**Problem 1: Inventory Management System 16-01-2025**

1. Description: Implement a linked list to manage the inventory of raw materials.
2. Operations: Create an inventory list. Insert a new raw material.
3. Delete a raw material from the inventory. Display the current inventory.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node{
7. int itemId;
8. char itemName[50];
9. int quantity;
10. struct Node \*next;
11. }\*first = NULL;
12. void createInventory(int[], char[][50], int[]);
13. void display(struct Node \*);
14. void insertRawMaterial(struct Node \*, int, int, char[]);
15. int deleteRawMaterial(struct Node \*, int);
16. int main() {
17. int itemIds[] = {1, 2, 3};
18. char itemNames[][50] = {"Steel", "Wood", "Cement"};
19. int quantities[] = {100, 50, 200};
21. createInventory(itemIds, itemNames, quantities);
22. display(first);
23. insertRawMaterial(first, 4, 6, "Glass");
24. printf("\n");
25. display(first);
26. deleteRawMaterial(first, 2);
27. printf("\n");
28. display(first);
29. return 0;
30. }
31. void createInventory(int itemIds[], char itemNames[][50], int quantities[]) {
32. int i;
33. struct Node \*temp, \*last;
34. first = (struct Node \*)malloc(sizeof(struct Node));
35. first->itemId = itemIds[0];
36. for (i = 0; itemNames[0][i] != '\0'; i++) {
37. first->itemName[i] = itemNames[0][i];
38. }
39. first->itemName[i] = '\0';
40. first->quantity = quantities[0];
41. first->next = NULL;
42. last = first;
43. for(i = 1; i < 3; i++) {
44. temp = (struct Node \*)malloc(sizeof(struct Node));
45. temp->itemId = itemIds[i];
46. for (int j = 0; itemNames[i][j] != '\0'; j++) {
47. temp->itemName[j] = itemNames[i][j];
48. }
49. temp->itemName[strlen(itemNames[i])] = '\0';
50. temp->quantity = quantities[i];
51. temp->next = NULL;
52. last->next = temp;
53. last = temp;
54. }
55. }
56. void display(struct Node \*p) {
57. while(p != NULL) {
58. printf("Item ID: %d, Name: %s, Quantity: %d\n", p->itemId, p->itemName, p->quantity);
59. p = p->next;
60. }
61. }
62. void insertRawMaterial(struct Node \*p, int id, int quantity, char name[]) {
63. struct Node \*temp;
64. temp = (struct Node \*)malloc(sizeof(struct Node));
65. temp->itemId = id;
66. for (int i = 0; name[i] != '\0'; i++) {
67. temp->itemName[i] = name[i];
68. }
69. temp->itemName[strlen(name)] = '\0';
70. temp->quantity = quantity;
71. temp->next = NULL;
72. while(p->next != NULL) {
73. p = p->next;
74. }
75. p->next = temp;
76. }
77. int deleteRawMaterial(struct Node \*p, int itemId) {
78. struct Node \*q = NULL;
79. if(p != NULL && p->itemId == itemId) {
80. first = p->next;
81. free(p);
82. return itemId;
83. }
84. while(p != NULL && p->itemId != itemId) {
85. q = p;
86. p = p->next;
87. }
88. if(p == NULL) {
89. printf("Item with ID %d not found.\n", itemId);
90. return -1;
91. }
92. q->next = p->next;
93. free(p);
94. return itemId;
95. }

**Problem 2: Production Line Queue**

1. Description: Use a linked list to manage the queue of tasks on a production line.
2. Operations: Create a production task queue. Insert a new task into the queue.
3. Delete a completed task. Display the current task queue.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int taskId;
8. char taskName[50];
9. struct Node \*next;
10. } \*first = NULL;
11. void createQueue(int[], char[][50], int);
12. void display(struct Node \*);
13. void insertTask(struct Node \*, int, char[]);
14. int deleteTask(struct Node \*, int);
15. int main() {
16. int taskIds[] = {1, 2, 3};
17. char taskNames[][50] = {"Assemble", "Paint", "Quality Check"};
19. createQueue(taskIds, taskNames, 3);
20. display(first);
21. insertTask(first, 4, "Packaging");
22. printf("\nUpdated Queue:\n");
23. display(first);
24. deleteTask(first, 2);
25. printf("\nQueue after deleting task with ID 2:\n");
26. display(first);
27. return 0;
28. }
29. // Function to create a task queue
30. void createQueue(int taskIds[], char taskNames[][50], int n) {
31. int i;
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->taskId = taskIds[0];
35. for (i = 0; taskNames[0][i] != '\0'; i++) {
36. first->taskName[i] = taskNames[0][i];
37. }
38. first->taskName[i] = '\0';
39. first->next = NULL;
40. last = first;
41. for(i = 1; i < n; i++) {
42. temp = (struct Node \*)malloc(sizeof(struct Node));
43. temp->taskId = taskIds[i];
44. for (int j = 0; taskNames[i][j] != '\0'; j++) {
45. temp->taskName[j] = taskNames[i][j];
46. }
47. temp->taskName[strlen(taskNames[i])] = '\0';
48. temp->next = NULL;
49. last->next = temp;
50. last = temp;
51. }
52. }
53. void display(struct Node \*p) {
54. while(p != NULL) {
55. printf("Task ID: %d, Name: %s\n", p->taskId, p->taskName);
56. p = p->next;
57. }
58. }
59. void insertTask(struct Node \*p, int id, char name[]) {
60. struct Node \*temp;
61. temp = (struct Node \*)malloc(sizeof(struct Node));
62. temp->taskId = id;
63. for (int i = 0; name[i] != '\0'; i++) {
64. temp->taskName[i] = name[i];
65. }
66. temp->taskName[strlen(name)] = '\0';
67. temp->next = NULL;
68. while(p->next != NULL) {
69. p = p->next;
70. }
71. p->next = temp;
72. }
73. int deleteTask(struct Node \*p, int taskId) {
74. struct Node \*q = NULL;
75. if(p != NULL && p->taskId == taskId) {
76. first = p->next;
77. free(p);
78. return taskId;
79. }
80. while(p != NULL && p->taskId != taskId) {
81. q = p;
82. p = p->next;
83. }
84. if(p == NULL) {
85. printf("Task with ID %d not found.\n", taskId);
86. return -1;
87. }
88. q->next = p->next;
89. free(p);
90. return taskId;
91. }

**Problem 3: Machine Maintenance Schedule**

1. Description: Develop a linked list to manage the maintenance schedule of machines.
2. Operations: Create a maintenance schedule. Insert a new maintenance task.
3. Delete a completed maintenance task. Display the maintenance schedule.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int taskId;
8. char machineName[50];
9. char maintenanceTask[50];
10. struct Node \*next;
11. } \*first = NULL;
12. void createSchedule(int[], char[][50], char[][50], int);
13. void display(struct Node \*);
14. void insertMaintenanceTask(struct Node \*, int, char[], char[]);
15. int deleteMaintenanceTask(struct Node \*, int);
16. int main() {
17. int taskIds[] = {1, 2, 3};
18. char machineNames[][50] = {"Machine A", "Machine B", "Machine C"};
19. char maintenanceTasks[][50] = {"Oil Change", "Belt Replacement", "Filter Cleaning"};
21. createSchedule(taskIds, machineNames, maintenanceTasks, 3);
22. display(first);
23. insertMaintenanceTask(first, 4, "Machine D", "Engine Check");
24. printf("\nUpdated Schedule:\n");
25. display(first);
26. deleteMaintenanceTask(first, 2);
27. printf("\nSchedule after deleting task with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createSchedule(int taskIds[], char machineNames[][50], char maintenanceTasks[][50], int n) {
32. int i;
33. struct Node \*temp, \*last;
34. first = (struct Node \*)malloc(sizeof(struct Node));
35. first->taskId = taskIds[0];
36. for (i = 0; machineNames[0][i] != '\0'; i++) {
37. first->machineName[i] = machineNames[0][i];
38. }
39. first->machineName[i] = '\0';
40. for (i = 0; maintenanceTasks[0][i] != '\0'; i++) {
41. first->maintenanceTask[i] = maintenanceTasks[0][i];
42. }
43. first->maintenanceTask[i] = '\0';
44. first->next = NULL;
45. last = first;
46. for(i = 1; i < n; i++) {
47. temp = (struct Node \*)malloc(sizeof(struct Node));
48. temp->taskId = taskIds[i];
49. for (int j = 0; machineNames[i][j] != '\0'; j++) {
50. temp->machineName[j] = machineNames[i][j];
51. }
52. temp->machineName[strlen(machineNames[i])] = '\0';
53. for (int k = 0; maintenanceTasks[i][k] != '\0'; k++) {
54. temp->maintenanceTask[k] = maintenanceTasks[i][k];
55. }
56. temp->maintenanceTask[strlen(maintenanceTasks[i])] = '\0';
57. temp->next = NULL;
58. last->next = temp;
59. last = temp;
60. }
61. }
62. void display(struct Node \*p) {
63. while(p != NULL) {
64. printf("Task ID: %d, Machine: %s, Maintenance Task: %s\n", p->taskId, p->machineName, p->maintenanceTask);
65. p = p->next;
66. }
67. }
68. void insertMaintenanceTask(struct Node \*p, int id, char machine[], char task[]) {
69. struct Node \*temp;
70. temp = (struct Node \*)malloc(sizeof(struct Node));
71. temp->taskId = id;
72. for (int i = 0; machine[i] != '\0'; i++) {
73. temp->machineName[i] = machine[i];
74. }
75. temp->machineName[strlen(machine)] = '\0';
76. for (int i = 0; task[i] != '\0'; i++) {
77. temp->maintenanceTask[i] = task[i];
78. }
79. temp->maintenanceTask[strlen(task)] = '\0';
80. temp->next = NULL;
81. while(p->next != NULL) {
82. p = p->next;
83. }
84. p->next = temp;
85. }
86. int deleteMaintenanceTask(struct Node \*p, int taskId) {
87. struct Node \*q = NULL;
88. if(p != NULL && p->taskId == taskId) {
89. first = p->next;
90. free(p);
91. return taskId;
92. }
93. while(p != NULL && p->taskId != taskId) {
94. q = p;
95. p = p->next;
96. }
97. if(p == NULL) {
98. printf("Task with ID %d not found.\n", taskId);
99. return -1;
100. }
101. q->next = p->next;
102. free(p);
103. return taskId;
104. }

**Problem 4: Employee Shift Management**

1. Description: Use a linked list to manage employee shifts in a manufacturing plant.
2. Operations: Create a shift schedule. Insert a new shift.
3. Delete a completed or canceled shift. Display the current shift schedule.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int shiftId;
8. char employeeName[50];
9. char shiftTime[20];
10. struct Node \*next;
11. } \*first = NULL;
12. void createShiftSchedule(int[], char[][50], char[][20], int);
13. void display(struct Node \*);
14. void insertShift(int, char[], char[]);
15. int deleteShift(int);
16. int main() {
17. int shiftIds[] = {1, 2, 3};
18. char employeeNames[][50] = {"Pavan", "Sanket", "Rahul"};
19. char shiftTimes[][20] = {"9 AM - 5 PM", "5 PM - 1 AM", "1 AM - 9 AM"};
21. createShiftSchedule(shiftIds, employeeNames, shiftTimes, 3);
22. display(first);
23. insertShift(4, "Michael Brown", "9 AM - 5 PM");
24. printf("\nUpdated Shift Schedule:\n");
25. display(first);
26. deleteShift(2);
27. printf("\nShift Schedule after deleting shift with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createShiftSchedule(int shiftIds[], char employeeNames[][50], char shiftTimes[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->shiftId = shiftIds[0];
35. strcpy(first->employeeName, employeeNames[0]);
36. strcpy(first->shiftTime, shiftTimes[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->shiftId = shiftIds[i];
42. strcpy(temp->employeeName, employeeNames[i]);
43. strcpy(temp->shiftTime, shiftTimes[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Shift ID: %d, Employee: %s, Shift Time: %s\n", p->shiftId, p->employeeName, p->shiftTime);
52. p = p->next;
53. }
54. }
55. void insertShift(int id, char name[], char time[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->shiftId = id;
58. strcpy(temp->employeeName, name);
59. strcpy(temp->shiftTime, time);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteShift(int shiftId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->shiftId == shiftId) {
70. first = p->next;
71. free(p);
72. return shiftId;
73. }
74. while(p != NULL && p->shiftId != shiftId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Shift with ID %d not found.\n", shiftId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return shiftId;
85. }

**5: Order Processing System**

1. Description: Implement a linked list to track customer orders.
2. Operations: Create an order list. Insert a new customer order.
3. Delete a completed or canceled order. Display all current orders
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int orderId;
8. char customerName[50];
9. char orderDetails[100];
10. struct Node \*next;
11. } \*first = NULL;
12. void createOrderList(int[], char[][50], char[][100], int);
13. void display(struct Node \*);
14. void insertOrder(int, char[], char[]);
15. int deleteOrder(int);
16. int main() {
17. int orderIds[] = {101, 102, 103};
18. char customerNames[][50] = {"Pavan", "Sanket", "Rahul"};
19. char orderDetails[][100] = {"Laptop", "Smartphone", "Tablet"};
21. createOrderList(orderIds, customerNames, orderDetails, 3);
22. display(first);
23. insertOrder(104, "David", "Smartwatch");
24. printf("\nUpdated Order List:\n");
25. display(first);
26. deleteOrder(102);
27. printf("\nOrder List after deleting order with ID 102:\n");
28. display(first);
29. return 0;
30. }
31. void createOrderList(int orderIds[], char customerNames[][50], char orderDetails[][100], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->orderId = orderIds[0];
35. strcpy(first->customerName, customerNames[0]);
36. strcpy(first->orderDetails, orderDetails[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->orderId = orderIds[i];
42. strcpy(temp->customerName, customerNames[i]);
43. strcpy(temp->orderDetails, orderDetails[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Order ID: %d, Customer: %s, Order: %s\n", p->orderId, p->customerName, p->orderDetails);
52. p = p->next;
53. }
54. }
55. void insertOrder(int id, char name[], char details[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->orderId = id;
58. strcpy(temp->customerName, name);
59. strcpy(temp->orderDetails, details);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteOrder(int orderId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->orderId == orderId) {
70. first = p->next;
71. free(p);
72. return orderId;
73. }
74. while(p != NULL && p->orderId != orderId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Order with ID %d not found.\n", orderId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return orderId;
85. }

**Problem 6: Tool Tracking System**

1. Description: Maintain a linked list to track tools used in the manufacturing process. Operations: Create a tool tracking list. Insert a new tool entry.
2. Delete a tool that is no longer in use. Display all tools currently tracked.
3. #include <stdio.h>
4. #include <stdlib.h>
5. struct Node {
6. int toolId;
7. char toolName[50];
8. char toolStatus[20]; // E.g., "In Use" or "Not In Use"
9. struct Node \*next;
10. } \*first = NULL;
11. void createToolTrackingList(int[], char[][50], char[][20], int);
12. void display(struct Node \*);
13. void insertTool(int, char[], char[]);
14. int deleteTool(int);
15. int main() {
16. int toolIds[] = {201, 202, 203};
17. char toolNames[][50] = {"Hammer", "Screwdriver", "Wrench"};
18. char toolStatuses[][20] = {"In Use", "Not In Use", "In Use"};
20. createToolTrackingList(toolIds, toolNames, toolStatuses, 3);
21. display(first);
22. insertTool(204, "Drill", "In Use");
23. printf("\nUpdated Tool Tracking List:\n");
24. display(first);
25. deleteTool(202);
26. printf("\nTool Tracking List after deleting tool with ID 202:\n");
27. display(first);
28. return 0;
29. }
30. void createToolTrackingList(int toolIds[], char toolNames[][50], char toolStatuses[][20], int n) {
31. struct Node \*temp, \*last;
32. first = (struct Node \*)malloc(sizeof(struct Node));
33. first->toolId = toolIds[0];
34. strcpy(first->toolName, toolNames[0]);
35. strcpy(first->toolStatus, toolStatuses[0]);
36. first->next = NULL;
37. last = first;
38. for(int i = 1; i < n; i++) {
39. temp = (struct Node \*)malloc(sizeof(struct Node));
40. temp->toolId = toolIds[i];
41. strcpy(temp->toolName, toolNames[i]);
42. strcpy(temp->toolStatus, toolStatuses[i]);
43. temp->next = NULL;
44. last->next = temp;
45. last = temp;
46. }
47. }
48. void display(struct Node \*p) {
49. while(p != NULL) {
50. printf("Tool ID: %d, Tool Name: %s, Status: %s\n", p->toolId, p->toolName, p->toolStatus);
51. p = p->next;
52. }
53. }
54. void insertTool(int id, char name[], char status[]) {
55. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
56. temp->toolId = id;
57. strcpy(temp->toolName, name);
58. strcpy(temp->toolStatus, status);
59. temp->next = NULL;
60. struct Node \*p = first;
61. while(p->next != NULL) {
62. p = p->next;
63. }
64. p->next = temp;
65. }
66. int deleteTool(int toolId) {
67. struct Node \*p = first, \*q = NULL;
68. if(p != NULL && p->toolId == toolId) {
69. first = p->next;
70. free(p);
71. return toolId;
72. }
73. while(p != NULL && p->toolId != toolId) {
74. q = p;
75. p = p->next;
76. }
77. if(p == NULL) {
78. printf("Tool with ID %d not found.\n", toolId);
79. return -1;
80. }
81. q->next = p->next;
82. free(p);
83. return toolId;
84. }

**Problem 7: Product Assembly Line**

1. Description: Use a linked list to manage the assembly stages of a product.
2. Operations: Create an assembly line stage list. Insert a new stage.
3. Delete a completed stage. Display the current assembly stages.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int stageId;
8. char stageName[50];
9. char stageStatus[20]; // E.g., "In Progress", "Completed"
10. struct Node \*next;
11. } \*first = NULL;
12. void createAssemblyLine(int[], char[][50], char[][20], int);
13. void display(struct Node \*);
14. void insertStage(int, char[], char[]);
15. int deleteStage(int);
16. int main() {
17. int stageIds[] = {1, 2, 3};
18. char stageNames[][50] = {"Design", "Manufacturing", "Packaging"};
19. char stageStatuses[][20] = {"Completed", "In Progress", "Not Started"};
21. createAssemblyLine(stageIds, stageNames, stageStatuses, 3);
22. display(first);
23. insertStage(4, "Quality Check", "Not Started");
24. printf("\nUpdated Assembly Line Stages:\n");
25. display(first);
26. deleteStage(2);
27. printf("\nAssembly Line after deleting stage with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createAssemblyLine(int stageIds[], char stageNames[][50], char stageStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->stageId = stageIds[0];
35. strcpy(first->stageName, stageNames[0]);
36. strcpy(first->stageStatus, stageStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->stageId = stageIds[i];
42. strcpy(temp->stageName, stageNames[i]);
43. strcpy(temp->stageStatus, stageStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Stage ID: %d, Stage Name: %s, Status: %s\n", p->stageId, p->stageName, p->stageStatus);
52. p = p->next;
53. }
54. }
55. void insertStage(int id, char name[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->stageId = id;
58. strcpy(temp->stageName, name);
59. strcpy(temp->stageStatus, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteStage(int stageId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->stageId == stageId) {
70. first = p->next;
71. free(p);
72. return stageId;
73. }
74. while(p != NULL && p->stageId != stageId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Stage with ID %d not found.\n", stageId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return stageId;
85. }

**Problem 8: Quality Control Checklist**

1. Description: Implement a linked list to manage a quality control checklist.
2. Operations: Create a quality control checklist. Insert a new checklist item.
3. Delete a completed or outdated checklist item. Display the current quality control checklist.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int itemId;
8. char itemName[50];
9. char itemStatus[20]; // E.g., "Completed", "Pending"
10. struct Node \*next;
11. } \*first = NULL;
12. void createChecklist(int[], char[][50], char[][20], int);
13. void display(struct Node \*);
14. void insertChecklistItem(int, char[], char[]);
15. int deleteChecklistItem(int);
16. int main() {
17. int itemIds[] = {1, 2, 3};
18. char itemNames[][50] = {"Check Product Dimensions", "Test Product Functionality", "Inspect Product Surface"};
19. char itemStatuses[][20] = {"Completed", "Pending", "Pending"};
21. createChecklist(itemIds, itemNames, itemStatuses, 3);
22. display(first);
23. insertChecklistItem(4, "Verify Packaging", "Pending");
24. printf("\nUpdated Quality Control Checklist:\n");
25. display(first);
26. deleteChecklistItem(2);
27. printf("\nQuality Control Checklist after deleting item with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createChecklist(int itemIds[], char itemNames[][50], char itemStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->itemId = itemIds[0];
35. strcpy(first->itemName, itemNames[0]);
36. strcpy(first->itemStatus, itemStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->itemId = itemIds[i];
42. strcpy(temp->itemName, itemNames[i]);
43. strcpy(temp->itemStatus, itemStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Item ID: %d, Item Name: %s, Status: %s\n", p->itemId, p->itemName, p->itemStatus);
52. p = p->next;
53. }
54. }
55. void insertChecklistItem(int id, char name[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->itemId = id;
58. strcpy(temp->itemName, name);
59. strcpy(temp->itemStatus, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteChecklistItem(int itemId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->itemId == itemId) {
70. first = p->next;
71. free(p);
72. return itemId;
73. }
74. while(p != NULL && p->itemId != itemId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Item with ID %d not found.\n", itemId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return itemId;
85. }

**Problem 9: Supplier Management System**

1. Description: Use a linked list to manage a list of suppliers.
2. Operations: Create a supplier list. Insert a new supplier.
3. Delete an inactive or outdated supplier. Display all current suppliers.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int supplierId;
8. char supplierName[50];
9. char supplierStatus[20]; // E.g., "Active", "Inactive"
10. struct Node \*next;
11. } \*first = NULL;
12. void createSupplierList(int[], char[][50], char[][20], int);
13. void display(struct Node \*);
14. void insertSupplier(int, char[], char[]);
15. int deleteSupplier(int);
16. int main() {
17. int supplierIds[] = {101, 102, 103};
18. char supplierNames[][50] = {"Supplier A", "Supplier B", "Supplier C"};
19. char supplierStatuses[][20] = {"Active", "Inactive", "Active"};
21. createSupplierList(supplierIds, supplierNames, supplierStatuses, 3);
22. display(first);
23. insertSupplier(104, "Supplier D", "Active");
24. printf("\nUpdated Supplier List:\n");
25. display(first);
26. deleteSupplier(102);
27. printf("\nSupplier List after deleting supplier with ID 102:\n");
28. display(first);
29. return 0;
30. }
31. void createSupplierList(int supplierIds[], char supplierNames[][50], char supplierStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->supplierId = supplierIds[0];
35. strcpy(first->supplierName, supplierNames[0]);
36. strcpy(first->supplierStatus, supplierStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->supplierId = supplierIds[i];
42. strcpy(temp->supplierName, supplierNames[i]);
43. strcpy(temp->supplierStatus, supplierStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Supplier ID: %d, Supplier Name: %s, Status: %s\n", p->supplierId, p->supplierName, p->supplierStatus);
52. p = p->next;
53. }
54. }
55. void insertSupplier(int id, char name[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->supplierId = id;
58. strcpy(temp->supplierName, name);
59. strcpy(temp->supplierStatus, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteSupplier(int supplierId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->supplierId == supplierId) {
70. first = p->next;
71. free(p);
72. return supplierId;
73. }
74. while(p != NULL && p->supplierId != supplierId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Supplier with ID %d not found.\n", supplierId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return supplierId;
85. }

**Problem 10: Manufacturing Project Timeline**

1. Description: Develop a linked list to manage the timeline of a manufacturing project.
2. Operations: Create a project timeline. Insert a new project milestone.
3. Delete a completed milestone. Display the current project timeline.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int milestoneId;
8. char milestoneName[50];
9. char milestoneStatus[20]; // E.g., "Completed", "Pending"
10. struct Node \*next;
11. } \*first = NULL;
12. void createProjectTimeline(int[], char[][50], char[][20], int);
13. void display(struct Node \*);
14. void insertMilestone(int, char[], char[]);
15. int deleteMilestone(int);
16. int main() {
17. int milestoneIds[] = {1, 2, 3};
18. char milestoneNames[][50] = {"Design Phase", "Production Phase", "Testing Phase"};
19. char milestoneStatuses[][20] = {"Completed", "Pending", "Pending"};
21. createProjectTimeline(milestoneIds, milestoneNames, milestoneStatuses, 3);
22. display(first);
23. insertMilestone(4, "Delivery Phase", "Pending");
24. printf("\nUpdated Project Timeline:\n");
25. display(first);
26. deleteMilestone(2);
27. printf("\nProject Timeline after deleting milestone with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createProjectTimeline(int milestoneIds[], char milestoneNames[][50], char milestoneStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->milestoneId = milestoneIds[0];
35. strcpy(first->milestoneName, milestoneNames[0]);
36. strcpy(first->milestoneStatus, milestoneStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->milestoneId = milestoneIds[i];
42. strcpy(temp->milestoneName, milestoneNames[i]);
43. strcpy(temp->milestoneStatus, milestoneStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Milestone ID: %d, Milestone Name: %s, Status: %s\n", p->milestoneId, p->milestoneName, p->milestoneStatus);
52. p = p->next;
53. }
54. }
55. void insertMilestone(int id, char name[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->milestoneId = id;
58. strcpy(temp->milestoneName, name);
59. strcpy(temp->milestoneStatus, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteMilestone(int milestoneId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->milestoneId == milestoneId) {
70. first = p->next;
71. free(p);
72. return milestoneId;
73. }
74. while(p != NULL && p->milestoneId != milestoneId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Milestone with ID %d not found.\n", milestoneId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return milestoneId;
85. }

**Problem 11: Warehouse Storage Management**

1. Description: Implement a linked list to manage the storage of goods in a warehouse.
2. Operations: Create a storage list. Insert a new storage entry.
3. Delete a storage entry when goods are shipped. Display the current warehouse storage.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int itemId;
8. char itemName[50];
9. int quantity;
10. struct Node \*next;
11. } \*first = NULL;
12. void createStorageList(int[], char[][50], int[], int);
13. void display(struct Node \*);
14. void insertStorageEntry(int, char[], int);
15. int deleteStorageEntry(int);
16. int main() {
17. int itemIds[] = {101, 102, 103};
18. char itemNames[][50] = {"Item A", "Item B", "Item C"};
19. int quantities[] = {50, 30, 100};
21. createStorageList(itemIds, itemNames, quantities, 3);
22. display(first);
23. insertStorageEntry(104, "Item D", 200);
24. printf("\nUpdated Warehouse Storage:\n");
25. display(first);
26. deleteStorageEntry(102);
27. printf("\nWarehouse Storage after deleting item with ID 102:\n");
28. display(first);
29. return 0;
30. }
31. void createStorageList(int itemIds[], char itemNames[][50], int quantities[], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->itemId = itemIds[0];
35. strcpy(first->itemName, itemNames[0]);
36. first->quantity = quantities[0];
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->itemId = itemIds[i];
42. strcpy(temp->itemName, itemNames[i]);
43. temp->quantity = quantities[i];
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Item ID: %d, Item Name: %s, Quantity: %d\n", p->itemId, p->itemName, p->quantity);
52. p = p->next;
53. }
54. }
55. void insertStorageEntry(int id, char name[], int quantity) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->itemId = id;
58. strcpy(temp->itemName, name);
59. temp->quantity = quantity;
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteStorageEntry(int itemId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->itemId == itemId) {
70. first = p->next;
71. free(p);
72. return itemId;
73. }
74. while(p != NULL && p->itemId != itemId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Item with ID %d not found.\n", itemId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return itemId;
85. }

**Problem 12: Machine Parts Inventory**

1. Description: Use a linked list to track machine parts inventory.
2. Operations: Create a parts inventory list. Insert a new part.
3. Delete a part that is used up or obsolete. Display the current parts inventory
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int partId;
8. char partName[50];
9. int quantity;
10. struct Node \*next;
11. } \*first = NULL;
12. void createPartsInventoryList(int[], char[][50], int[], int);
13. void display(struct Node \*);
14. void insertPart(int, char[], int);
15. int deletePart(int);
16. int main() {
17. int partIds[] = {1001, 1002, 1003};
18. char partNames[][50] = {"Gear", "Bolt", "Screw"};
19. int quantities[] = {10, 20, 50};
21. createPartsInventoryList(partIds, partNames, quantities, 3);
22. display(first);
23. insertPart(1004, "Washer", 30);
24. printf("\nUpdated Parts Inventory:\n");
25. display(first);
26. deletePart(1002);
27. printf("\nParts Inventory after deleting part with ID 1002:\n");
28. display(first);
29. return 0;
30. }
31. void createPartsInventoryList(int partIds[], char partNames[][50], int quantities[], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->partId = partIds[0];
35. strcpy(first->partName, partNames[0]);
36. first->quantity = quantities[0];
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->partId = partIds[i];
42. strcpy(temp->partName, partNames[i]);
43. temp->quantity = quantities[i];
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Part ID: %d, Part Name: %s, Quantity: %d\n", p->partId, p->partName, p->quantity);
52. p = p->next;
53. }
54. }
55. void insertPart(int id, char name[], int quantity) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->partId = id;
58. strcpy(temp->partName, name);
59. temp->quantity = quantity;
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deletePart(int partId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->partId == partId) {
70. first = p->next;
71. free(p);
72. return partId;
73. }
74. while(p != NULL && p->partId != partId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Part with ID %d not found.\n", partId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return partId;
85. }

**Problem 13: Packaging Line Schedule**

1. Description: Manage the schedule of packaging tasks using a linked list.
2. Operations: Create a packaging task schedule. Insert a new packaging task.
3. Delete a completed packaging task. Display the current packaging schedule.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int taskId;
8. char taskName[50];
9. char taskStatus[20];
10. struct Node \*next;
11. } \*first = NULL;
12. void createPackagingSchedule(int[], char[][50], char[][20], int);
13. void display(struct Node \*);
14. void insertPackagingTask(int, char[], char[]);
15. int deletePackagingTask(int);
16. int main() {
17. int taskIds[] = {1, 2, 3};
18. char taskNames[][50] = {"Package Product A", "Package Product B", "Package Product C"};
19. char taskStatuses[][20] = {"Pending", "In Progress", "Completed"};
21. createPackagingSchedule(taskIds, taskNames, taskStatuses, 3);
22. display(first);
23. insertPackagingTask(4, "Package Product D", "Pending");
24. printf("\nUpdated Packaging Schedule:\n");
25. display(first);
26. deletePackagingTask(2);
27. printf("\nPackaging Schedule after deleting task with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createPackagingSchedule(int taskIds[], char taskNames[][50], char taskStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->taskId = taskIds[0];
35. strcpy(first->taskName, taskNames[0]);
36. strcpy(first->taskStatus, taskStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->taskId = taskIds[i];
42. strcpy(temp->taskName, taskNames[i]);
43. strcpy(temp->taskStatus, taskStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Task ID: %d, Task Name: %s, Status: %s\n", p->taskId, p->taskName, p->taskStatus);
52. p = p->next;
53. }
54. }
55. void insertPackagingTask(int id, char name[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->taskId = id;
58. strcpy(temp->taskName, name);
59. strcpy(temp->taskStatus, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deletePackagingTask(int taskId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->taskId == taskId) {
70. first = p->next;
71. free(p);
72. return taskId;
73. }
74. while(p != NULL && p->taskId != taskId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Task with ID %d not found.\n", taskId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return taskId;
85. }

**Problem 14: Production Defect Tracking**

1. Description: Implement a linked list to track defects in the production process.
2. Operations: Create a defect tracking list. Insert a new defect report.
3. Delete a resolved defect. Display all current defects.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int defectId;
8. char defectDescription[100];
9. char status[20];
10. struct Node \*next;
11. } \*first = NULL;
12. void createDefectTrackingList(int[], char[][100], char[][20], int);
13. void display(struct Node \*);
14. void insertDefect(int, char[], char[]);
15. int deleteDefect(int);
16. int main() {
17. int defectIds[] = {101, 102, 103};
18. char defectDescriptions[][100] = {"Cracked Part", "Broken Sensor", "Misaligned Machine"};
19. char defectStatuses[][20] = {"Unresolved", "In Progress", "Resolved"};
21. createDefectTrackingList(defectIds, defectDescriptions, defectStatuses, 3);
22. display(first);
23. insertDefect(104, "Faulty Wiring", "Unresolved");
24. printf("\nUpdated Defect Tracking List:\n");
25. display(first);
26. deleteDefect(102);
27. printf("\nDefect Tracking List after deleting defect with ID 102:\n");
28. display(first);
29. return 0;
30. }
31. void createDefectTrackingList(int defectIds[], char defectDescriptions[][100], char defectStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->defectId = defectIds[0];
35. strcpy(first->defectDescription, defectDescriptions[0]);
36. strcpy(first->status, defectStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->defectId = defectIds[i];
42. strcpy(temp->defectDescription, defectDescriptions[i]);
43. strcpy(temp->status, defectStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Defect ID: %d, Description: %s, Status: %s\n", p->defectId, p->defectDescription, p->status);
52. p = p->next;
53. }
54. }
55. void insertDefect(int id, char description[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->defectId = id;
58. strcpy(temp->defectDescription, description);
59. strcpy(temp->status, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteDefect(int defectId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->defectId == defectId) {
70. first = p->next;
71. free(p);
72. return defectId;
73. }
74. while(p != NULL && p->defectId != defectId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Defect with ID %d not found.\n", defectId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return defectId;
85. }

**Problem 15: Finished Goods Dispatch System**

1. Description: Use a linked list to manage the dispatch schedule of finished goods.
2. Operations: Create a dispatch schedule. Insert a new dispatch entry.
3. Delete a dispatched or canceled entry. Display the current dispatch schedule.
4. #include <stdio.h>
5. #include <stdlib.h>
6. struct Node {
7. int dispatchId;
8. char goodsName[100];
9. char dispatchStatus[20];
10. struct Node \*next;
11. } \*first = NULL;
12. void createDispatchSchedule(int[], char[][100], char[][20], int);
13. void display(struct Node \*);
14. void insertDispatch(int, char[], char[]);
15. int deleteDispatch(int);
16. int main() {
17. int dispatchIds[] = {1, 2, 3};
18. char goodsNames[][100] = {"Finished Product A", "Finished Product B", "Finished Product C"};
19. char dispatchStatuses[][20] = {"Pending", "Dispatched", "Cancelled"};
21. createDispatchSchedule(dispatchIds, goodsNames, dispatchStatuses, 3);
22. display(first);
23. insertDispatch(4, "Finished Product D", "Pending");
24. printf("\nUpdated Dispatch Schedule:\n");
25. display(first);
26. deleteDispatch(2);
27. printf("\nDispatch Schedule after deleting dispatch entry with ID 2:\n");
28. display(first);
29. return 0;
30. }
31. void createDispatchSchedule(int dispatchIds[], char goodsNames[][100], char dispatchStatuses[][20], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->dispatchId = dispatchIds[0];
35. strcpy(first->goodsName, goodsNames[0]);
36. strcpy(first->dispatchStatus, dispatchStatuses[0]);
37. first->next = NULL;
38. last = first;
39. for(int i = 1; i < n; i++) {
40. temp = (struct Node \*)malloc(sizeof(struct Node));
41. temp->dispatchId = dispatchIds[i];
42. strcpy(temp->goodsName, goodsNames[i]);
43. strcpy(temp->dispatchStatus, dispatchStatuses[i]);
44. temp->next = NULL;
45. last->next = temp;
46. last = temp;
47. }
48. }
49. void display(struct Node \*p) {
50. while(p != NULL) {
51. printf("Dispatch ID: %d, Goods Name: %s, Status: %s\n", p->dispatchId, p->goodsName, p->dispatchStatus);
52. p = p->next;
53. }
54. }
55. void insertDispatch(int id, char name[], char status[]) {
56. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
57. temp->dispatchId = id;
58. strcpy(temp->goodsName, name);
59. strcpy(temp->dispatchStatus, status);
60. temp->next = NULL;
61. struct Node \*p = first;
62. while(p->next != NULL) {
63. p = p->next;
64. }
65. p->next = temp;
66. }
67. int deleteDispatch(int dispatchId) {
68. struct Node \*p = first, \*q = NULL;
69. if(p != NULL && p->dispatchId == dispatchId) {
70. first = p->next;
71. free(p);
72. return dispatchId;
73. }
74. while(p != NULL && p->dispatchId != dispatchId) {
75. q = p;
76. p = p->next;
77. }
78. if(p == NULL) {
79. printf("Dispatch entry with ID %d not found.\n", dispatchId);
80. return -1;
81. }
82. q->next = p->next;
83. free(p);
84. return dispatchId;
85. }

**Problem 1: Team Roster Management**

**Description:** Implement a linked list to manage the roster of players in a sports team.

**Operations:**

1. Create a team roster.
2. Insert a new player.
3. Delete a player who leaves the team.
4. Display the current team roster.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int playerId;
9. char playerName[100];
10. struct Node \*next;
11. } \*first = NULL;
12. void createTeamRoster(int[], char[][100], int);
13. void display(struct Node \*);
14. void insertPlayer(int, char[]);
15. int deletePlayer(int);
16. int main() {
17. int playerIds[] = {1, 2, 3};
18. char playerNames[][100] = {"Player A", "Player B", "Player C"};
19. createTeamRoster(playerIds, playerNames, 3);
20. display(first);
21. insertPlayer(4, "Player D");
22. printf("\nUpdated Team Roster:\n");
23. display(first);
24. deletePlayer(2);
25. printf("\nTeam Roster after removing player with ID 2:\n");
26. display(first);
27. return 0;
28. }
29. void createTeamRoster(int playerIds[], char playerNames[][100], int n) {
30. struct Node \*temp, \*last;
31. first = (struct Node \*)malloc(sizeof(struct Node));
32. first->playerId = playerIds[0];
33. strcpy(first->playerName, playerNames[0]);
34. first->next = NULL;
35. last = first;
36. for (int i = 1; i < n; i++) {
37. temp = (struct Node \*)malloc(sizeof(struct Node));
38. temp->playerId = playerIds[i];
39. strcpy(temp->playerName, playerNames[i]);
40. temp->next = NULL;
41. last->next = temp;
42. last = temp;
43. }
44. }
45. void display(struct Node \*p) {
46. while (p != NULL) {
47. printf("Player ID: %d, Player Name: %s\n", p->playerId, p->playerName);
48. p = p->next;
49. }
50. }
51. void insertPlayer(int id, char name[]) {
52. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
53. temp->playerId = id;
54. strcpy(temp->playerName, name);
55. temp->next = NULL;
56. struct Node \*p = first;
57. while (p->next != NULL) {
58. p = p->next;
59. }
60. p->next = temp;
61. }
62. int deletePlayer(int playerId) {
63. struct Node \*p = first, \*q = NULL;
64. if (p != NULL && p->playerId == playerId) {
65. first = p->next;
66. free(p);
67. return playerId;
68. }
69. while (p != NULL && p->playerId != playerId) {
70. q = p;
71. p = p->next;
72. }
73. if (p == NULL) {
74. printf("Player with ID %d not found.\n", playerId);
75. return -1;
76. }
77. q->next = p->next;
78. free(p);
79. return playerId;
80. }

**Problem 2: Tournament Match Scheduling**

**Description:** Use a linked list to schedule matches in a tournament.

**Operations:**

1. Create a match schedule.
2. Insert a new match.
3. Delete a completed or canceled match.
4. Display the current match schedule.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int matchId;
9. char matchDetails[100];
10. struct Node \*next;
11. } \*first = NULL;
12. void createMatchSchedule(int[], char[][100], int);
13. void display(struct Node \*);
14. void insertMatch(int, char[]);
15. int deleteMatch(int);
16. int main() {
17. int matchIds[] = {101, 102, 103};
18. char matchDetails[][100] = {"Team A vs Team B", "Team C vs Team D", "Team E vs Team F"};
19. createMatchSchedule(matchIds, matchDetails, 3);
20. display(first);
21. insertMatch(104, "Team G vs Team H");
22. printf("\nUpdated Match Schedule:\n");
23. display(first);
24. deleteMatch(102);
25. printf("\nMatch Schedule after removing match with ID 102:\n");
26. display(first);
27. return 0;
28. }
29. void createMatchSchedule(int matchIds[], char matchDetails[][100], int n) {
30. struct Node \*temp, \*last;
31. first = (struct Node \*)malloc(sizeof(struct Node));
32. first->matchId = matchIds[0];
33. strcpy(first->matchDetails, matchDetails[0]);
34. first->next = NULL;
35. last = first;
36. for (int i = 1; i < n; i++) {
37. temp = (struct Node \*)malloc(sizeof(struct Node));
38. temp->matchId = matchIds[i];
39. strcpy(temp->matchDetails, matchDetails[i]);
40. temp->next = NULL;
41. last->next = temp;
42. last = temp;
43. }
44. }
45. void display(struct Node \*p) {
46. while (p != NULL) {
47. printf("Match ID: %d, Match Details: %s\n", p->matchId, p->matchDetails);
48. p = p->next;
49. }
50. }
51. void insertMatch(int id, char details[]) {
52. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
53. temp->matchId = id;
54. strcpy(temp->matchDetails, details);
55. temp->next = NULL;
56. struct Node \*p = first;
57. while (p->next != NULL) {
58. p = p->next;
59. }
60. p->next = temp;
61. }
62. int deleteMatch(int matchId) {
63. struct Node \*p = first, \*q = NULL;
64. if (p != NULL && p->matchId == matchId) {
65. first = p->next;
66. free(p);
67. return matchId;
68. }
69. while (p != NULL && p->matchId != matchId) {
70. q = p;
71. p = p->next;
72. }
73. if (p == NULL) {
74. printf("Match with ID %d not found.\n", matchId);
75. return -1;
76. }
77. q->next = p->next;
78. free(p);
79. return matchId;
80. }

**Problem 3: Athlete Training Log**

**Description:** Develop a linked list to log training sessions for athletes.

**Operations:**

1. Create a training log.
2. Insert a new training session.
3. Delete a completed or canceled session.
4. Display the training log.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int sessionId;
9. char sessionDetails[100];
10. struct Node \*next;
11. } \*first = NULL;
12. void createTrainingLog(int[], char[][100], int);
13. void display(struct Node \*);
14. void insertSession(int, char[]);
15. int deleteSession(int);
16. int main() {
17. int sessionIds[] = {1, 2, 3};
18. char sessionDetails[][100] = {"Running - 5km", "Weight Training - 1 hour", "Swimming - 30 minutes"};
19. createTrainingLog(sessionIds, sessionDetails, 3);
20. display(first);
21. insertSession(4, "Yoga - 45 minutes");
22. printf("\nUpdated Training Log:\n");
23. display(first);
24. deleteSession(2);
25. printf("\nTraining Log after removing session with ID 2:\n");
26. display(first);
27. return 0;
28. }
29. void createTrainingLog(int sessionIds[], char sessionDetails[][100], int n) {
30. struct Node \*temp, \*last;
31. first = (struct Node \*)malloc(sizeof(struct Node));
32. first->sessionId = sessionIds[0];
33. strcpy(first->sessionDetails, sessionDetails[0]);
34. first->next = NULL;
35. last = first;
36. for (int i = 1; i < n; i++) {
37. temp = (struct Node \*)malloc(sizeof(struct Node));
38. temp->sessionId = sessionIds[i];
39. strcpy(temp->sessionDetails, sessionDetails[i]);
40. temp->next = NULL;
41. last->next = temp;
42. last = temp;
43. }
44. }
45. void display(struct Node \*p) {
46. while (p != NULL) {
47. printf("Session ID: %d, Session Details: %s\n", p->sessionId, p->sessionDetails);
48. p = p->next;
49. }
50. }
51. void insertSession(int id, char details[]) {
52. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
53. temp->sessionId = id;
54. strcpy(temp->sessionDetails, details);
55. temp->next = NULL;
56. struct Node \*p = first;
57. while (p->next != NULL) {
58. p = p->next;
59. }
60. p->next = temp;
61. }
62. int deleteSession(int sessionId) {
63. struct Node \*p = first, \*q = NULL;
64. if (p != NULL && p->sessionId == sessionId) {
65. first = p->next;
66. free(p);
67. return sessionId;
68. }
69. while (p != NULL && p->sessionId != sessionId) {
70. q = p;
71. p = p->next;
72. }
73. if (p == NULL) {
74. printf("Session with ID %d not found.\n", sessionId);
75. return -1;
76. }
77. q->next = p->next;
78. free(p);
79. return sessionId;
80. }

**Problem 4: Sports Equipment Inventory**

**Description:** Use a linked list to manage the inventory of sports equipment.

**Operations:**

1. Create an equipment inventory list.
2. Insert a new equipment item.
3. Delete an item that is no longer usable.
4. Display the current equipment inventory.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int equipmentId;
9. char equipmentName[100];
10. struct Node \*next;
11. } \*first = NULL;
12. void createEquipmentInventory(int[], char[][100], int);
13. void display(struct Node \*);
14. void insertEquipment(int, char[]);
15. int deleteEquipment(int);
16. int main() {
17. int equipmentIds[] = {101, 102, 103};
18. char equipmentNames[][100] = {"Football", "Basketball", "Tennis Racket"};
19. createEquipmentInventory(equipmentIds, equipmentNames, 3);
20. display(first);
21. insertEquipment(104, "Volleyball");
22. printf("\nUpdated Equipment Inventory:\n");
23. display(first);
24. deleteEquipment(102);
25. printf("\nEquipment Inventory after removing item with ID 102:\n");
26. display(first);
27. return 0;
28. }
29. void createEquipmentInventory(int equipmentIds[], char equipmentNames[][100], int n) {
30. struct Node \*temp, \*last;
31. first = (struct Node \*)malloc(sizeof(struct Node));
32. first->equipmentId = equipmentIds[0];
33. strcpy(first->equipmentName, equipmentNames[0]);
34. first->next = NULL;
35. last = first;
36. for (int i = 1; i < n; i++) {
37. temp = (struct Node \*)malloc(sizeof(struct Node));
38. temp->equipmentId = equipmentIds[i];
39. strcpy(temp->equipmentName, equipmentNames[i]);
40. temp->next = NULL;
41. last->next = temp;
42. last = temp;
43. }
44. }
45. void display(struct Node \*p) {
46. while (p != NULL) {
47. printf("Equipment ID: %d, Equipment Name: %s\n", p->equipmentId, p->equipmentName);
48. p = p->next;
49. }
50. }
51. void insertEquipment(int id, char name[]) {
52. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
53. temp->equipmentId = id;
54. strcpy(temp->equipmentName, name);
55. temp->next = NULL;
56. struct Node \*p = first;
57. while (p->next != NULL) {
58. p = p->next;
59. }
60. p->next = temp;
61. }
62. int deleteEquipment(int equipmentId) {
63. struct Node \*p = first, \*q = NULL;
64. if (p != NULL && p->equipmentId == equipmentId) {
65. first = p->next;
66. free(p);
67. return equipmentId;
68. }
69. while (p != NULL && p->equipmentId != equipmentId) {
70. q = p;
71. p = p->next;
72. }
73. if (p == NULL) {
74. printf("Equipment with ID %d not found.\n", equipmentId);
75. return -1;
76. }
77. q->next = p->next;
78. free(p);
79. return equipmentId;
80. }

**Problem 5: Player Performance Tracking**

**Description:** Implement a linked list to track player performance over the season.

**Operations:**

1. Create a performance record list.
2. Insert a new performance entry.
3. Delete an outdated or erroneous entry.
4. Display all performance records.
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8. struct Node {
9. int playerId;
10. int matchesPlayed;
11. int runsScored;
12. struct Node \*next;
13. } \*first = NULL;
14. void createPerformanceList(int[], int[], int[], int);
15. void display(struct Node \*);
16. void insertPerformance(int, int, int);
17. int deletePerformance(int);
18. int main() {
19. int playerIds[] = {1, 2, 3};
20. int matchesPlayed[] = {10, 15, 12};
21. int runsScored[] = {500, 750, 600};
22. createPerformanceList(playerIds, matchesPlayed, runsScored, 3);
23. display(first);
24. insertPerformance(4, 8, 400);
25. printf("\nUpdated Performance List:\n");
26. display(first);
27. deletePerformance(2);
28. printf("\nPerformance List after removing player with ID 2:\n");
29. display(first);
30. return 0;
31. }
32. void createPerformanceList(int playerIds[], int matchesPlayed[], int runsScored[], int n) {
33. struct Node \*temp, \*last;
34. first = (struct Node \*)malloc(sizeof(struct Node));
35. first->playerId = playerIds[0];
36. first->matchesPlayed = matchesPlayed[0];
37. first->runsScored = runsScored[0];
38. first->next = NULL;
39. last = first;
40. for (int i = 1; i < n; i++) {
41. temp = (struct Node \*)malloc(sizeof(struct Node));
42. temp->playerId = playerIds[i];
43. temp->matchesPlayed = matchesPlayed[i];
44. temp->runsScored = runsScored[i];
45. temp->next = NULL;
46. last->next = temp;
47. last = temp;
48. }
49. }
50. void display(struct Node \*p) {
51. while (p != NULL) {
52. printf("Player ID: %d, Matches Played: %d, Runs Scored: %d\n",
53. p->playerId, p->matchesPlayed, p->runsScored);
54. p = p->next;
55. }
56. }
57. void insertPerformance(int playerId, int matchesPlayed, int runsScored) {
58. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
59. temp->playerId = playerId;
60. temp->matchesPlayed = matchesPlayed;
61. temp->runsScored = runsScored;
62. temp->next = NULL;
63. struct Node \*p = first;
64. while (p->next != NULL) {
65. p = p->next;
66. }
67. p->next = temp;
68. }
69. int deletePerformance(int playerId) {
70. struct Node \*p = first, \*q = NULL;
71. if (p != NULL && p->playerId == playerId) {
72. first = p->next;
73. free(p);
74. return playerId;
75. }
76. while (p != NULL && p->playerId != playerId) {
77. q = p;
78. p = p->next;
79. }
80. if (p == NULL) {
81. printf("Player with ID %d not found.\n", playerId);
82. return -1;
83. }
84. q->next = p->next;
85. free(p);
86. return playerId;
87. }

**Problem 6: Event Registration System**

**Description:** Use a linked list to manage athlete registrations for sports events.

**Operations:**

1. Create a registration list.
2. Insert a new registration.
3. Delete a canceled registration.
4. Display all current registrations.
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8. struct Node {
9. int athleteId;
10. char eventName[50];
11. struct Node \*next;
12. } \*first = NULL;
13. void createRegistrationList(int[], char[][50], int);
14. void display(struct Node \*);
15. void insertRegistration(int, char[]);
16. int deleteRegistration(int);
17. int main() {
18. int athleteIds[] = {101, 102, 103};
19. char eventNames[][50] = {"100m Race", "Long Jump", "High Jump"};
20. createRegistrationList(athleteIds, eventNames, 3);
21. display(first);
22. insertRegistration(104, "200m Race");
23. printf("\nUpdated Registration List:\n");
24. display(first);
25. deleteRegistration(102);
26. printf("\nRegistration List after removing athlete with ID 102:\n");
27. display(first);
28. return 0;
29. }
30. void createRegistrationList(int athleteIds[], char eventNames[][50], int n) {
31. struct Node \*temp, \*last;
32. first = (struct Node \*)malloc(sizeof(struct Node));
33. first->athleteId = athleteIds[0];
34. strcpy(first->eventName, eventNames[0]);
35. first->next = NULL;
36. last = first;
37. for (int i = 1; i < n; i++) {
38. temp = (struct Node \*)malloc(sizeof(struct Node));
39. temp->athleteId = athleteIds[i];
40. strcpy(temp->eventName, eventNames[i]);
41. temp->next = NULL;
42. last->next = temp;
43. last = temp;
44. }
45. }
46. void display(struct Node \*p) {
47. while (p != NULL) {
48. printf("Athlete ID: %d, Event Name: %s\n", p->athleteId, p->eventName);
49. p = p->next;
50. }
51. }
52. void insertRegistration(int athleteId, char eventName[]) {
53. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
54. temp->athleteId = athleteId;
55. strcpy(temp->eventName, eventName);
56. temp->next = NULL;
57. struct Node \*p = first;
58. while (p->next != NULL) {
59. p = p->next;
60. }
61. p->next = temp;
62. }
63. int deleteRegistration(int athleteId) {
64. struct Node \*p = first, \*q = NULL;
65. if (p != NULL && p->athleteId == athleteId) {
66. first = p->next;
67. free(p);
68. return athleteId;
69. }
70. while (p != NULL && p->athleteId != athleteId) {
71. q = p;
72. p = p->next;
73. }
74. if (p == NULL) {
75. printf("Athlete with ID %d not found.\n", athleteId);
76. return -1;
77. }
78. q->next = p->next;
79. free(p);
80. return athleteId;
81. }

**Problem 7: Sports League Standings**

**Description:** Develop a linked list to manage the standings of teams in a sports league. **Operations:**

1. Create a league standings list.
2. Insert a new team.
3. Delete a team that withdraws.
4. Display the current league standing.
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8. struct Node {
9. int teamId;
10. char teamName[50];
11. int points;
12. struct Node \*next;
13. } \*first = NULL;
14. void createLeagueStandings(int[], char[][50], int[], int);
15. void display(struct Node \*);
16. void insertTeam(int, char[], int);
17. int deleteTeam(int);
18. int main() {
19. int teamIds[] = {1, 2, 3};
20. char teamNames[][50] = {"Team A", "Team B", "Team C"};
21. int points[] = {10, 15, 8};
22. createLeagueStandings(teamIds, teamNames, points, 3);
23. display(first);
24. insertTeam(4, "Team D", 12);
25. printf("\nUpdated League Standings:\n");
26. display(first);
27. deleteTeam(2);
28. printf("\nLeague Standings after removing Team 2:\n");
29. display(first);
30. return 0;
31. }
32. void createLeagueStandings(int teamIds[], char teamNames[][50], int points[], int n) {
33. struct Node \*temp, \*last;
34. first = (struct Node \*)malloc(sizeof(struct Node));
35. first->teamId = teamIds[0];
36. strcpy(first->teamName, teamNames[0]);
37. first->points = points[0];
38. first->next = NULL;
39. last = first;
40. for (int i = 1; i < n; i++) {
41. temp = (struct Node \*)malloc(sizeof(struct Node));
42. temp->teamId = teamIds[i];
43. strcpy(temp->teamName, teamNames[i]);
44. temp->points = points[i];
45. temp->next = NULL;
46. last->next = temp;
47. last = temp;
48. }
49. }
50. void display(struct Node \*p) {
51. printf("League Standings:\n");
52. printf("ID\tTeam Name\tPoints\n");
53. while (p != NULL) {
54. printf("%d\t%s\t\t%d\n", p->teamId, p->teamName, p->points);
55. p = p->next;
56. }
57. }
58. void insertTeam(int teamId, char teamName[], int points) {
59. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
60. temp->teamId = teamId;
61. strcpy(temp->teamName, teamName);
62. temp->points = points;
63. temp->next = NULL;
64. struct Node \*p = first;
65. while (p->next != NULL) {
66. p = p->next;
67. }
68. p->next = temp;
69. }
70. int deleteTeam(int teamId) {
71. struct Node \*p = first, \*q = NULL;
72. if (p != NULL && p->teamId == teamId) {
73. first = p->next;
74. free(p);
75. return teamId;
76. }
77. while (p != NULL && p->teamId != teamId) {
78. q = p;
79. p = p->next;
80. }
81. if (p == NULL) {
82. printf("Team with ID %d not found.\n", teamId);
83. return -1;
84. }
85. q->next = p->next;
86. free(p);
87. return teamId;
88. }

**Problem 8: Match Result Recording**

**Description:** Implement a linked list to record results of matches.

**Operations:**

1. Create a match result list.
2. Insert a new match result.
3. Delete an incorrect or outdated result.
4. Display all recorded match results.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int matchId;
9. char team1[50];
10. char team2[50];
11. char result[50];
12. struct Node \*next;
13. } \*first = NULL;
14. void createMatchResultList(int[], char[][50], char[][50], char[][50], int);
15. void display(struct Node \*);
16. void insertMatchResult(int, char[], char[], char[]);
17. int deleteMatchResult(int);
18. int main() {
19. int matchIds[] = {1, 2, 3};
20. char teams1[][50] = {"Team A", "Team C", "Team E"};
21. char teams2[][50] = {"Team B", "Team D", "Team F"};
22. char results[][50] = {"Team A won", "Draw", "Team F won"};
23. createMatchResultList(matchIds, teams1, teams2, results, 3);
24. display(first);
25. insertMatchResult(4, "Team G", "Team H", "Team H won");
26. printf("\nUpdated Match Results:\n");
27. display(first);
28. deleteMatchResult(2);
29. printf("\nMatch Results after deleting Match 2:\n");
30. display(first);
31. return 0;
32. }
33. void createMatchResultList(int matchIds[], char teams1[][50], char teams2[][50], char results[][50], int n) {
34. struct Node \*temp, \*last;
35. first = (struct Node \*)malloc(sizeof(struct Node));
36. first->matchId = matchIds[0];
37. for (int i = 0; i < 50; i++) {
38. first->team1[i] = teams1[0][i];
39. first->team2[i] = teams2[0][i];
40. first->result[i] = results[0][i];
41. }
42. first->next = NULL;
43. last = first;
44. for (int i = 1; i < n; i++) {
45. temp = (struct Node \*)malloc(sizeof(struct Node));
46. temp->matchId = matchIds[i];
47. for (int j = 0; j < 50; j++) {
48. temp->team1[j] = teams1[i][j];
49. temp->team2[j] = teams2[i][j];
50. temp->result[j] = results[i][j];
51. }
52. temp->next = NULL;
53. last->next = temp;
54. last = temp;
55. }
56. }
57. void display(struct Node \*p) {
58. printf("Match Results:\n");
59. printf("ID\tTeam 1\t\tTeam 2\t\tResult\n");
60. while (p != NULL) {
61. printf("%d\t%s\t%s\t%s\n", p->matchId, p->team1, p->team2, p->result);
62. p = p->next;
63. }
64. }
65. void insertMatchResult(int matchId, char team1[], char team2[], char result[]) {
66. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
67. temp->matchId = matchId;
68. for (int i = 0; i < 50; i++) {
69. temp->team1[i] = team1[i];
70. temp->team2[i] = team2[i];
71. temp->result[i] = result[i];
72. }
73. temp->next = NULL;
74. struct Node \*p = first;
75. while (p->next != NULL) {
76. p = p->next;
77. }
78. p->next = temp;
79. }
80. int deleteMatchResult(int matchId) {
81. struct Node \*p = first, \*q = NULL;
82. if (p != NULL && p->matchId == matchId) {
83. first = p->next;
84. free(p);
85. return matchId;
86. }
87. while (p != NULL && p->matchId != matchId) {
88. q = p;
89. p = p->next;
90. }
91. if (p == NULL) {
92. printf("Match with ID %d not found.\n", matchId);
93. return -1;
94. }
95. q->next = p->next;
96. free(p);
97. return matchId;
98. }

**Problem 9: Player Injury Tracker**

**Description:** Use a linked list to track injuries of players.

**Operations:**

1. Create an injury tracker list.
2. Insert a new injury report.
3. Delete a resolved or erroneous injury report.
4. Display all current injury reports.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int playerId;
9. char name[50];
10. char injuryDetails[100];
11. struct Node \*next;
12. } \*first = NULL;
13. void createInjuryList(int[], char[][50], char[][100], int);
14. void display(struct Node \*);
15. void insertInjuryReport(int, char[], char[]);
16. int deleteInjuryReport(int);
17. int main() {
18. int playerIds[] = {1, 2, 3};
19. char names[][50] = {"John Doe", "Jane Smith", "Bob Brown"};
20. char injuries[][100] = {"Knee Injury", "Sprained Ankle", "Shoulder Dislocation"};
21. createInjuryList(playerIds, names, injuries, 3);
22. display(first);
23. insertInjuryReport(4, "Alice Green", "Concussion");
24. printf("\nUpdated Injury Reports:\n");
25. display(first);
26. deleteInjuryReport(2);
27. printf("\nInjury Reports after resolving Player 2's injury:\n");
28. display(first);
29. return 0;
30. }
31. void createInjuryList(int playerIds[], char names[][50], char injuries[][100], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->playerId = playerIds[0];
35. for (int i = 0; i < 50; i++) {
36. first->name[i] = names[0][i];
37. }
38. for (int i = 0; i < 100; i++) {
39. first->injuryDetails[i] = injuries[0][i];
40. }
41. first->next = NULL;
42. last = first;
43. for (int i = 1; i < n; i++) {
44. temp = (struct Node \*)malloc(sizeof(struct Node));
45. temp->playerId = playerIds[i];
46. for (int j = 0; j < 50; j++) {
47. temp->name[j] = names[i][j];
48. }
49. for (int j = 0; j < 100; j++) {
50. temp->injuryDetails[j] = injuries[i][j];
51. }
52. temp->next = NULL;
53. last->next = temp;
54. last = temp;
55. }
56. }
57. void display(struct Node \*p) {
58. printf("Player Injury Reports:\n");
59. printf("ID\tName\t\t\tInjury Details\n");
60. while (p != NULL) {
61. printf("%d\t%s\t%s\n", p->playerId, p->name, p->injuryDetails);
62. p = p->next;
63. }
64. }
65. void insertInjuryReport(int playerId, char name[], char injuryDetails[]) {
66. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
67. temp->playerId = playerId;
68. for (int i = 0; i < 50; i++) {
69. temp->name[i] = name[i];
70. }
71. for (int i = 0; i < 100; i++) {
72. temp->injuryDetails[i] = injuryDetails[i];
73. }
74. temp->next = NULL;
75. struct Node \*p = first;
76. while (p->next != NULL) {
77. p = p->next;
78. }
79. p->next = temp;
80. }
81. int deleteInjuryReport(int playerId) {
82. struct Node \*p = first, \*q = NULL;
83. if (p != NULL && p->playerId == playerId) {
84. first = p->next;
85. free(p);
86. return playerId;
87. }
88. while (p != NULL && p->playerId != playerId) {
89. q = p;
90. p = p->next;
91. }
92. if (p == NULL) {
93. printf("Player with ID %d not found.\n", playerId);
94. return -1;
95. }
96. q->next = p->next;
97. free(p);
98. return playerId;
99. }

**Problem 10: Sports Facility Booking System**

**Description:** Manage bookings for sports facilities using a linked list.

**Operations:**

1. Create a booking list.
2. Insert a new booking.
3. Delete a canceled or completed booking.
4. Display all current bookings.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int bookingId;
9. char facilityName[50];
10. char customerName[50];
11. char bookingDate[20];
12. struct Node \*next;
13. } \*first = NULL;
14. void createBookingList(int[], char[][50], char[][50], char[][20], int);
15. void display(struct Node \*);
16. void insertBooking(int, char[], char[], char[]);
17. int deleteBooking(int);
18. int main() {
19. int bookingIds[] = {101, 102, 103};
20. char facilities[][50] = {"Tennis Court", "Football Field", "Basketball Court"};
21. char customers[][50] = {"Ravi Kumar", "Priya Sharma", "Amit Verma"};
22. char dates[][20] = {"2025-01-20", "2025-01-21", "2025-01-22"};
23. createBookingList(bookingIds, facilities, customers, dates, 3);
24. display(first);
25. insertBooking(104, "Swimming Pool", "Sunita Gupta", "2025-01-23");
26. printf("\nUpdated Booking List:\n");
27. display(first);
28. deleteBooking(102);
29. printf("\nBooking List after deleting booking ID 102:\n");
30. display(first);
31. return 0;
32. }
33. void createBookingList(int bookingIds[], char facilities[][50], char customers[][50], char dates[][20], int n) {
34. struct Node \*temp, \*last;
35. first = (struct Node \*)malloc(sizeof(struct Node));
36. first->bookingId = bookingIds[0];
37. for (int i = 0; i < 50; i++) {
38. first->facilityName[i] = facilities[0][i];
39. first->customerName[i] = customers[0][i];
40. }
41. for (int i = 0; i < 20; i++) {
42. first->bookingDate[i] = dates[0][i];
43. }
44. first->next = NULL;
45. last = first;
46. for (int i = 1; i < n; i++) {
47. temp = (struct Node \*)malloc(sizeof(struct Node));
48. temp->bookingId = bookingIds[i];
49. for (int j = 0; j < 50; j++) {
50. temp->facilityName[j] = facilities[i][j];
51. temp->customerName[j] = customers[i][j];
52. }
53. for (int j = 0; j < 20; j++) {
54. temp->bookingDate[j] = dates[i][j];
55. }
56. temp->next = NULL;
57. last->next = temp;
58. last = temp;
59. }
60. }
61. void display(struct Node \*p) {
62. while (p != NULL) {
63. printf("Booking ID: %d\n", p->bookingId);
64. printf("Facility: %s\n", p->facilityName);
65. printf("Customer: %s\n", p->customerName);
66. printf("Booking Date: %s\n", p->bookingDate);
67. printf("------------------------\n");
68. p = p->next;
69. }
70. }
71. void insertBooking(int bookingId, char facility[], char customer[], char date[]) {
72. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
73. struct Node \*last = first;
74. temp->bookingId = bookingId;
75. for (int i = 0; i < 50; i++) {
76. temp->facilityName[i] = facility[i];
77. temp->customerName[i] = customer[i];
78. }
79. for (int i = 0; i < 20; i++) {
80. temp->bookingDate[i] = date[i];
81. }
82. temp->next = NULL;
83. if (first == NULL) {
84. first = temp;
85. } else {
86. while (last->next != NULL) {
87. last = last->next;
88. }
89. last->next = temp;
90. }
91. }
92. int deleteBooking(int bookingId) {
93. struct Node \*temp = first;
94. struct Node \*prev = NULL;
95. if (temp != NULL && temp->bookingId == bookingId) {
96. first = temp->next;
97. free(temp);
98. return 1;
99. }
100. while (temp != NULL && temp->bookingId != bookingId) {
101. prev = temp;
102. temp = temp->next;
103. }
104. if (temp == NULL) {
105. return 0;
106. }
107. prev->next = temp->next;
108. free(temp);
109. return 1;
110. }

**Problem 11: Coaching Staff Management**

**Description:** Use a linked list to manage the coaching staff of a sports team.

**Operations:**

1. Create a coaching staff list.
2. Insert a new coach.
3. Delete a coach who leaves the team.
4. Display the current coaching staff.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int staffId;
9. char name[50];
10. char role[50];
11. struct Node \*next;
12. } \*first = NULL;
13. void createStaffList(int[], char[][50], char[][50], int);
14. void display(struct Node \*);
15. void insertStaff(int, char[], char[]);
16. int deleteStaff(int);
17. int main() {
18. int staffIds[] = {201, 202, 203};
19. char names[][50] = {"Anil Kumar", "Sushma Sharma", "Rajesh Verma"};
20. char roles[][50] = {"Head Coach", "Assistant Coach", "Physio"};
21. createStaffList(staffIds, names, roles, 3);
22. display(first);
23. insertStaff(204, "Priya Reddy", "Goalkeeping Coach");
24. printf("\nUpdated Coaching Staff List:\n");
25. display(first);
26. deleteStaff(202);
27. printf("\nCoaching Staff List after deleting staff ID 202:\n");
28. display(first);
29. return 0;
30. }
31. void createStaffList(int staffIds[], char names[][50], char roles[][50], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->staffId = staffIds[0];
35. for (int i = 0; i < 50; i++) {
36. first->name[i] = names[0][i];
37. first->role[i] = roles[0][i];
38. }
39. first->next = NULL;
40. last = first;
41. for (int i = 1; i < n; i++) {
42. temp = (struct Node \*)malloc(sizeof(struct Node));
43. temp->staffId = staffIds[i];
44. for (int j = 0; j < 50; j++) {
45. temp->name[j] = names[i][j];
46. temp->role[j] = roles[i][j];
47. }
48. temp->next = NULL;
49. last->next = temp;
50. last = temp;
51. }
52. }
53. void display(struct Node \*p) {
54. while (p != NULL) {
55. printf("Staff ID: %d\n", p->staffId);
56. printf("Name: %s\n", p->name);
57. printf("Role: %s\n", p->role);
58. printf("------------------------\n");
59. p = p->next;
60. }
61. }
62. void insertStaff(int staffId, char name[], char role[]) {
63. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
64. struct Node \*last = first;
65. temp->staffId = staffId;
66. for (int i = 0; i < 50; i++) {
67. temp->name[i] = name[i];
68. temp->role[i] = role[i];
69. }
70. temp->next = NULL;
71. if (first == NULL) {
72. first = temp;
73. } else {
74. while (last->next != NULL) {
75. last = last->next;
76. }
77. last->next = temp;
78. }
79. }
80. int deleteStaff(int staffId) {
81. struct Node \*temp = first;
82. struct Node \*prev = NULL;
83. if (temp != NULL && temp->staffId == staffId) {
84. first = temp->next;
85. free(temp);
86. return 1;
87. }
88. while (temp != NULL && temp->staffId != staffId) {
89. prev = temp;
90. temp = temp->next;
91. }
92. if (temp == NULL) {
93. return 0;
94. }
95. prev->next = temp->next;
96. free(temp);
97. return 1;
98. }

**Problem 12: Fan Club Membership Management**

**Description:** Implement a linked list to manage memberships in a sports team’s fan club.

**Operations:**

1. Create a membership list.
2. Insert a new member.
3. Delete a member who cancels their membership.
4. Display all current members.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int memberId;
9. char name[50];
10. char city[50];
11. struct Node \*next;
12. } \*first = NULL;
13. void createMembershipList(int[], char[][50], char[][50], int);
14. void display(struct Node \*);
15. void insertMember(int, char[], char[]);
16. int deleteMember(int);
17. int main() {
18. int memberIds[] = {101, 102, 103};
19. char names[][50] = {"Amit Patel", "Rajesh Kumar", "Simran Gupta"};
20. char cities[][50] = {"Mumbai", "Delhi", "Bangalore"};
21. createMembershipList(memberIds, names, cities, 3);
22. display(first);
23. insertMember(104, "Priya Sharma", "Chennai");
24. printf("\nUpdated Fan Club Members List:\n");
25. display(first);
26. deleteMember(102);
27. printf("\nFan Club Members List after deleting member ID 102:\n");
28. display(first);
29. return 0;
30. }
31. void createMembershipList(int memberIds[], char names[][50], char cities[][50], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->memberId = memberIds[0];
35. for (int i = 0; i < 50; i++) {
36. first->name[i] = names[0][i];
37. first->city[i] = cities[0][i];
38. }
39. first->next = NULL;
40. last = first;
41. for (int i = 1; i < n; i++) {
42. temp = (struct Node \*)malloc(sizeof(struct Node));
43. temp->memberId = memberIds[i];
44. for (int j = 0; j < 50; j++) {
45. temp->name[j] = names[i][j];
46. temp->city[j] = cities[i][j];
47. }
48. temp->next = NULL;
49. last->next = temp;
50. last = temp;
51. }
52. }
53. void display(struct Node \*p) {
54. while (p != NULL) {
55. printf("Member ID: %d\n", p->memberId);
56. printf("Name: %s\n", p->name);
57. printf("City: %s\n", p->city);
58. printf("------------------------\n");
59. p = p->next;
60. }
61. }
62. void insertMember(int memberId, char name[], char city[]) {
63. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
64. struct Node \*last = first;
65. temp->memberId = memberId;
66. for (int i = 0; i < 50; i++) {
67. temp->name[i] = name[i];
68. temp->city[i] = city[i];
69. }
70. temp->next = NULL;
71. if (first == NULL) {
72. first = temp;
73. } else {
74. while (last->next != NULL) {
75. last = last->next;
76. }
77. last->next = temp;
78. }
79. }
80. int deleteMember(int memberId) {
81. struct Node \*temp = first;
82. struct Node \*prev = NULL;
83. if (temp != NULL && temp->memberId == memberId) {
84. first = temp->next;
85. free(temp);
86. return 1;
87. }
88. while (temp != NULL && temp->memberId != memberId) {
89. prev = temp;
90. temp = temp->next;
91. }
92. if (temp == NULL) {
93. return 0;
94. }
95. prev->next = temp->next;
96. free(temp);
97. return 1;
98. }

**Problem 13: Sports Event Scheduling**

**Description:** Use a linked list to manage the schedule of sports events.

**Operations:**

1. Create an event schedule.
2. Insert a new event.
3. Delete a completed or canceled event.
4. Display the current event schedule.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Node {
8. int eventId;
9. char eventName[50];
10. char eventDate[20];
11. struct Node \*next;
12. } \*first = NULL;
13. void createEventSchedule(int[], char[][50], char[][50], int);
14. void display(struct Node \*);
15. void insertEvent(int, char[], char[]);
16. int deleteEvent(int);
17. int main() {
18. int eventIds[] = {201, 202, 203};
19. char eventNames[][50] = {"Football Tournament", "Basketball Finals", "Tennis Championship"};
20. char eventDates[][50] = {"2025-03-15", "2025-03-20", "2025-03-25"};
21. createEventSchedule(eventIds, eventNames, eventDates, 3);
22. display(first);
23. insertEvent(204, "Cricket World Cup", "2025-04-10");
24. printf("\nUpdated Event Schedule:\n");
25. display(first);
26. deleteEvent(202);
27. printf("\nEvent Schedule after deleting event ID 202:\n");
28. display(first);
29. return 0;
30. }
31. void createEventSchedule(int eventIds[], char eventNames[][50], char eventDates[][50], int n) {
32. struct Node \*temp, \*last;
33. first = (struct Node \*)malloc(sizeof(struct Node));
34. first->eventId = eventIds[0];
35. for (int i = 0; i < 50; i++) {
36. first->eventName[i] = eventNames[0][i];
37. first->eventDate[i] = eventDates[0][i];
38. }
39. first->next = NULL;
40. last = first;
41. for (int i = 1; i < n; i++) {
42. temp = (struct Node \*)malloc(sizeof(struct Node));
43. temp->eventId = eventIds[i];
44. for (int j = 0; j < 50; j++) {
45. temp->eventName[j] = eventNames[i][j];
46. temp->eventDate[j] = eventDates[i][j];
47. }
48. temp->next = NULL;
49. last->next = temp;
50. last = temp;
51. }
52. }
53. void display(struct Node \*p) {
54. while (p != NULL) {
55. printf("Event ID: %d\n", p->eventId);
56. printf("Event Name: %s\n", p->eventName);
57. printf("Event Date: %s\n", p->eventDate);
58. printf("------------------------\n");
59. p = p->next;
60. }
61. }
62. void insertEvent(int eventId, char eventName[], char eventDate[]) {
63. struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));
64. struct Node \*last = first;
65. temp->eventId = eventId;
66. for (int i = 0; i < 50; i++) {
67. temp->eventName[i] = eventName[i];
68. temp->eventDate[i] = eventDate[i];
69. }
70. temp->next = NULL;
71. if (first == NULL) {
72. first = temp;
73. } else {
74. while (last->next != NULL) {
75. last = last->next;
76. }
77. last->next = temp;
78. }
79. }
80. int deleteEvent(int eventId) {
81. struct Node \*temp = first;
82. struct Node \*prev = NULL;
83. if (temp != NULL && temp->eventId == eventId) {
84. first = temp->next;
85. free(temp);
86. return 1;
87. }
88. while (temp != NULL && temp->eventId != eventId) {
89. prev = temp;
90. temp = temp->next;
91. }
92. if (temp == NULL) {
93. return 0;
94. }
95. prev->next = temp->next;
96. free(temp);
97. return 1;
98. }

**Problem 14: Player Transfer Records**

**Description:** Maintain a linked list to track player transfers between teams.

**Operations:**

1. Create a transfer record list.
2. Insert a new transfer record.
3. Delete an outdated or erroneous transfer record.
4. Display all current transfer records.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Player {
8. int playerID;
9. char name[50];
10. struct Player\* next;
11. } \*first = NULL;
12. void createPlayerList();
13. void displayPlayers();
14. void insertPlayer(int id, char name[50]);
15. void deletePlayer(int id);
16. int main() {
17. createPlayerList();
18. insertPlayer(1, "Amit Kumar");
19. insertPlayer(2, "Ravi Singh");
20. insertPlayer(3, "Priya Sharma");
21. displayPlayers();
22. deletePlayer(2);
23. displayPlayers();
24. return 0;
25. }
26. void createPlayerList() {
27. first = NULL;
28. }
29. void insertPlayer(int id, char name[50]) {
30. struct Player\* temp = (struct Player\*)malloc(sizeof(struct Player));
31. temp->playerID = id;
32. strcpy(temp->name, name);
33. temp->next = NULL;
34. if (first == NULL) {
35. first = temp;
36. } else {
37. struct Player\* p = first;
38. while (p->next != NULL) {
39. p = p->next;
40. }
41. p->next = temp;
42. }
43. }
44. void deletePlayer(int id) {
45. struct Player \*p = first, \*q = NULL;
46. if (p != NULL && p->playerID == id) {
47. first = p->next;
48. free(p);
49. return;
50. }
51. while (p != NULL && p->playerID != id) {
52. q = p;
53. p = p->next;
54. }
55. if (p == NULL) return;
56. q->next = p->next;
57. free(p);
58. }
59. void displayPlayers() {
60. struct Player\* p = first;
61. while (p != NULL) {
62. printf("Player ID: %d, Name: %s\n", p->playerID, p->name);
63. p = p->next;
64. }
65. printf("\n");
66. }

**Problem 15: Championship Points Tracker**

**Description:** Implement a linked list to track championship points for teams.

**Operations:**

1. Create a points tracker list.
2. Insert a new points entry.
3. Delete an incorrect or outdated points entry.
4. Display all current points standings.
5. #include <stdio.h>
6. #include <stdlib.h>
7. struct Team {
8. int teamID;
9. char name[50];
10. int points;
11. struct Team\* next;
12. } \*first = NULL;
13. void createPointsTracker();
14. void displayPointsStandings();
15. void insertPointsEntry(int id, char name[50], int points);
16. void deletePointsEntry(int id);
17. int main() {
18. createPointsTracker();
19. insertPointsEntry(1, "Delhi Tigers", 20);
20. insertPointsEntry(2, "Mumbai Warriors", 15);
21. insertPointsEntry(3, "Chennai Eagles", 18);
22. displayPointsStandings();
23. deletePointsEntry(2);
24. displayPointsStandings();
25. return 0;
26. }
27. void createPointsTracker() {
28. first = NULL;
29. }
30. void insertPointsEntry(int id, char name[50], int points) {
31. struct Team\* temp = (struct Team\*)malloc(sizeof(struct Team));
32. temp->teamID = id;
33. strcpy(temp->name, name);
34. temp->points = points;
35. temp->next = NULL;
36. if (first == NULL) {
37. first = temp;
38. } else {
39. struct Team\* p = first;
40. while (p->next != NULL) {
41. p = p->next;
42. }
43. p->next = temp;
44. }
45. }
46. void deletePointsEntry(int id) {
47. struct Team\* p = first;
48. struct Team\* q = NULL;
49. if (p != NULL && p->teamID == id) {
50. first = p->next;
51. free(p);
52. return;
53. }
54. while (p != NULL && p->teamID != id) {
55. q = p;
56. p = p->next;
57. }
58. if (p == NULL) return;
59. q->next = p->next;
60. free(p);
61. }
62. void displayPointsStandings() {
63. struct Team\* p = first;
64. while (p != NULL) {
65. printf("Team: %s, Points: %d\n", p->name, p->points);
66. p = p->next;
67. }
68. printf("\n");
69. }