

Singly Linked List Implementation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure for singly linked list node
5  typedef struct Node {
6      int data;
7      struct Node* next;
8  } Node;
9
10 // Function to insert at the beginning
11 Node* insertAtBeginning(Node* head, int data) {
12     Node* newNode = (Node*)malloc(sizeof(Node));
13     newNode->data = data;
14     newNode->next = head;
15     return newNode;
16 }
17
18 // Function to insert at the end
19 Node* insertAtEnd(Node* head, int data) {
20     Node* newNode = (Node*)malloc(sizeof(Node));
21     newNode->data = data;
22     newNode->next = NULL;
23     if (head == NULL) {
24         return newNode;
25     }
26     Node* temp = head;
27     while (temp->next != NULL) {
28         temp = temp->next;
29     }
30     temp->next = newNode;
31     return head;
32 }
33
34 // Function to insert at a specified position
35 Node* insertAtPosition(Node* head, int data, int position) {
36     Node* newNode = (Node*)malloc(sizeof(Node));
37     newNode->data = data;
38     if (position == 1) {
39         newNode->next = head;
40         return newNode;
41     }
42     Node* temp = head;
43     for (int i = 1; i < position - 1 && temp != NULL; i++) {
44         temp = temp->next;
```

```

44     temp = temp->next;
45 }
46 if (temp == NULL) {
47     free(newNode);
48     printf("Position out of bounds.\n");
49     return head;
50 }
51 newNode->next = temp->next;
52 temp->next = newNode;
53 return head;
54 }
55
56 // Function to delete from the beginning
57 Node* deleteFromBeginning(Node* head) {
58     if (head == NULL) {
59         printf("List is empty.\n");
60         return NULL;
61     }
62     Node* temp = head;
63     head = head->next;
64     free(temp);
65     return head;
66 }
67
68 // Function to delete from the end
69 Node* deleteFromEnd(Node* head) {
70     if (head == NULL || head->next == NULL) {
71         free(head);
72         return NULL;
73     }
74     Node* temp = head;
75     while (temp->next->next != NULL) {
76         temp = temp->next;
77     }
78     free(temp->next);
79     temp->next = NULL;
80     return head;
81 }
82
83 // Function to delete from a specified position
84 Node* deleteFromPosition(Node* head, int position) {
85     if (head == NULL) {
86         printf("List is empty.\n");
87         return NULL;

```

```

87     return NULL;
88 }
89 if (position == 1) {
90     return deleteFromBeginning(head);
91 }
92 Node* temp = head;
93 for (int i = 1; i < position - 1 && temp->next != NULL; i++) {
94     temp = temp->next;
95 }
96 if (temp->next == NULL) {
97     printf("Position out of bounds.\n");
98     return head;
99 }
100 Node* nodeToDelete = temp->next;
101 temp->next = nodeToDelete->next;
102 free(nodeToDelete);
103 return head;
104 }
105
106 // Function to display the list
107 void displayList(Node* head) {
108     Node* temp = head;
109     while (temp != NULL) {
110         printf("%d -> ", temp->data);
111         temp = temp->next;
112     }
113     printf("NULL\n");
114 }
115
116 // Function to free the list and prevent memory leaks
117 void freeList(Node* head) {
118     Node* temp;
119     while (head != NULL) {
120         temp = head;
121         head = head->next;
122         free(temp);
123     }
124 }
125
126 int main() {
127     Node* head = NULL;
128
129     // Sample operations
130     head = insertAtBeginning(head, 10);

```

```

100     Node* nodeToDelete = temp->next;
101     temp->next = nodeToDelete->next;
102     free(nodeToDelete);
103     return head;
104 }
105
106 // Function to display the list
107 void displayList(Node* head) {
108     Node* temp = head;
109     while (temp != NULL) {
110         printf("%d -> ", temp->data);
111         temp = temp->next;
112     }
113     printf("NULL\n");
114 }
115
116 // Function to free the list and prevent memory leaks
117 void freeList(Node* head) {
118     Node* temp;
119     while (head != NULL) {
120         temp = head;
121         head = head->next;
122         free(temp);
123     }
124 }
125
126 int main() {
127     Node* head = NULL;
128
129     // Sample operations
130     head = insertAtBeginning(head, 10);
131     head = insertAtEnd(head, 20);
132     head = insertAtPosition(head, 15, 2);
133     displayList(head);
134
135     head = deleteFromPosition(head, 2);
136     displayList(head);
137
138     freeList(head);
139     return 0;
140 }

```

Double Linked List Implementation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure for doubly linked list node
5  typedef struct DNode {
6      int data;
7      struct DNode* next;
8      struct DNode* prev;
9  } DNode;
10
11 // Function to insert at the beginning
12 DNode* insertAtBeginning(DNode* head, int data) {
13     DNode* newNode = (DNode*)malloc(sizeof(DNode));
14     newNode->data = data;
15     newNode->next = head;
16     newNode->prev = NULL;
17     if (head != NULL) {
18         head->prev = newNode;
19     }
20     return newNode;
21 }
22
23 // Function to insert at the end
24 DNode* insertAtEnd(DNode* head, int data) {
25     DNode* newNode = (DNode*)malloc(sizeof(DNode));
26     newNode->data = data;
27     newNode->next = NULL;
28     if (head == NULL) {
29         newNode->prev = NULL;
30         return newNode;
31     }
32     DNode* temp = head;
33     while (temp->next != NULL) {
34         temp = temp->next;
35     }
36     temp->next = newNode;
37     newNode->prev = temp;
38     return head;
39 }
40
41 // Function to insert at a specified position
42 DNode* insertAtPosition(DNode* head, int data, int position) {
43     DNode* newNode = (DNode*)malloc(sizeof(DNode));
44     newNode->data = data;
```

```

44     newNode->data = data;
45     if (position == 1) {
46         newNode->next = head;
47         newNode->prev = NULL;
48         if (head != NULL) {
49             head->prev = newNode;
50         }
51         return newNode;
52     }
53     DNode* temp = head;
54     for (int i = 1; i < position - 1 && temp != NULL; i++) {
55         temp = temp->next;
56     }
57     if (temp == NULL) {
58         free(newNode);
59         printf("Position out of bounds.\n");
60         return head;
61     }
62     newNode->next = temp->next;
63     if (temp->next != NULL) {
64         temp->next->prev = newNode;
65     }
66     temp->next = newNode;
67     newNode->prev = temp;
68     return head;
69 }
70
71 // Function to delete from the beginning
72 DNode* deleteFromBeginning(DNode* head) {
73     if (head == NULL) {
74         printf("List is empty.\n");
75         return NULL;
76     }
77     DNode* temp = head;
78     head = head->next;
79     if (head != NULL) {
80         head->prev = NULL;
81     }
82     free(temp);
83     return head;
84 }
85
86 // Function to delete from the end
87 DNode* deleteFromEnd(DNode* head) {

```

```

87 - DNode* deleteFromEnd(DNode* head) {
88 -     if (head == NULL || head->next == NULL) {
89         free(head);
90         return NULL;
91     }
92     DNode* temp = head;
93 -     while (temp->next != NULL) {
94         temp = temp->next;
95     }
96     temp->prev->next = NULL;
97     free(temp);
98     return head;
99 }
100
101 // Function to delete from a specified position
102 - DNode* deleteFromPosition(DNode* head, int position) {
103 -     if (head == NULL) {
104         printf("List is empty.\n");
105         return NULL;
106     }
107 -     if (position == 1) {
108         return deleteFromBeginning(head);
109     }
110     DNode* temp = head;
111 -     for (int i = 1; i < position && temp != NULL; i++) {
112         temp = temp->next;
113     }
114 -     if (temp == NULL) {
115         printf("Position out of bounds.\n");
116         return head;
117     }
118 -     if (temp->next != NULL) {
119         temp->next->prev = temp->prev;
120     }
121 -     if (temp->prev != NULL) {
122         temp->prev->next = temp->next;
123     }
124     free(temp);
125     return head;
126 }
127
128 // Function to display the list
129 - void displayDList(DNode* head) {
130     DNode* temp = head;

```

```

121 -     if (temp->prev != NULL) {
122         temp->prev->next = temp->next;
123     }
124     free(temp);
125     return head;
126 }
127
128 // Function to display the list
129 - void displayDList(DNode* head) {
130     DNode* temp = head;
131 -     while (temp != NULL) {
132         printf("%d <-> ", temp->data);
133         temp = temp->next;
134     }
135     printf("NULL\n");
136 }
137
138 // Function to free the list and prevent memory leaks
139 - void freeDList(DNode* head) {
140     DNode* temp;
141 -     while (head != NULL) {
142         temp = head;
143         head = head->next;
144         free(temp);
145     }
146 }
147
148 - int main() {
149     DNode* head = NULL;
150
151     // Sample operations
152     head = insertAtBeginning(head, 30);
153     head = insertAtEnd(head, 40);
154     head = insertAtPosition(head, 35, 2);
155     displayDList(head);
156
157     head = deleteFromPosition(head, 2);
158     displayDList(head);
159
160     freeDList(head);
161     return 0;
162 }

```


To-Do List Application

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // Structure for a to-do list node
6  typedef struct Task {
7      char task[100];
8      struct Task* next;
9  } Task;
10
11 // Function to add a task at the end of the list
12 Task* addTask(Task* head, const char* taskDescription) {
13     Task* newTask = (Task*)malloc(sizeof(Task));
14     if (newTask == NULL) {
15         printf("Memory allocation failed.\n");
16         return head;
17     }
18     strcpy(newTask->task, taskDescription);
19     newTask->next = NULL;
20
21     if (head == NULL) {
22         return newTask;
23     }
24
25     Task* temp = head;
26     while (temp->next != NULL) {
27         temp = temp->next;
28     }
29     temp->next = newTask;
30     return head;
31 }
32
33 // Function to remove the first task from the list (mark as completed)
34 Task* removeTask(Task* head) {
35     if (head == NULL) {
36         printf("No tasks to remove.\n");
37         return NULL;
38     }
39     Task* temp = head;
40     head = head->next;
41     printf("Task completed: %s\n", temp->task);
42     free(temp);
43     return head;
44 }
```

```

46 // Function to display all tasks in the to-do list
47 void displayTasks(Task* head) {
48     if (head == NULL) {
49         printf("No tasks in the to-do list.\n");
50         return;
51     }
52     printf("To-Do List:\n");
53     Task* temp = head;
54     while (temp != NULL) {
55         printf("- %s\n", temp->task);
56         temp = temp->next;
57     }
58 }
59
60 // Function to free all memory and prevent memory leaks
61 void freeTasks(Task* head) {
62     Task* temp;
63     while (head != NULL) {
64         temp = head;
65         head = head->next;
66         free(temp);
67     }
68 }
69
70 int main() {
71     Task* toDoList = NULL;
72     int choice;
73     char taskDescription[100];
74
75     while (1) {
76         printf("\nTo-Do List Menu:\n");
77         printf("1. Add Task\n");
78         printf("2. Remove First Task\n");
79         printf("3. Display Tasks\n");
80         printf("4. Exit\n");
81         printf("Enter your choice: ");
82         scanf("%d", &choice);
83         getchar(); // Clear newline character from buffer
84
85         switch (choice) {
86             case 1:
87                 printf("Enter task description: ");
88                 fgets(taskDescription, sizeof(taskDescription), stdin);
89                 taskDescription[strcspn(taskDescription, "\n")] = '\0'; // Remove newline

```

```

88         fgets(taskDescription, sizeof(taskDescription), stdin);
89         taskDescription[strcspn(taskDescription, "\n")] = '\0'; // Remove newline
           character
90         toDoList = addTask(toDoList, taskDescription);
91         break;
92     case 2:
93         toDoList = removeTask(toDoList);
94         break;
95     case 3:
96         displayTasks(toDoList);
97         break;
98     case 4:
99         freeTasks(toDoList);
100        printf("Exiting. All tasks have been cleared.\n");
101        return 0;
102    default:
103        printf("Invalid choice. Please try again.\n");
104    }
105 }
106
107     return 0;
108 }

```