
115. printf("No optional data added.\n");
116. }
117. printf("Appointment scheduled successfully with ID: %d\n", appointment->id);
118. }
119. // Function to view an appointment by ID
120. void viewAppointment() {
121. int id;
122. printf("Enter appointment ID: ");
123. scanf("%d", &id);
124. if (id <= 0 || id > totalAppointments) {
125. printf("Appointment not found.\n");
126. return;
127. }
128. struct Appointment \*appointment = &appointments[id - 1];
129. printf("\nAppointment ID: %d\n", appointment->id);
130. printf("Patient Name: %s\n", appointment->patient);
131. printf("Doctor Name: %s\n", appointment->doctor);
132. printf("Date: %s\n", appointment->date);
133. printf("Time: %s\n", appointment->time);
134. }
135. // Function to update an appointment by ID
136. void updateAppointment() {
137. int id;
138. printf("Enter appointment ID: ");
139. scanf("%d", &id);
140. if (id <= 0 || id > totalAppointments) {
141. printf("Appointment not found.\n");
142. return;
143. }
144. struct Appointment \*appointment = &appointments[id - 1];
145. printf("Enter new patient name: ");
146. scanf(" %[^\n]%\*c", appointment->patient);
147. printf("Enter new doctor name: ");
148. scanf(" %[^\n]%\*c", appointment->doctor);
149. printf("Enter new appointment date (dd/mm/yyyy): ");
150. scanf("%s", appointment->date);
151. printf("Enter new appointment time (hh:mm AM/PM): ");
152. scanf("%s", appointment->time);
153. printf("Appointment updated successfully!\n");
154. }
155. // Function to cancel an appointment by ID
156. void cancelAppointment() {
157. int id;
158. printf("Enter appointment ID: ");
159. scanf("%d", &id);
160. if (id <= 0 || id > totalAppointments) {
161. printf("Appointment not found.\n");
162. return;
163. }
164. for (int i = id - 1; i < totalAppointments - 1; i++) {
165. appointments[i] = appointments[i + 1];
166. }
167. totalAppointments--;
168. printf("Appointment cancelled successfully!\n");
169. }
170. // Function to list all appointments
171. void listAppointments() {
172. if (totalAppointments == 0) {
173. printf("No appointments scheduled.\n");
174. return;
175. }
176. printf("\nList of Appointments:\n");
177. for (int i = 0; i < totalAppointments; i++) {
178. struct Appointment \*appointment = &appointments[i];
179. printf("ID: %d, Patient: %s, Doctor: %s, Date: %s, Time: %s\n",
180. appointment->id, appointment->patient, appointment->doctor,
181. appointment->date, appointment->time);
182. }
183. }

**Problem 4: Patient Billing System**

**Description:** Create a billing system for patients.

**Menu Options:**

1. Generate Bill
2. View Bill
3. Update Bill
4. Delete Bill
5. List All Bills
6. Exit

**Requirements:**

1. Declare variables for billing information.
2. Use static and const for fixed billing rates.
3. Implement switch case for billing operations.
4. Utilize loops for generating bills.
5. Use pointers for bill calculations.
6. Create functions for billing processes.
7. Use arrays for storing billing records.
8. Define structures for billing components.
9. Employ nested structures for detailed billing breakdown.
10. Use unions for variable billing elements.
11. Apply nested unions for complex billing scenarios.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #define MAX\_BILLS 100
15. // Fixed billing rates
16. const float ROOM\_RATE = 500.0;
17. const float TREATMENT\_RATE = 200.0;
18. const float MED\_RATE = 100.0;
19. // Union for variable billing elements (Days or Hours)
20. typedef union {
21. int days;
22. float hours;
23. } BillingDetails;
24. // Nested structure for detailed billing breakdown
25. typedef struct {
26. float roomCharge;
27. float treatmentCharge;
28. float medCharge;
29. } BillingBreakdown;
30. // Bill structure containing union and nested structure
31. typedef struct {
32. int id;
33. char name[50];
34. int age;
35. BillingDetails details;
36. BillingBreakdown breakdown;
37. } Bill;
38. // Array to store bills
39. Bill \*bills[MAX\_BILLS];
40. int billCount = 0;
41. // Function prototypes
42. void generateBill();
43. void viewBill();
44. void listBills();
45. float calculateTotal(Bill \*b);
46. int main() {
47. int choice;
48. do {
49. printf("\nPatient Billing System\n");
50. printf("1. Generate Bill\n");
51. printf("2. View Bill\n");
52. printf("3. List Bills\n");
53. printf("4. Exit\n");
54. printf("Enter your choice: ");
55. scanf("%d", &choice);
56. switch (choice) {
57. case 1:
58. generateBill();
59. break;
60. case 2:
61. viewBill();
62. break;
63. case 3:
64. listBills();
65. break;
66. case 4:
67. printf("Goodbye!\n");
68. break;
69. default:
70. printf("Invalid choice. Try again.\n");
71. }
72. } while (choice != 4);
73. return 0;
74. }
75. // Generate a bill
76. void generateBill() {
77. if (billCount >= MAX\_BILLS) {
78. printf("No more space for bills.\n");
79. return;
80. }
81. Bill \*newBill = (Bill \*)malloc(sizeof(Bill));
82. printf("Enter Bill ID: ");
83. scanf("%d", &newBill->id);
84. printf("Enter Patient Name: ");
85. scanf("%s", newBill->name);
86. printf("Enter Patient Age: ");
87. scanf("%d", &newBill->age);
88. // Input hospital stay or treatment hours
89. printf("Enter Number of Days in Hospital: ");
90. scanf("%d", &newBill->details.days);
91. printf("Enter Treatment Hours: ");
92. scanf("%f", &newBill->details.hours);
93. // Calculate charges using the union fields
94. newBill->breakdown.roomCharge = ROOM\_RATE \* newBill->details.days;
95. newBill->breakdown.treatmentCharge = TREATMENT\_RATE \* newBill->details.hours;
96. newBill->breakdown.medCharge = MED\_RATE;
97. bills[billCount++] = newBill;
98. printf("Bill generated successfully!\n");
99. }
100. // View a specific bill
101. void viewBill() {
102. int id;
103. printf("Enter Bill ID to view: ");
104. scanf("%d", &id);
105. for (int i = 0; i < billCount; i++) {
106. if (bills[i]->id == id) {
107. printf("Bill ID: %d\n", bills[i]->id);
108. printf("Name: %s\n", bills[i]->name);
109. printf("Age: %d\n", bills[i]->age);
110. printf("Room Charge: %.2f\n", bills[i]->breakdown.roomCharge);
111. printf("Treatment Charge: %.2f\n", bills[i]->breakdown.treatmentCharge);
112. printf("Medication Charge: %.2f\n", bills[i]->breakdown.medCharge);
113. printf("Total Bill: %.2f\n", calculateTotal(bills[i]));
114. return;
115. }
116. }
117. printf("Bill not found!\n");
118. }
119. // List all bills
120. void listBills() {
121. if (billCount == 0) {
122. printf("No bills to display.\n");
123. return;
124. }
125. printf("\nList of Bills:\n");
126. for (int i = 0; i < billCount; i++) {
127. printf("Bill ID: %d, Patient: %s, Total: %.2f\n",
128. bills[i]->id, bills[i]->name, calculateTotal(bills[i]));
129. }
130. }
131. // Calculate the total bill
132. float calculateTotal(Bill \*b) {
133. return b->breakdown.roomCharge + b->breakdown.treatmentCharge + b->breakdown.medCharge;
134. }

**Problem 5: Medical Test Result Management**

**Description:** Develop a system to manage and store patient test results

**Menu Options:**

1. Add Test Result
2. View Test Result
3. Update Test Result
4. Delete Test Result
5. List All Test Results
6. Exit

**Requirements:**

1. Declare variables for test results.
2. Use static and const for standard test ranges.
3. Implement switch case for result operations.
4. Utilize loops for result input and output.
5. Use pointers for handling result data.
6. Create functions for result management.
7. Use arrays for storing test results.
8. Define structures for test result details.
9. Employ nested structures for detailed test parameters.
10. Utilize unions for optional test data.
11. Apply nested unions for complex test result data.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #define MAX\_TESTS 100
15. // Static values for test ranges
16. static const float MIN\_TEMP = 97.0;
17. static const float MAX\_TEMP = 99.5;
18. static const int MIN\_BP = 90;
19. static const int MAX\_BP = 120;
20. // Union for holding test data
21. typedef union {
22. float temperature;
23. int bloodPressure;
24. } TestData;
25. // Structure for storing a test result
26. typedef struct {
27. int id;
28. char name[50];
29. int age;
30. TestData data;
31. int testType;
32. } TestResult;
33. // Array to store the test results
34. TestResult tests[MAX\_TESTS];
35. int testCount = 0;
36. // Function prototypes
37. void addTestResult();
38. void viewTestResult();
39. void listAllTestResults();
40. void updateTestResult();
41. void deleteTestResult();
42. int main() {
43. int choice;
44. do {
45. printf("\n--- Medical Test Result Management ---\n");
46. printf("1. Add Test Result\n");
47. printf("2. View Test Result\n");
48. printf("3. Update Test Result\n");
49. printf("4. Delete Test Result\n");
50. printf("5. List All Test Results\n");
51. printf("6. Exit\n");
52. printf("Enter your choice: ");
53. scanf("%d", &choice);
54. switch (choice) {
55. case 1:
56. addTestResult();
57. break;
58. case 2:
59. viewTestResult();
60. break;
61. case 3:
62. updateTestResult();
63. break;
64. case 4:
65. deleteTestResult();
66. break;
67. case 5:
68. listAllTestResults();
69. break;
70. case 6:
71. printf("Exiting the program.\n");
72. break;
73. default:
74. printf("Invalid choice! Try again.\n");
75. }
76. } while (choice != 6);
77. return 0;
78. }
79. // Add a test result
80. void addTestResult() {
81. if (testCount >= MAX\_TESTS) {
82. printf(". Maximum reached.\n");
83. return;
84. }
85. TestResult \*newTest = &tests[testCount];
86. printf("Enter Test ID: ");
87. scanf("%d", &newTest->id);
88. printf("Enter Patient Name: ");
89. scanf("%s", newTest->name);
90. printf("Enter Patient Age: ");
91. scanf("%d", &newTest->age);
92. // Choose test type
93. printf("Enter test type (1 for Temperature, 2 for Blood Pressure): ");
94. scanf("%d", &newTest->testType);
95. if (newTest->testType == 1) {
96. printf("Enter Temperature: ");
97. scanf("%f", &newTest->data.temperature);
98. } else if (newTest->testType == 2) {
99. printf("Enter Blood Pressure: ");
100. scanf("%d", &newTest->data.bloodPressure);
101. } else {
102. printf("Invalid test type!\n");
103. return;
104. }
105. testCount++;
106. printf("Test result added successfully!\n");
107. }
108. // View a specific test result
109. void viewTestResult() {
110. int id;
111. printf("Enter Test ID to view: ");
112. scanf("%d", &id);
113. for (int i = 0; i < testCount; i++) {
114. if (tests[i].id == id) {
115. printf("Test ID: %d\n", tests[i].id);
116. printf("Patient Name: %s\n", tests[i].name);
117. printf("Patient Age: %d\n", tests[i].age);
118. if (tests[i].testType == 1) {
119. printf("Temperature: %.2f\n", tests[i].data.temperature);
120. } else if (tests[i].testType == 2) {
121. printf("Blood Pressure: %d\n", tests[i].data.bloodPressure);
122. }
123. return;
124. }
125. }
126. printf("Test not found!\n");
127. }
128. // List all test results
129. void listAllTestResults() {
130. if (testCount == 0) {
131. printf("No test results available.\n");
132. return;
133. }
134. for (int i = 0; i < testCount; i++) {
135. printf("Test ID: %d, Patient: %s, Age: %d, ", tests[i].id, tests[i].name, tests[i].age);
136. if (tests[i].testType == 1) {
137. printf("Temperature: %.2f\n", tests[i].data.temperature);
138. } else if (tests[i].testType == 2) {
139. printf("Blood Pressure: %d\n", tests[i].data.bloodPressure);
140. }
141. }
142. }
143. // Update an existing test result
144. void updateTestResult() {
145. int id;
146. printf("Enter Test ID to update: ");
147. scanf("%d", &id);
148. for (int i = 0; i < testCount; i++) {
149. if (tests[i].id == id) {
150. printf("Enter New Patient Name: ");
151. scanf("%s", tests[i].name);
152. printf("Enter New Age: ");
153. scanf("%d", &tests[i].age);
154. printf("Enter new test type (1 for Temperature, 2 for Blood Pressure): ");
155. scanf("%d", &tests[i].testType);
156. if (tests[i].testType == 1) {
157. printf("Enter New Temperature: ");
158. scanf("%f", &tests[i].data.temperature);
159. } else if (tests[i].testType == 2) {
160. printf("Enter New Blood Pressure: ");
161. scanf("%d", &tests[i].data.bloodPressure);
162. } else {
163. printf("Invalid test type!\n");
164. return;
165. }
166. printf("Test result updated successfully!\n");
167. return;
168. }
169. }
170. printf("Test not found!\n");
171. }
172. // Delete a specific test result
173. void deleteTestResult() {
174. int id;
175. printf("Enter Test ID to delete: ");
176. scanf("%d", &id);
177. for (int i = 0; i < testCount; i++) {
178. if (tests[i].id == id) {
179. for (int j = i; j < testCount - 1; j++) {
180. tests[j] = tests[j + 1];
181. }
182. testCount--;
183. printf("Test result deleted successfully!\n");
184. return;
185. }
186. }
187. printf("Test not found!\n");
188. }

**Problem 7: Emergency Contact Management System**

Description: Design a system to manage emergency contacts for patients.

Menu Options:

1. Add Emergency Contact
2. View Emergency Contact
3. Update Emergency Contact
4. Delete Emergency Contact
5. List All Emergency Contacts
6. Exit

**Requirements:**

1. Declare variables for contact details.
2. Use static and const for non-changing contact data.
3. Implement switch case for contact operations.
4. Utilize loops for contact handling.
5. Use pointers for dynamic memory allocation.
6. Create functions for managing contacts.
7. Use arrays for storing contacts.
8. Define structures for contact details.
9. Employ nested structures for detailed contact information.
10. Utilize unions for optional contact data.
11. Apply nested unions for complex contact entries.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #define MAX\_CONTACTS 5
15. // Static and const for non-changing data (hospital details)
16. static const char \*hospitalName = "City Hospital";
17. // Union for optional contact information
18. typedef union {
19. char relationship[30];
20. } EmergencyType;
21. // Nested structure for detailed contact information
22. typedef struct {
23. char name[50];
24. char phone[15];
25. EmergencyType emergencyInfo;
26. } ContactDetails;
27. // Structure for storing emergency contacts
28. typedef struct {
29. int id;
30. ContactDetails contact;
31. } EmergencyContact;
32. // Array to store emergency contacts
33. EmergencyContact contacts[MAX\_CONTACTS];
34. int contactCount = 0;
35. // Function prototypes
36. void addContact();
37. void viewContact();
38. void updateContact();
39. void deleteContact();
40. void listContacts();
41. int main() {
42. int choice;
43. while (1) {
44. printf("\n--- Emergency Contact Management System ---\n");
45. printf("1. Add Emergency Contact\n");
46. printf("2. View Emergency Contact\n");
47. printf("3. Update Emergency Contact\n");
48. printf("4. Delete Emergency Contact\n");
49. printf("5. List All Emergency Contacts\n");
50. printf("6. Exit\n");
51. printf("Enter your choice: ");
52. scanf("%d", &choice);
53. switch (choice) {
54. case 1:
55. addContact();
56. break;
57. case 2:
58. viewContact();
59. break;
60. case 3:
61. updateContact();
62. break;
63. case 4:
64. deleteContact();
65. break;
66. case 5:
67. listContacts();
68. break;
69. case 6:
70. printf("Exiting...\n");
71. return 0;
72. default:
73. printf("Invalid choice. Please try again.\n");
74. }
75. }
76. }
77. // Function to add a new emergency contact
78. void addContact() {
79. if (contactCount >= MAX\_CONTACTS) {
80. printf("Contact list is full. Cannot add more contacts.\n");
81. return;
82. }
83. EmergencyContact \*newContact = &contacts[contactCount];
84. printf("Enter Contact ID: ");
85. scanf("%d", &newContact->id);
86. printf("Enter Contact Name: ");
87. scanf("%s", newContact->contact.name);
88. printf("Enter Contact Phone Number: ");
89. scanf("%s", newContact->contact.phone);
90. printf("Enter Relationship with Patient: ");
91. scanf("%s", newContact->contact.emergencyInfo.relationship);
92. contactCount++;
93. printf("Emergency contact added successfully!\n");
94. }
95. // Function to view a specific emergency contact
96. void viewContact() {
97. int id;
98. printf("Enter Contact ID to view: ");
99. scanf("%d", &id);
100. for (int i = 0; i < contactCount; i++) {
101. if (contacts[i].id == id) {
102. printf("\nContact ID: %d\n", contacts[i].id);
103. printf("Name: %s\n", contacts[i].contact.name);
104. printf("Phone: %s\n", contacts[i].contact.phone);
105. printf("Relationship: %s\n", contacts[i].contact.emergencyInfo.relationship);
106. return;
107. }
108. }
109. printf("Contact not found.\n");
110. }
111. // Function to update a specific emergency contact
112. void updateContact() {
113. int id;
114. printf("Enter Contact ID to update: ");
115. scanf("%d", &id);
116. for (int i = 0; i < contactCount; i++) {
117. if (contacts[i].id == id) {
118. printf("Enter New Contact Name: ");
119. scanf("%s", contacts[i].contact.name);
120. printf("Enter New Contact Phone Number: ");
121. scanf("%s", contacts[i].contact.phone);
122. printf("Enter New Relationship with Patient: ");
123. scanf("%s", contacts[i].contact.emergencyInfo.relationship);
124. printf("Emergency contact updated successfully!\n");
125. return;
126. }
127. }
128. printf("Contact not found.\n");
129. }
130. // Function to delete a specific emergency contact
131. void deleteContact() {
132. int id;
133. printf("Enter Contact ID to delete: ");
134. scanf("%d", &id);
135. for (int i = 0; i < contactCount; i++) {
136. if (contacts[i].id == id) {
137. // Shift the array to delete the contact
138. for (int j = i; j < contactCount - 1; j++) {
139. contacts[j] = contacts[j + 1];
140. }
141. contactCount--;
142. printf("Emergency contact deleted successfully!\n");
143. return;
144. }
145. }
146. printf("Contact not found.\n");
147. }
148. // Function to list all emergency contacts
149. void listContacts() {
150. if (contactCount == 0) {
151. printf("No emergency contacts to display.\n");
152. return;
153. }
154. printf("\nList of Emergency Contacts:\n");
155. for (int i = 0; i < contactCount; i++) {
156. printf("ID: %d, Name: %s, Phone: %s, Relationship: %s\n",
157. contacts[i].id, contacts[i].contact.name,
158. contacts[i].contact.phone, contacts[i].contact.emergencyInfo.relationship);
159. }
160. }

**Problem 8: Medical Record Update System**

**Description:** Create a system for updating patient medical records.

**Menu Options:**

1. Add Medical Record
2. View Medical Record
3. Update Medical Record
4. Delete Medical Record
5. List All Medical Records
6. Exit

**Requirements:**

1. Use variables for record details.
2. Apply static and const for immutable data like record ID.
3. Implement switch case for update operations.
4. Utilize loops for record updating.
5. Use pointers for handling records.
6. Create functions for record management.
7. Use arrays for storing records.
8. Define structures for record details.
9. Employ nested structures for detailed medical history.
10. Utilize unions for optional record fields.
11. Apply nested unions for complex record data.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #define MAX\_RECORDS 5
15. // Static and const for immutable data
16. static const char \*hospitalName = "City Hospital";
17. // Union for optional record fields
18. typedef union {
19. char allergy[30];
20. } OptionalField;
21. // Nested structure for detailed medical history
22. typedef struct {
23. char diagnosis[100];
24. char treatment[100];
25. OptionalField optional;
26. } MedicalHistory;
27. // Structure for storing medical records
28. typedef struct {
29. int id;
30. char name[50];
31. int age;
32. MedicalHistory history;
33. } MedicalRecord;
34. // Array to store medical records
35. MedicalRecord records[MAX\_RECORDS];
36. int recordCount = 0;  // Track the number of records
37. // Function prototypes
38. void addRecord();
39. void viewRecord();
40. void updateRecord();
41. void deleteRecord();
42. void listRecords();
43. int main() {
44. int choice;
45. while (1) {
46. printf("\n--- Medical Record Update System ---\n");
47. printf("1. Add Medical Record\n");
48. printf("2. View Medical Record\n");
49. printf("3. Update Medical Record\n");
50. printf("4. Delete Medical Record\n");
51. printf("5. List All Medical Records\n");
52. printf("6. Exit\n");
53. printf("Enter your choice: ");
54. scanf("%d", &choice);
55. switch (choice) {
56. case 1:
57. addRecord();
58. break;
59. case 2:
60. viewRecord();
61. break;
62. case 3:
63. updateRecord();
64. break;
65. case 4:
66. deleteRecord();
67. break;
68. case 5:
69. listRecords();
70. break;
71. case 6:
72. printf("Exiting...\n");
73. return 0;
74. default:
75. printf("Invalid choice. Please try again.\n");
76. }
77. }
78. }
79. // Function to add a new medical record
80. void addRecord() {
81. if (recordCount >= MAX\_RECORDS) {
82. printf("Record list is full. Cannot add more records.\n");
83. return;
84. }
85. MedicalRecord \*newRecord = &records[recordCount];
86. printf("Enter Record ID: ");
87. scanf("%d", &newRecord->id);
88. printf("Enter Patient Name: ");
89. scanf("%s", newRecord->name);
90. printf("Enter Patient Age: ");
91. scanf("%d", &newRecord->age);
92. printf("Enter Diagnosis: ");
93. scanf("%s", newRecord->history.diagnosis);
94. printf("Enter Treatment: ");
95. scanf("%s", newRecord->history.treatment);
96. printf("Enter Optional Allergy Info (leave blank if none): ");
97. scanf("%s", newRecord->history.optional.allergy);
98. recordCount++;
99. printf("Medical record added successfully!\n");
100. }
101. // Function to view a specific medical record
102. void viewRecord() {
103. int id;
104. printf("Enter Record ID to view: ");
105. scanf("%d", &id);
106. for (int i = 0; i < recordCount; i++) {
107. if (records[i].id == id) {
108. printf("\nRecord ID: %d\n", records[i].id);
109. printf("Name: %s\n", records[i].name);
110. printf("Age: %d\n", records[i].age);
111. printf("Diagnosis: %s\n", records[i].history.diagnosis);
112. printf("Treatment: %s\n", records[i].history.treatment);
113. printf("Allergy Info: %s\n", records[i].history.optional.allergy);
114. return;
115. }
116. }
117. printf("Record not found.\n");
118. }
119. // Function to update a specific medical record
120. void updateRecord() {
121. int id;
122. printf("Enter Record ID to update: ");
123. scanf("%d", &id);
124. for (int i = 0; i < recordCount; i++) {
125. if (records[i].id == id) {
126. printf("Enter New Patient Name: ");
127. scanf("%s", records[i].name);
128. printf("Enter New Patient Age: ");
129. scanf("%d", &records[i].age);
130. printf("Enter New Diagnosis: ");
131. scanf("%s", records[i].history.diagnosis);
132. printf("Enter New Treatment: ");
133. scanf("%s", records[i].history.treatment);
134. printf("Enter New Optional Allergy Info: ");
135. scanf("%s", records[i].history.optional.allergy);
136. printf("Medical record updated successfully!\n");
137. return;
138. }
139. }
140. printf("Record not found.\n");
141. }
142. // Function to delete a specific medical record
143. void deleteRecord() {
144. int id;
145. printf("Enter Record ID to delete: ");
146. scanf("%d", &id);
147. for (int i = 0; i < recordCount; i++) {
148. if (records[i].id == id) {
149. // Shift the array to delete the record
150. for (int j = i; j < recordCount - 1; j++) {
151. records[j] = records[j + 1];
152. }
153. recordCount--;
154. printf("Medical record deleted successfully!\n");
155. return;
156. }
157. }
158. printf("Record not found.\n");
159. }
160. // Function to list all medical records
161. void listRecords() {
162. if (recordCount == 0) {
163. printf("No medical records to display.\n");
164. return;
165. }
166. printf("\nList of Medical Records:\n");
167. for (int i = 0; i < recordCount; i++) {
168. printf("ID: %d, Name: %s, Age: %d, Diagnosis: %s, Treatment: %s, Allergy Info: %s\n",
169. records[i].id, records[i].name, records[i].age,
170. records[i].history.diagnosis, records[i].history.treatment,
171. records[i].history.optional.allergy);
172. }
173. }

**Problem 9: Patient Diet Plan Management**

**Description:** Develop a system to manage diet plans for patients.

**Menu Options:**

1. Add Diet Plan
2. View Diet Plan
3. Update Diet Plan
4. Delete Diet Plan
5. List All Diet Plans
6. Exit

**Requirements:**

1. Declare variables for diet plan details.
2. Use static and const for fixed dietary guidelines.
3. Implement switch case for diet plan operations.
4. Utilize loops for diet plan handling.
5. Use pointers for dynamic diet data.
6. Create functions for diet plan management.
7. Use arrays for storing diet plans.
8. Define structures for diet plan details.
9. Employ nested structures for detailed dietary breakdowns.
10. Use unions for optional diet attributes.
11. Apply nested unions for complex diet plan data.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #define MAX\_DIET\_PLANS 5
15. // Static and const
16. static const char \*dietaryGuidelines = "Follow your prescribed diet and exercise routine.";
17. // Union for optional diet attributes
18. typedef union {
19. char foodPreference[50];
20. char foodRestriction[50];
21. } OptionalDiet;
22. // Structure for storing diet plan details
23. typedef struct {
24. int id;
25. char patientName[50];
26. int calories;
27. int carbs;
28. int proteins;
29. OptionalDiet optional;
30. } DietPlan;
31. // Array to store diet plans
32. DietPlan dietPlans[MAX\_DIET\_PLANS];
33. int dietPlanCount = 0;
34. // Function prototypes
35. void addDietPlan();
36. void viewDietPlan();
37. void updateDietPlan();
38. void deleteDietPlan();
39. void listDietPlans();
40. int main() {
41. int choice;
42. while (1) {
43. printf("\n--- Patient Diet Plan Management System ---\n");
44. printf("1. Add Diet Plan\n");
45. printf("2. View Diet Plan\n");
46. printf("3. Update Diet Plan\n");
47. printf("4. Delete Diet Plan\n");
48. printf("5. List All Diet Plans\n");
49. printf("6. Exit\n");
50. printf("Enter your choice: ");
51. scanf("%d", &choice);
52. switch (choice) {
53. case 1:
54. addDietPlan();
55. break;
56. case 2:
57. viewDietPlan();
58. break;
59. case 3:
60. updateDietPlan();
61. break;
62. case 4:
63. deleteDietPlan();
64. break;
65. case 5:
66. listDietPlans();
67. break;
68. case 6:
69. printf("Exiting...\n");
70. return 0;
71. default:
72. printf("Invalid choice. Please try again.\n");
73. }
74. }
75. }
76. // Function to add a new diet plan
77. void addDietPlan() {
78. if (dietPlanCount >= MAX\_DIET\_PLANS) {
79. printf("Diet plan list is full. Cannot add more plans.\n");
80. return;
81. }
82. DietPlan \*newPlan = &dietPlans[dietPlanCount];
83. printf("Enter Diet Plan ID: ");
84. scanf("%d", &newPlan->id);
85. printf("Enter Patient Name: ");
86. scanf("%s", newPlan->patientName);
87. printf("Enter Calories per Day: ");
88. scanf("%d", &newPlan->calories);
89. printf("Enter Carbs per Day: ");
90. scanf("%d", &newPlan->carbs);
91. printf("Enter Proteins per Day: ");
92. scanf("%d", &newPlan->proteins);
93. printf("Enter Food Preference (leave blank if none): ");
94. scanf("%s", newPlan->optional.foodPreference);
95. printf("Enter Food Restriction (leave blank if none): ");
96. scanf("%s", newPlan->optional.foodRestriction);
97. dietPlanCount++;
98. printf("Diet plan added successfully!\n");
99. }
100. // Function to view a specific diet plan
101. void viewDietPlan() {
102. int id;
103. printf("Enter Diet Plan ID to view: ");
104. scanf("%d", &id);
105. for (int i = 0; i < dietPlanCount; i++) {
106. if (dietPlans[i].id == id) {
107. printf("\nDiet Plan ID: %d\n", dietPlans[i].id);
108. printf("Patient Name: %s\n", dietPlans[i].patientName);
109. printf("Calories per Day: %d\n", dietPlans[i].calories);
110. printf("Carbs per Day: %d\n", dietPlans[i].carbs);
111. printf("Proteins per Day: %d\n", dietPlans[i].proteins);
112. printf("Food Preference: %s\n", dietPlans[i].optional.foodPreference);
113. printf("Food Restriction: %s\n", dietPlans[i].optional.foodRestriction);
114. return;
115. }
116. }
117. printf("Diet plan not found.\n");
118. }
119. // Function to update a specific diet plan
120. void updateDietPlan() {
121. int id;
122. printf("Enter Diet Plan ID to update: ");
123. scanf("%d", &id);
124. for (int i = 0; i < dietPlanCount; i++) {
125. if (dietPlans[i].id == id) {
126. printf("Enter New Patient Name: ");
127. scanf("%s", dietPlans[i].patientName);
128. printf("Enter New Calories per Day: ");
129. scanf("%d", &dietPlans[i].calories);
130. printf("Enter New Carbs per Day: ");
131. scanf("%d", &dietPlans[i].carbs);
132. printf("Enter New Proteins per Day: ");
133. scanf("%d", &dietPlans[i].proteins);
134. printf("Enter New Food Preference: ");
135. scanf("%s", dietPlans[i].optional.foodPreference);
136. printf("Enter New Food Restriction: ");
137. scanf("%s", dietPlans[i].optional.foodRestriction);
138. printf("Diet plan updated successfully!\n");
139. return;
140. }
141. }
142. printf("Diet plan not found.\n");
143. }
144. // Function to delete a specific diet plan
145. void deleteDietPlan() {
146. int id;
147. printf("Enter Diet Plan ID to delete: ");
148. scanf("%d", &id);
149. for (int i = 0; i < dietPlanCount; i++) {
150. if (dietPlans[i].id == id) {
151. // Shift the array to delete the plan
152. for (int j = i; j < dietPlanCount - 1; j++) {
153. dietPlans[j] = dietPlans[j + 1];
154. }
155. dietPlanCount--;
156. printf("Diet plan deleted successfully!\n");
157. return;
158. }
159. }
160. printf("Diet plan not found.\n");
161. }
162. // Function to list all diet plans
163. void listDietPlans() {
164. if (dietPlanCount == 0) {
165. printf("No diet plans to display.\n");
166. return;
167. }
168. printf("\nList of Diet Plans:\n");
169. for (int i = 0; i < dietPlanCount; i++) {
170. printf("ID: %d, Patient: %s, Calories: %d, Carbs: %d, Proteins: %d, Preference: %s, Restriction: %s\n",
171. dietPlans[i].id, dietPlans[i].patientName, dietPlans[i].calories,
172. dietPlans[i].carbs, dietPlans[i].proteins,
173. dietPlans[i].optional.foodPreference, dietPlans[i].optional.foodRestriction);
174. }
175. }

**Problem 10: Surgery Scheduling System**

**Description:** Design a system for scheduling surgeries.

**Menu Options:**

1. Schedule Surgery
2. View Surgery Schedule
3. Update Surgery Schedule
4. Cancel Surgery
5. List All Surgeries
6. Exit

**Requirements:**

1. Use variables for surgery details.
2. Apply static and const for immutable data like surgery types.
3. Implement switch case for scheduling operations.
4. Utilize loops for surgery scheduling.
5. Use pointers for handling surgery data.
6. Create functions for surgery management.
7. Use arrays for storing surgery schedules.
8. Define structures for surgery details.
9. Employ nested structures for detailed surgery information.
10. Utilize unions for optional surgery data.
11. Apply nested unions for complex surgery entries.
12. #include <stdio.h>
13. #include <stdlib.h>
14. #define MAX\_SURGERIES 5
15. // Define surgery types
16. const char \*surgeryTypes[] = {"Appendectomy", "C-section", "Knee Replacement", "Heart Bypass"};
17. // Structure for surgery details
18. typedef struct {
19. int id;
20. char patientName[50];
21. int surgeryType;
22. char surgeryDate[20];
23. } Surgery;
24. // Array to store surgery schedules
25. Surgery surgeries[MAX\_SURGERIES];
26. int surgeryCount = 0;
27. // Function prototypes
28. void scheduleSurgery();
29. void viewSurgery();
30. void updateSurgery();
31. void cancelSurgery();
32. void listSurgeries();
33. int main() {
34. int choice;
35. while (1) {
36. printf("\n--- Surgery Scheduling System ---\n");
37. printf("1. Schedule Surgery\n");
38. printf("2. View Surgery Schedule\n");
39. printf("3. Update Surgery Schedule\n");
40. printf("4. Cancel Surgery\n");
41. printf("5. List All Surgeries\n");
42. printf("6. Exit\n");
43. printf("Enter your choice: ");
44. scanf("%d", &choice);
45. switch (choice) {
46. case 1:
47. scheduleSurgery();
48. break;
49. case 2:
50. viewSurgery();
51. break;
52. case 3:
53. updateSurgery();
54. break;
55. case 4:
56. cancelSurgery();
57. break;
58. case 5:
59. listSurgeries();
60. break;
61. case 6:
62. printf("Exiting...\n");
63. return 0;
64. default:
65. printf("Invalid choice. \n");
66. }
67. }
68. }
69. // Function to schedule a new surgery
70. void scheduleSurgery() {
71. if (surgeryCount >= MAX\_SURGERIES) {
72. printf("Surgery schedule is full.\n");
73. return;
74. }
75. Surgery \*newSurgery = &surgeries[surgeryCount];
76. printf("Enter Surgery ID: ");
77. scanf("%d", &newSurgery->id);
78. printf("Enter Patient Name: ");
79. scanf("%s", newSurgery->patientName);
80. printf("Select Surgery Type:\n");
81. for (int i = 0; i < 4; i++) {
82. printf("%d. %s\n", i + 1, surgeryTypes[i]);
83. }
84. printf("Enter surgery type number: ");
85. scanf("%d", &newSurgery->surgeryType);
86. newSurgery->surgeryType--;
87. printf("Enter Surgery Date (DD/MM/YYYY): ");
88. scanf("%s", newSurgery->surgeryDate);
89. surgeryCount++;
90. printf("Surgery scheduled successfully!\n");
91. }
92. // Function to view a specific surgery schedule
93. void viewSurgery() {
94. int id;
95. printf("Enter Surgery ID to view: ");
96. scanf("%d", &id);
97. for (int i = 0; i < surgeryCount; i++) {
98. if (surgeries[i].id == id) {
99. printf("\nSurgery ID: %d\n", surgeries[i].id);
100. printf("Patient Name: %s\n", surgeries[i].patientName);
101. printf("Surgery Type: %s\n", surgeryTypes[surgeries[i].surgeryType]);
102. printf("Surgery Date: %s\n", surgeries[i].surgeryDate);
103. return;
104. }
105. }
106. printf("Surgery schedule not found.\n");
107. }
108. // Function to update a surgery schedule
109. void updateSurgery() {
110. int id;
111. printf("Enter Surgery ID to update: ");
112. scanf("%d", &id);
113. for (int i = 0; i < surgeryCount; i++) {
114. if (surgeries[i].id == id) {
115. printf("Enter New Patient Name: ");
116. scanf("%s", surgeries[i].patientName);
117. printf("Select New Surgery Type:\n");
118. for (int j = 0; j < 4; j++) {
119. printf("%d. %s\n", j + 1, surgeryTypes[j]);
120. }
121. printf("Enter new surgery type number: ");
122. scanf("%d", &surgeries[i].surgeryType);
123. surgeries[i].surgeryType--;  // Adjusting index
124. printf("Enter New Surgery Date (DD/MM/YYYY): ");
125. scanf("%s", surgeries[i].surgeryDate);
126. printf("Surgery schedule updated successfully!\n");
127. return;
128. }
129. }
130. printf("Surgery schedule not found.\n");
131. }
132. // Function to cancel a surgery
133. void cancelSurgery() {
134. int id;
135. printf("Enter Surgery ID to cancel: ");
136. scanf("%d", &id);
137. for (int i = 0; i < surgeryCount; i++) {
138. if (surgeries[i].id == id) {
139. // Shift the array to cancel the surgery
140. for (int j = i; j < surgeryCount - 1; j++) {
141. surgeries[j] = surgeries[j + 1];
142. }
143. surgeryCount--;
144. printf("Surgery canceled successfully!\n");
145. return;
146. }
147. }
148. printf("Surgery schedule not found.\n");
149. }
150. // Function to list all surgeries
151. void listSurgeries() {
152. if (surgeryCount == 0) {
153. printf("No surgeries scheduled.\n");
154. return;
155. }
156. printf("\nList of Surgeries:\n");
157. for (int i = 0; i < surgeryCount; i++) {
158. printf("ID: %d, Patient: %s, Surgery: %s, Date: %s\n",
159. surgeries[i].id, surgeries[i].patientName, surgeryTypes[surgeries[i].surgeryType], surgeries[i].surgeryDate);
160. }
161. }

**Problem 11: Prescription Management System**

**Description:** Develop a system to manage patient prescriptions.

**Menu Options:**

1. Add Prescription
2. View Prescription
3. Update Prescription
4. Delete Prescription
5. List All Prescriptions
6. Exit

**Requirements:**

1. Declare variables for prescription details.
2. Use static and const for fixed prescription guidelines.
3. Implement switch case for prescription operations.
4. Utilize loops for prescription handling.
5. Use pointers for dynamic prescription data.
6. Create functions for prescription management.
7. Use arrays for storing prescriptions.
8. Define structures for prescription details.
9. Employ nested structures for detailed prescription information.
10. Use unions for optional prescription fields.
11. Apply nested unions for complex prescription data.
12. #include <stdio.h>
13. #include <string.h>
14. // Define a structure for prescription details
15. struct Prescription {
16. char patientName[50];
17. char medicine[30];
18. int quantity;
19. };
20. // Declare an array to store prescriptions
21. struct Prescription prescriptions[100];
22. int prescriptionCount = 0;
23. // Function prototypes
24. void addPrescription();
25. void viewPrescription();
26. void listAllPrescriptions();
27. int main() {
28. int choice;
29. // Menu loop
30. do {
31. printf("\n--- Prescription Management System ---\n");
32. printf("1. Add Prescription\n");
33. printf("2. View Prescription\n");
34. printf("3. List All Prescriptions\n");
35. printf("4. Exit\n");
36. printf("Enter your choice: ");
37. scanf("%d", &choice);
38. switch (choice) {
39. case 1:
40. addPrescription();
41. break;
42. case 2:
43. viewPrescription();
44. break;
45. case 3:
46. listAllPrescriptions();
47. break;
48. case 4:
49. printf("Exiting the system...\n");
50. break;
51. default:
52. printf("Invalid choice. Please try again.\n");
53. }
54. } while (choice != 4);
55. return 0;
56. }
57. // Function to add a prescription
58. void addPrescription() {
59. if (prescriptionCount < 100) {
60. printf("Enter patient name: ");
61. getchar(); // To consume the newline character left by scanf
62. fgets(prescriptions[prescriptionCount].patientName, 50, stdin);
63. prescriptions[prescriptionCount].patientName[strcspn(prescriptions[prescriptionCount].patientName, "\n")] = 0; // Remove newline
64. printf("Enter medicine name: ");
65. fgets(prescriptions[prescriptionCount].medicine, 30, stdin);
66. prescriptions[prescriptionCount].medicine[strcspn(prescriptions[prescriptionCount].medicine, "\n")] = 0; // Remove newline
67. printf("Enter quantity: ");
68. scanf("%d", &prescriptions[prescriptionCount].quantity);
69. prescriptionCount++;
70. printf("Prescription added successfully!\n");
71. } else {
72. printf("Cannot add more prescriptions. List is full.\n");
73. }
74. }
75. // Function to view a specific prescription
76. void viewPrescription() {
77. int index;
78. printf("Enter prescription index to view (0-%d): ", prescriptionCount - 1);
79. scanf("%d", &index);
80. if (index >= 0 && index < prescriptionCount) {
81. printf("\nPatient Name: %s\n", prescriptions[index].patientName);
82. printf("Medicine: %s\n", prescriptions[index].medicine);
83. printf("Quantity: %d\n", prescriptions[index].quantity);
84. } else {
85. printf("Invalid index!\n");
86. }
87. }
88. // Function to list all prescriptions
89. void listAllPrescriptions() {
90. if (prescriptionCount == 0) {
91. printf("No prescriptions available.\n");
92. } else {
93. for (int i = 0; i < prescriptionCount; i++) {
94. printf("\nPrescription %d:\n", i + 1);
95. printf("Patient Name: %s\n", prescriptions[i].patientName);
96. printf("Medicine: %s\n", prescriptions[i].medicine);
97. printf("Quantity: %d\n", prescriptions[i].quantity);
98. }
99. }
100. }

**Problem 12: Doctor Consultation Management**

**Description:** Create a system for managing doctor consultations.

**Menu Options:**

1. Schedule Consultation
2. View Consultation
3. Update Consultation
4. Cancel Consultation
5. List All Consultations
6. Exit

**Requirements:**

1. Use variables for consultation details.
2. Apply static and const for non-changing data like consultation fees.
3. #include <stdio.h>
4. #define MAX\_CONSULTATIONS 100
5. #define CONSULTATION\_FEE 100.0
6. struct Consultation {
7. char patientName[50];
8. char doctorName[50];
9. char date[20];
10. char time[10];
11. };
12. struct Consultation consultations[MAX\_CONSULTATIONS];
13. int consultationCount = 0;
14. void scheduleConsultation() {
15. if (consultationCount < MAX\_CONSULTATIONS) {
16. printf("Enter patient name: ");
17. scanf("%49s", consultations[consultationCount].patientName);
18. printf("Enter doctor name: ");
19. scanf("%49s", consultations[consultationCount].doctorName);
20. printf("Enter consultation date (DD/MM/YYYY): ");
21. scanf("%19s", consultations[consultationCount].date);
22. printf("Enter consultation time (HH:MM): ");
23. scanf("%9s", consultations[consultationCount].time);
24. consultationCount++;
25. printf("Consultation scheduled successfully! Fee: $%.2f\n", CONSULTATION\_FEE);
26. } else {
27. printf("Cannot schedule more consultations. List is full.\n");
28. }
29. }
30. void viewConsultation() {
31. int index;
32. printf("Enter consultation index (0-%d): ", consultationCount - 1);
33. scanf("%d", &index);
34. if (index >= 0 && index < consultationCount) {
35. printf("\nPatient Name: %s\n", consultations[index].patientName);
36. printf("Doctor Name: %s\n", consultations[index].doctorName);
37. printf("Date: %s\n", consultations[index].date);
38. printf("Time: %s\n", consultations[index].time);
39. } else {
40. printf("Invalid index!\n");
41. }
42. }
43. void listAllConsultations() {
44. if (consultationCount == 0) {
45. printf("No consultations available.\n");
46. } else {
47. for (int i = 0; i < consultationCount; i++) {
48. printf("\nConsultation %d:\n", i + 1);
49. printf("Patient Name: %s\n", consultations[i].patientName);
50. printf("Doctor Name: %s\n", consultations[i].doctorName);
51. printf("Date: %s\n", consultations[i].date);
52. printf("Time: %s\n", consultations[i].time);
53. }
54. }
55. }
56. int main() {
57. int choice;
58. do {
59. printf("\n--- Doctor Consultation Management ---\n");
60. printf("1. Schedule Consultation\n");
61. printf("2. View Consultation\n");
62. printf("3. List All Consultations\n");
63. printf("4. Exit\n");
64. printf("Enter your choice: ");
65. scanf("%d", &choice);
66. switch (choice) {
67. case 1:
68. scheduleConsultation();
69. break;
70. case 2:
71. viewConsultation();
72. break;
73. case 3:
74. listAllConsultations();
75. break;
76. case 4:
77. printf("Exiting the system...\n");
78. break;
79. default:
80. printf("Invalid choice. Please try again.\n");
81. }
82. } while (choice != 4);
83. return 0;
84. }

**Linked List**

**Problem 1: Patient Queue Management**

Description: Implement a linked list to manage a queue of patients waiting for consultation. Operations:

1. Create a new patient queue.
2. Insert a patient into the queue.
3. Display the current queue of patients.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure
8. struct Node {
9. char name[50];
10. struct Node \*next;
11. } \*first = NULL, \*rear = NULL;
12. void createQueue();
13. void insertPatient(char name[]);
14. void displayQueue();
15. int main() {
16. createQueue();
17. insertPatient("Pavan");
18. insertPatient("Kevin");
19. insertPatient("Noel");
20. printf("Current Queue:\n");
21. displayQueue();
22. return 0;
23. }
24. void createQueue() {
25. first = rear = NULL;
26. }
27. // Function to insert a patient into the queue
28. void insertPatient(char name[]) {
29. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
30. strcpy(temp->name, name);
31. temp->next = NULL;
32. if (first == NULL) {
33. first = rear = temp;
34. } else {
35. rear->next = temp;
36. rear = temp;
37. }
38. }
39. // Function to display the current queue of patients
40. void displayQueue() {
41. struct Node \*p = first;
42. while (p != NULL) {
43. printf("%s -> ", p->name);
44. p = p->next;
45. }
46. printf("NULL\n");
47. }

**Problem 2: Hospital Ward Allocation**

Description: Use a linked list to allocate beds in a hospital ward. Operations:

1. Create a list of available beds.
2. Insert a patient into an available bed.
3. Display the current bed allocation.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for bed allocation
8. struct Node {
9. int bedNumber;
10. char patientName[50];
11. struct Node \*next;
12. } \*first = NULL;
13. void createBedList();
14. void insertPatient(int bedNumber, char patientName[]);
15. void displayBedAllocation();
16. int main() {
17. createBedList();
18. insertPatient(101, "Pavan");
19. insertPatient(102, "kevin");
20. insertPatient(103, "Noel");
21. printf("Current Bed Allocation:\n");
22. displayBedAllocation();
23. return 0;
24. }
25. // Function to create a list of available beds
26. void createBedList() {
27. first = NULL;
28. }
29. // Function to insert a patient into an available bed
30. void insertPatient(int bedNumber, char patientName[]) {
31. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
32. temp->bedNumber = bedNumber;
33. strcpy(temp->patientName, patientName);
34. temp->next = NULL;
35. if (first == NULL) {
36. first = temp;
37. } else {
38. struct Node \*p = first;
39. while (p->next != NULL) {
40. p = p->next;
41. }
42. p->next = temp;
43. }
44. }
45. // Function to display the current bed allocation
46. void displayBedAllocation() {
47. struct Node \*p = first;
48. while (p != NULL) {
49. printf("Bed Number: %d, Patient: %s\n", p->bedNumber, p->patientName);
50. p = p->next;
51. }
52. }

**Problem 3: Medical Inventory Tracking**

Description: Maintain a linked list to track inventory items in a medical store. Operations:

1. Create an inventory list.
2. Insert a new inventory item.
3. Display the current inventory.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for inventory items
8. struct Node {
9. int itemId;
10. char itemName[50];
11. int quantity;
12. struct Node \*next;
13. } \*first = NULL;
14. // Function prototypes
15. void createInventoryList();
16. void insertInventoryItem(int itemId, char itemName[], int quantity);
17. void displayInventory();
18. int main() {
19. createInventoryList();
20. insertInventoryItem(101, "Dolo 650", 50);
21. insertInventoryItem(102, "Paracetamol", 30);
22. insertInventoryItem(103, "Cough Syrup", 20);
23. printf("Current Inventory:\n");
24. displayInventory();
25. return 0;
26. }
27. // Function to create an inventory list
28. void createInventoryList() {
29. first = NULL;
30. }
31. // Function to insert a new inventory item
32. void insertInventoryItem(int itemId, char itemName[], int quantity) {
33. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
34. temp->itemId = itemId;
35. strcpy(temp->itemName, itemName);
36. temp->quantity = quantity;
37. temp->next = NULL;
38. if (first == NULL) {
39. first = temp;
40. } else {
41. struct Node \*p = first;
42. while (p->next != NULL) {
43. p = p->next;
44. }
45. p->next = temp;
46. }
47. }
48. // Function to display the current inventory
49. void displayInventory() {
50. struct Node \*p = first;
51. while (p != NULL) {
52. printf("Item ID: %d, Item Name: %s, Quantity: %d\n", p->itemId, p->itemName, p->quantity);
53. p = p->next;
54. }
55. }

**Problem 4: Doctor Appointment Scheduling**

Description: Develop a linked list to schedule doctor appointments. Operations:

1. Create an appointment list.
2. Insert a new appointment.
3. Display all scheduled appointments.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for appointments
8. struct Node {
9. int appointmentId;
10. char patientName[50];
11. char doctorName[50];
12. char appointmentDate[20];
13. struct Node \*next;
14. } \*first = NULL;
15. // Function prototypes
16. void createAppointmentList();
17. void insertAppointment(int appointmentId, char patientName[], char doctorName[], char appointmentDate[]);
18. void displayAppointments();
19. int main() {
20. createAppointmentList();
21. insertAppointment(1, "Pavan", "Dr. Patil", "2025-01-20");
22. insertAppointment(2, "Kevin", "Dr. Nitin", "2025-01-21");
23. insertAppointment(3, "Noel", "Dr. Sardar", "2025-01-22");
24. printf("Scheduled Appointments:\n");
25. displayAppointments();
26. return 0;
27. }
28. // Function to create an appointment list
29. void createAppointmentList() {
30. first = NULL;
31. }
32. // Function to insert a new appointment
33. void insertAppointment(int appointmentId, char patientName[], char doctorName[], char appointmentDate[]) {
34. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
35. temp->appointmentId = appointmentId;
36. strcpy(temp->patientName, patientName);
37. strcpy(temp->doctorName, doctorName);
38. strcpy(temp->appointmentDate, appointmentDate);
39. temp->next = NULL;
40. if (first == NULL) {
41. first = temp;
42. } else {
43. struct Node \*p = first;
44. while (p->next != NULL) {
45. p = p->next;
46. }
47. p->next = temp;
48. }
49. }
50. // Function to display all scheduled appointments
51. void displayAppointments() {
52. struct Node \*p = first;
53. while (p != NULL) {
54. printf("Appointment ID: %d, Patient: %s, Doctor: %s, Date: %s\n", p->appointmentId, p->patientName, p->doctorName, p->appointmentDate);
55. p = p->next;
56. }
57. }

**Problem 5: Emergency Contact List**Description: Implement a linked list to manage emergency contacts for hospital staff. Operations:

1. Create a contact list.
2. Insert a new contact.
3. Display all emergency contacts.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for emergency contacts
8. struct Node {
9. int contactId;
10. char name[50];
11. char phone[15];
12. struct Node \*next;
13. } \*first = NULL;
14. // Function prototypes
15. void createContactList();
16. void insertContact(int contactId, char name[], char phone[]);
17. void displayContacts();
18. int main() {
19. createContactList();
20. insertContact(1, "Dr. Patil", "755860254");
21. insertContact(2, "Nurse Priyanka", "9876543210");
22. insertContact(3, "Dr. Pavan", "9541234567");
23. printf("Emergency Contact List:\n");
24. displayContacts();
25. return 0;
26. }
27. // Function to create a contact list
28. void createContactList() {
29. first = NULL;  // Start with an empty list
30. }
31. // Function to insert a new contact into the list
32. void insertContact(int contactId, char name[], char phone[]) {
33. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
34. temp->contactId = contactId;
35. strcpy(temp->name, name);
36. strcpy(temp->phone, phone);
37. temp->next = NULL;
38. if (first == NULL) {
39. first = temp;
40. } else {
41. struct Node \*p = first;
42. while (p->next != NULL) {
43. p = p->next;
44. }
45. p->next = temp;
46. }
47. }
48. // Function to display all emergency contacts
49. void displayContacts() {
50. struct Node \*p = first;
51. while (p != NULL) {
52. printf("Contact ID: %d, Name: %s, Phone: %s\n", p->contactId, p->name, p->phone);
53. p = p->next;
54. }
55. }

**Problem 6: Surgery Scheduling System**: Description: Use a linked list to manage surgery schedules. Operations:

1. Create a surgery schedule.
2. Insert a new surgery into the schedule.
3. Display all scheduled surgeries.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for surgery schedules
8. struct Node {
9. int surgeryId;
10. char patientName[50];
11. char doctorName[50];
12. char surgeryDate[20];
13. struct Node \*next;
14. } \*first = NULL;
15. // Function prototypes
16. void createSurgerySchedule();
17. void insertSurgery(int surgeryId, char patientName[], char doctorName[], char surgeryDate[]);
18. void displaySurgeries();
19. int main() {
20. createSurgerySchedule();
21. insertSurgery(1, "Pavan", "Dr. Patil", "2025-01-20");
22. insertSurgery(2, "Kevin", "Dr. Nitin", "2025-01-21");
23. insertSurgery(3, "Noel", "Dr. Sardar", "2025-01-22");
24. printf("Scheduled Surgeries:\n");
25. displaySurgeries();
26. return 0;
27. }
28. // Function to create a surgery schedule
29. void createSurgerySchedule() {
30. first = NULL;
31. }
32. // Function to insert a new surgery into the schedule
33. void insertSurgery(int surgeryId, char patientName[], char doctorName[], char surgeryDate[]) {
34. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
35. temp->surgeryId = surgeryId;
36. strcpy(temp->patientName, patientName);
37. strcpy(temp->doctorName, doctorName);
38. strcpy(temp->surgeryDate, surgeryDate);
39. temp->next = NULL;
40. if (first == NULL) {
41. first = temp;
42. } else {
43. struct Node \*p = first;
44. while (p->next != NULL) {
45. p = p->next;
46. }
47. p->next = temp;
48. }
49. }
50. // Function to display all scheduled surgeries
51. void displaySurgeries() {
52. struct Node \*p = first;
53. while (p != NULL) {
54. printf("Surgery ID: %d, Patient: %s, Doctor: %s, Date: %s\n", p->surgeryId, p->patientName, p->doctorName, p->surgeryDate);
55. p = p->next;
56. }
57. }

**Problem 7: Patient History Record**

Description: Maintain a linked list to keep track of patient history records. Operations:

1. Create a history record list.
2. Insert a new record.
3. Display all patient history records.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for patient history records
8. struct Node {
9. int recordId;
10. char patientName[50];
11. char diagnosis[100];
12. char treatment[100];
13. struct Node \*next;
14. } \*first = NULL;
15. // Function prototypes
16. void createHistoryRecordList();
17. void insertHistoryRecord(int recordId, char patientName[], char diagnosis[], char treatment[]);
18. void displayHistoryRecords();
19. int main() {
20. createHistoryRecordList();
21. insertHistoryRecord(1, "Pavan", "Flu", "Rest and hydration");
22. insertHistoryRecord(2, "Noel", "Cold", "Rest and medication");
23. insertHistoryRecord(3, "kevin", "Back pain", "Physical therapy");
24. printf("Patient History Records:\n");
25. displayHistoryRecords();
26. return 0;
27. }
28. // Function to create a history record list
29. void createHistoryRecordList() {
30. first = NULL;
31. }
32. // Function to insert a new history record
33. void insertHistoryRecord(int recordId, char patientName[], char diagnosis[], char treatment[]) {
34. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
35. temp->recordId = recordId;
36. strcpy(temp->patientName, patientName);
37. strcpy(temp->diagnosis, diagnosis);
38. strcpy(temp->treatment, treatment);
39. temp->next = NULL;
40. if (first == NULL) {
41. first = temp;
42. } else {
43. struct Node \*p = first;
44. while (p->next != NULL) {
45. p = p->next;
46. }
47. p->next = temp;
48. }
49. }
50. // Function to display all patient history records
51. void displayHistoryRecords() {
52. struct Node \*p = first;
53. while (p != NULL) {
54. printf("Record ID: %d, Patient: %s, Diagnosis: %s, Treatment: %s\n", p->recordId, p->patientName, p->diagnosis, p->treatment);
55. p = p->next;
56. }
57. }

**Problem 8: Medical Test Tracking**

Description: Implement a linked list to track medical tests for patients. Operations:

1. Create a list of medical tests.
2. Insert a new test result.
3. Display all test results.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for medical tests
8. struct Node {
9. int testId;
10. char patientName[50];
11. char testName[50];
12. char testResult[100];
13. struct Node \*next;
14. } \*first = NULL;
15. // Function prototypes
16. void createTestList();
17. void insertTestResult(int testId, char patientName[], char testName[], char testResult[]);
18. void displayTestResults();
19. int main() {
20. createTestList();
21. insertTestResult(1, "Pavan", "Blood Test", "Normal");
22. insertTestResult(2, "Noel", "X-ray", "Clear");
23. insertTestResult(3, "Kevin", "ECG", "Abnormal");
24. printf("Medical Test Results:\n");
25. displayTestResults();
26. return 0;
27. }
28. // Function to create a medical test list
29. void createTestList() {
30. first = NULL;
31. }
32. // Function to insert a new test result
33. void insertTestResult(int testId, char patientName[], char testName[], char testResult[]) {
34. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
35. temp->testId = testId;
36. strcpy(temp->patientName, patientName);
37. strcpy(temp->testName, testName);
38. strcpy(temp->testResult, testResult);
39. temp->next = NULL;
40. if (first == NULL) {
41. first = temp;
42. } else {
43. struct Node \*p = first;
44. while (p->next != NULL) {
45. p = p->next;
46. }
47. p->next = temp;
48. }
49. }
50. // Function to display all medical test results
51. void displayTestResults() {
52. struct Node \*p = first;
53. while (p != NULL) {
54. printf("Test ID: %d, Patient: %s, Test: %s, Result: %s\n", p->testId, p->patientName, p->testName, p->testResult);
55. p = p->next;
56. }
57. }

**Problem 9: Prescription Management System**

Description: Use a linked list to manage patient prescriptions. Operations:

1. Create a prescription list.
2. Insert a new prescription.
3. Display all prescriptions.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for prescriptions
8. struct Node {
9. int prescriptionId;
10. char patientName[50];
11. char medication[50];
12. char dosage[50];
13. struct Node \*next;
14. } \*first = NULL;
15. // Function prototypes
16. void createPrescriptionList();
17. void insertPrescription(int prescriptionId, char patientName[], char medication[], char dosage[]);
18. void displayPrescriptions();
19. int main() {
20. createPrescriptionList();
21. insertPrescription(1, "Pavan", "Paracetamol", "500mg twice daily");
22. insertPrescription(2, "Rohit", "Ibuprofen", "200mg once daily");
23. insertPrescription(3, "nandan", "Amoxicillin", "250mg thrice daily");
24. printf("Prescription List:\n");
25. displayPrescriptions();
26. return 0;
27. }
28. // Function to create a prescription list
29. void createPrescriptionList() {
30. first = NULL;
31. }
32. // Function to insert a new prescription
33. void insertPrescription(int prescriptionId, char patientName[], char medication[], char dosage[]) {
34. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
35. temp->prescriptionId = prescriptionId;
36. strcpy(temp->patientName, patientName);
37. strcpy(temp->medication, medication);
38. strcpy(temp->dosage, dosage);
39. temp->next = NULL;
40. if (first == NULL) {
41. first = temp;
42. } else {
43. struct Node \*p = first;
44. while (p->next != NULL) {
45. p = p->next;
46. }
47. p->next = temp;
48. }
49. }
50. // Function to display all prescriptions
51. void displayPrescriptions() {
52. struct Node \*p = first;
53. while (p != NULL) {
54. printf("Prescription ID: %d, Patient: %s, Medication: %s, Dosage: %s\n", p->prescriptionId, p->patientName, p->medication, p->dosage);
55. p = p->next;
56. }
57. }

**Problem 10: Hospital Staff Roster**

Description: Develop a linked list to manage the hospital staff roster. Operations:

1. Create a staff roster.
2. Insert a new staff member into the roster.
3. Display the current staff roster.
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. // Define the Node structure for staff roster
8. struct Node {
9. int staffId;
10. char staffName[50];
11. char staffRole[50];
12. char contactNumber[15];
13. struct Node \*next;
14. } \*first = NULL;
15. // Function prototypes
16. void createStaffRoster();
17. void insertStaffMember(int staffId, char staffName[], char staffRole[], char contactNumber[]);
18. void displayStaffRoster();
19. int main() {
20. createStaffRoster();
21. insertStaffMember(1, "Dr. Rohit", "Surgeon", "1234567890");
22. insertStaffMember(2, "Nurse Priyanka", "Nurse", "9876543210");
23. insertStaffMember(3, "Dr. Suresh", "Pediatrician", "9841234567");
24. printf("Hospital Staff Roster:\n");
25. displayStaffRoster();
26. return 0;
27. }
28. // Function to create a staff roster
29. void createStaffRoster() {
30. first = NULL;
31. }
32. // Function to insert a new staff member into the roster
33. void insertStaffMember(int staffId, char staffName[], char staffRole[], char contactNumber[]) {
34. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
35. temp->staffId = staffId;
36. strcpy(temp->staffName, staffName);
37. strcpy(temp->staffRole, staffRole);
38. strcpy(temp->contactNumber, contactNumber);
39. temp->next = NULL;
40. if (first == NULL) {
41. first = temp;
42. } else {
43. struct Node \*p = first;
44. while (p->next != NULL) {
45. p = p->next;
46. }
47. p->next = temp;
48. }
49. }
50. // Function to display the current staff roster
51. void displayStaffRoster() {
52. struct Node \*p = first;
53. while (p != NULL) {
54. printf("Staff ID: %d, Name: %s, Role: %s, Contact: %s\n", p->staffId, p->staffName, p->staffRole, p->contactNumber);
55. p = p->next;
56. }
57. }