

Singly Linked List:

```
4 // Define a structure for a node in the singly linked list
5 struct Node {
6     int data;
7     struct Node* next;
8 }
9
10 // Function to create a new node
11 struct Node* createNode(int data) {
12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13     newNode->data = data;
14     newNode->next = NULL;
15     return newNode;
16 }
17
18 // Function to insert data at the end of the linked list
19 void insertAtEnd(struct Node** head, int data) {
20     struct Node* newNode = createNode(data);
21     if (*head == NULL) {
22         *head = newNode;
23     } else {
24         struct Node* current = *head;
25         while (current->next != NULL) {
26             current = current->next;
27         }
28         current->next = newNode;
29     }
30 }
31
32 // Function to insert data at the beginning of the linked list
33 void insertAtStart(struct Node** head, int data) {
34     struct Node* newNode = createNode(data);
35     newNode->next = *head;
36     *head = newNode;
37 }
38
```

```
68 // Function to check if the linked list is empty
69 int isEmpty(struct Node* head) {
70     return head == NULL;
71 }
72
73 // Function to delete data from the start of the linked list
74 void deleteAtStart(struct Node** head) {
75     if (*head == NULL) {
76         printf("Linked list is empty. Nothing to delete.\n");
77         return;
78     }
79
80     struct Node* temp = *head;
81     *head = (*head)->next;
82     free(temp);
83 }
84
```

```

39 // Function to insert data at a specific index position
40 void insertAtIndex(struct Node** head, int data, int position) {
41     if (position < 0) {
42         printf("Invalid position\n");
43         return;
44     }
45
46     if (position == 0) {
47         insertAtStart(head, data);
48         return;
49     }
50
51     struct Node* newNode = createNode(data);
52     struct Node* current = *head;
53     int currentPos = 0;
54     while (current != NULL && currentPos < position - 1) {
55         current = current->next;
56         currentPos++;
57     }
58
59     if (current == NULL) {
60         printf("Invalid position\n");
61         free(newNode);
62     } else {
63         newNode->next = current->next;
64         current->next = newNode;
65     }
66 }

```

```

85 // Function to delete data from the end of the linked list
86 void deleteAtEnd(struct Node** head) {
87     if (*head == NULL) {
88         printf("Linked list is empty. Nothing to delete.\n");
89         return;
90     }
91
92     if ((*head)->next == NULL) {
93         free(*head);
94         *head = NULL;
95         return;
96     }
97
98     struct Node* current = *head;
99     while (current->next->next != NULL) {
100         current = current->next;
101     }
102
103     free(current->next);
104     current->next = NULL;
105 }

```

```

135 // Function to print the linked list
136 void displayList(struct Node* head) {
137     struct Node* current = head;
138     while (current != NULL) {
139         printf("%d -> ", current->data);
140         current = current->next;
141     }
142     printf("NULL\n");
143 }

```

```

107 // Function to delete data at a specific index position
108 void deleteAtIndex(struct Node** head, int position) {
109     if (position < 0) {
110         printf("Invalid position\n");
111         return;
112     }
113
114     if (position == 0) {
115         deleteAtStart(head);
116         return;
117     }
118
119     struct Node* current = *head;
120     int currentPos = 0;
121     while (current != NULL && currentPos < position - 1) {
122         current = current->next;
123         currentPos++;
124     }
125
126     if (current == NULL || current->next == NULL) {
127         printf("Invalid position\n");
128     } else {
129         struct Node* temp = current->next;
130         current->next = temp->next;
131         free(temp);
132     }
133 }

```

```

145 int main() {
146     struct Node* head = NULL;
147     int choice, data, position;
148
149     while (1) {
150         printf("1. Insert at the end\n");
151         printf("2. Insert at the beginning\n");
152         printf("3. Insert at a specific position\n");
153         printf("4. Check if the linked list is empty\n");
154         printf("5. Delete from the beginning\n");
155         printf("6. Delete from the end\n");
156         printf("7. Delete from a specific position\n");
157         printf("8. Display the linked list\n");
158         printf("9. Quit\n");
159         printf("Enter your choice: ");
160         scanf("%d", &choice);

```

```

162         switch (choice) {
163             case 1:
164                 printf("Enter data to insert at the end: ");
165                 scanf("%d", &data);
166                 insertAtEnd(&head, data);
167                 break;
168
169             case 2:
170                 printf("Enter data to insert at the beginning: ");
171                 scanf("%d", &data);
172                 insertAtStart(&head, data);
173                 break;

```

```
175     case 3:
176         printf("Enter data to insert: ");
177         scanf("%d", &data);
178         printf("Enter the position to insert at: ");
179         scanf("%d", &position);
180         insertAtIndex(&head, data, position);
181         break;
182
183     case 4:
184         if (isEmpty(head)) {
185             printf("Linked list is empty.\n");
186         } else {
187             printf("Linked list is not empty.\n");
188         }
189         break;
190
191     case 5:
192         deleteAtStart(&head);
193         break;
194
195     case 6:
196         deleteAtEnd(&head);
197         break;
198
199     case 7:
200         printf("Enter the position to delete from: ");
201         scanf("%d", &position);
202         deleteAtIndex(&head, position);
203         break;
204
205     case 8:
206         displayList(head);
207         break;
```

```
209     default:
210         printf("Invalid choice. Please try again.\n");
211         break;
212     }
213 }
214
215 return 0;
216 }
```

Doubly Linked List:

```
4 // Define a structure for a node in the doubly linked list
5 struct Node {
6     int data;
7     struct Node* prev;
8     struct Node* next;
9 };
10
11 // Function to create a new node
12 struct Node* createNode(int data) {
13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = data;
15     newNode->prev = NULL;
16     newNode->next = NULL;
17     return newNode;
18 }
19
20 // Function to insert data at the end of the doubly linked list
21 void insertAtEnd(struct Node** head, int data) {
22     struct Node* newNode = createNode(data);
23     if (*head == NULL) {
24         *head = newNode;
25     } else {
26         struct Node* current = *head;
27         while (current->next != NULL) {
28             current = current->next;
29         }
30         current->next = newNode;
31         newNode->prev = current;
32     }
33 }
34
```

```
35 // Function to insert data at the beginning of the doubly linked list
36 void insertAtStart(struct Node** head, int data) {
37     struct Node* newNode = createNode(data);
38     newNode->next = *head;
39     if (*head != NULL) {
40         (*head)->prev = newNode;
41     }
42     *head = newNode;
43 }
44
45 // Function to check if the doubly linked list is empty
46 int isEmpty(struct Node* head) {
47     return head == NULL;
48 }
49
```

```

50 // Function to insert data at a specific index position
51 void insertAtIndex(struct Node** head, int data, int position) {
52     if (position < 0) {
53         printf("Invalid position\n");
54         return;
55     }
56
57     if (position == 0) {
58         insertAtStart(head, data);
59         return;
60     }
61
62     struct Node* newNode = createNode(data);
63     struct Node* current = *head;
64     int currentPos = 0;
65     while (current != NULL && currentPos < position - 1) {
66         current = current->next;
67         currentPos++;
68     }
69
70     if (current == NULL) {
71         printf("Invalid position\n");
72         free(newNode);
73     } else {
74         newNode->prev = current;
75         newNode->next = current->next;
76         if (current->next != NULL) {
77             current->next->prev = newNode;
78         }
79         current->next = newNode;
80     }
81 }

```

```

84 // Function to delete data from the start of the doubly linked list
85 void deleteAtStart(struct Node** head) {
86     if (*head == NULL) {
87         printf("Doubly linked list is empty. Nothing to delete.\n");
88         return;
89     }
90
91     struct Node* temp = *head;
92     *head = (*head)->next;
93     if (*head != NULL) {
94         (*head)->prev = NULL;
95     }
96     free(temp);
97 }
98
99 // Function to delete data from the end of the doubly linked list
100 void deleteAtEnd(struct Node** head) {
101     if (*head == NULL) {
102         printf("Doubly linked list is empty. Nothing to delete.\n");
103         return;
104     }
105
106     struct Node* current = *head;
107     while (current->next != NULL) {
108         current = current->next;
109     }
110
111     if (current->prev != NULL) {
112         current->prev->next = NULL;
113     } else {
114         *head = NULL;
115     }
116     free(current);
117 }

```

```

119 // Function to delete data at a specific index position
120 void deleteAtIndex(struct Node** head, int position) {
121     if (position < 0) {
122         printf("Invalid position\n");
123         return;
124     }
125
126     if (position == 0) {
127         deleteAtStart(head);
128         return;
129     }
130
131     struct Node* current = *head;
132     int currentPos = 0;
133     while (current != NULL && currentPos < position) {
134         current = current->next;
135         currentPos++;
136     }
137
138     if (current == NULL) {
139         printf("Invalid position\n");
140     } else {
141         if (current->prev != NULL) {
142             current->prev->next = current->next;
143         } else {
144             *head = current->next;
145         }
146         if (current->next != NULL) {
147             current->next->prev = current->prev;
148         }
149         free(current);
150     }
151 }

```

```

153 // Function to print the doubly linked list in both forward and backward directions
154 void displayList(struct Node* head) {
155     struct Node* current = head;
156
157     printf("Forward: ");
158     while (current != NULL) {
159         printf("%d -> ", current->data);
160         current = current->next;
161     }
162     printf("NULL\n");
163
164     current = head;
165     printf("Backward: ");
166     while (current->next != NULL) {
167         current = current->next;
168     }
169     while (current != NULL) {
170         printf("%d -> ", current->data);
171         current = current->prev;
172     }
173     printf("NULL\n");
174 }
175

```

```
176 int main() {
177     struct Node* head = NULL;
178     int choice, data, position;
179
180     while (1) {
181         printf("1. Insert at the end\n");
182         printf("2. Insert at the beginning\n");
183         printf("3. Insert at a specific position\n");
184         printf("4. Check if the doubly linked list is empty\n");
185         printf("5. Delete from the beginning\n");
186         printf("6. Delete from the end\n");
187         printf("7. Delete from a specific position\n");
188         printf("8. Display the doubly linked list\n");
189         printf("9. Quit\n");
190         printf("Enter your choice: ");
191         scanf("%d", &choice);
```

```
193     switch (choice) {
194         case 1:
195             printf("Enter data to insert at the end: ");
196             scanf("%d", &data);
197             insertAtEnd(&head, data);
198             break;
199
200         case 2:
201             printf("Enter data to insert at the beginning: ");
202             scanf("%d", &data);
203             insertAtStart(&head, data);
204             break;
205
206         case 3:
207             printf("Enter data to insert: ");
208             scanf("%d", &data);
209             printf("Enter the position to insert at: ");
210             scanf("%d", &position);
211             insertAtIndex(&head, data, position);
212             break;
213
214         case 4:
215             if (isEmpty(head)) {
216                 printf("Doubly linked list is empty.\n");
217             } else {
218                 printf("Doubly linked list is not empty.\n");
219             }
220             break;
221
222         case 5:
223             deleteAtStart(&head);
224             break;
```



```

226         case 6:
227             deleteAtEnd(&head);
228             break;
229
230         case 7:
231             printf("Enter the position to delete from: ");
232             scanf("%d", &position);
233             deleteAtIndex(&head, position);
234             break;
235
236         case 8:
237             displayList(head);
238             break;
239
240         default:
241             printf("Invalid choice. Please try again.\n");
242     }
243 }
244
245 return 0;
246 }

```

Circular LinkedList:

```

4 // Define a structure for a node in the circular linked list
5 struct Node {
6     int data;
7     struct Node* next;
8 };
9
10 // Function to create a new node
11 struct Node* createNode(int data) {
12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13     newNode->data = data;
14     newNode->next = NULL;
15     return newNode;
16 }
17
18 // Function to insert data at the end of the circular linked list
19 void insertAtEnd(struct Node** head, int data) {
20     struct Node* newNode = createNode(data);
21     if (*head == NULL) {
22         *head = newNode;
23         newNode->next = *head;
24     } else {
25         struct Node* tail = (*head)->next;
26         while (tail->next != *head) {
27             tail = tail->next;
28         }
29         tail->next = newNode;
30         newNode->next = *head;
31     }
32 }
33

```

```

34 // Function to insert data at the beginning of the circular linked list
35 void insertAtStart(struct Node** head, int data) {
36     struct Node* newNode = createNode(data);
37     if (*head == NULL) {
38         *head = newNode;
39         newNode->next = newNode;
40     } else {
41         struct Node* tail = (*head)->next;
42         while (tail->next != *head) {
43             tail = tail->next;
44         }
45         newNode->next = *head;
46         tail->next = newNode;
47         *head = newNode;
48     }
49 }

```

```

51 // Function to insert data at a specific index position
52 void insertAtIndex(struct Node** head, int data, int position) {
53     if (position < 0) {
54         printf("Invalid position\n");
55         return;
56     }
57
58     if (position == 0) {
59         insertAtStart(head, data);
60         return;
61     }
62
63     struct Node* newNode = createNode(data);
64     if (*head == NULL) {
65         printf("List is empty. Insertion at the specified position is not possible.\n");
66         free(newNode);
67         return;
68     }
69
70     struct Node* current = *head;
71     int currentPos = 0;
72     while (current != *head && currentPos < position - 1) {
73         current = current->next;
74         currentPos++;
75     }
76
77     if (current == *head) {
78         printf("Invalid position\n");
79         free(newNode);
80     } else {
81         newNode->next = current->next;
82         current->next = newNode;
83     }
84 }
85

```

```

86 // Function to check if the circular linked list is empty
87 int isEmpty(struct Node* head) {
88     return head == NULL;
89 }
90
91 // Function to delete data from the start of the circular linked list
92 void deleteAtStart(struct Node** head) {
93     if (*head == NULL) {
94         printf("Circular linked list is empty. Nothing to delete.\n");
95         return;
96     }
97
98     struct Node* tail = (*head)->next;
99     while (tail->next != *head) {
100         tail = tail->next;
101     }
102
103     if (*head == tail) {
104         free(*head);
105         *head = NULL;
106     } else {
107         struct Node* temp = *head;
108         *head = (*head)->next;
109         tail->next = *head;
110         free(temp);
111     }
112 }
113

```

```

114 // Function to delete data from the end of the circular linked list
115 void deleteAtEnd(struct Node** head) {
116     if (*head == NULL) {
117         printf("Circular linked list is empty. Nothing to delete.\n");
118         return;
119     }
120
121     struct Node* current = *head;
122     struct Node* prev = NULL;
123     while (current->next != *head) {
124         prev = current;
125         current = current->next;
126     }
127
128     if (prev == NULL) {
129         free(*head);
130         *head = NULL;
131     } else {
132         prev->next = *head;
133         free(current);
134     }
135 }
136

```

```

137 // Function to delete data at a specific index position
138 void deleteAtIndex(struct Node** head, int position) {
139     if (position < 0) {
140         printf("Invalid position\n");
141         return;
142     }
143
144     if (position == 0) {
145         deleteAtStart(head);
146         return;
147     }
148
149     if (*head == NULL) {
150         printf("Circular linked list is empty. Nothing to delete.\n");
151         return;
152     }
153
154     struct Node* current = *head;
155     struct Node* prev = NULL;
156     int currentPos = 0;
157     while (current->next != *head && currentPos < position) {
158         prev = current;
159         current = current->next;
160         currentPos++;
161     }
162
163     if (current == *head) {
164         printf("Invalid position\n");
165     } else {
166         prev->next = current->next;
167         free(current);
168     }
169 }
170

```

```

171 // Function to print the circular linked list
172 void displayList(struct Node* head) {
173     if (head == NULL) {
174         printf("Circular linked list is empty.\n");
175         return;
176     }
177
178     struct Node* current = head;
179     do {
180         printf("%d -> ", current->data);
181         current = current->next;
182     } while (current != head);
183     printf("Head\n");
184 }
185

```

```

186 int main() {
187     struct Node* head = NULL;
188     int choice, data, position;
189
190     while (1) {
191         printf("1. Insert at the end\n");
192         printf("2. Insert at the beginning\n");
193         printf("3. Insert at a specific position\n");
194         printf("4. Check if the circular linked list is empty\n");
195         printf("5. Delete from the beginning\n");
196         printf("6. Delete from the end\n");
197         printf("7. Delete from a specific position\n");
198         printf("8. Display the circular linked list\n");
199         printf("9. Quit\n");
200         printf("Enter your choice: ");
201         scanf("%d", &choice);
202
203         switch (choice) {
204             case 1:
205                 printf("Enter data to insert at the end: ");
206                 scanf("%d", &data);
207                 insertAtEnd(&head, data);
208                 break;
209
210             case 2:
211                 printf("Enter data to insert at the beginning: ");
212                 scanf("%d", &data);
213                 insertAtStart(&head, data);
214                 break;

```

```

216             case 3:
217                 printf("Enter data to insert: ");
218                 scanf("%d", &data);
219                 printf("Enter the position to insert at: ");
220                 scanf("%d", &position);
221                 insertAtIndex(&head, data, position);
222                 break;
223
224             case 4:
225                 if (isEmpty(head)) {
226                     printf("Circular linked list is empty.\n");
227                 } else {
228                     printf("Circular linked list is not empty.\n");
229                 }
230                 break;
231
232             case 5:
233                 deleteAtStart(&head);
234                 break;
235
236             case 6:
237                 deleteAtEnd(&head);
238                 break;

```

```

240         case 7:
241             printf("Enter the position to delete from: ");
242             scanf("%d", &position);
243             deleteAtIndex(&head, position);
244             break;
245
246         case 8:
247             displayList(head);
248             break;
249
250         case 9:
251             exit(0);
252
253         default:
254             printf("Invalid choice. Please try again.\n");
255     }
256 }
257
258 return 0;
259 }

```

Circular Doubly Linked List:

```

4 // Define a structure for a node in the circular doubly linked list
5 struct Node {
6     int data;
7     struct Node* prev;
8     struct Node* next;
9 };
10
11 // Function to create a new node
12 struct Node* createNode(int data) {
13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = data;
15     newNode->prev = NULL;
16     newNode->next = NULL;
17     return newNode;
18 }
19
20 // Function to insert data at the end of the circular doubly linked list
21 void insertAtEnd(struct Node** head, int data) {
22     struct Node* newNode = createNode(data);
23     if (*head == NULL) {
24         *head = newNode;
25         newNode->next = newNode;
26         newNode->prev = newNode;
27     } else {
28         struct Node* tail = (*head)->prev;
29         tail->next = newNode;
30         newNode->prev = tail;
31         newNode->next = *head;
32         (*head)->prev = newNode;
33     }
34 }
35

```

```

36 // Function to insert data at the beginning of the circular doubly linked list
37 void insertAtStart(struct Node** head, int data) {
38     struct Node* newNode = createNode(data);
39     if (*head == NULL) {
40         *head = newNode;
41         newNode->next = newNode;
42         newNode->prev = newNode;
43     } else {
44         newNode->next = *head;
45         newNode->prev = (*head)->prev;
46         (*head)->prev->next = newNode;
47         (*head)->prev = newNode;
48         *head = newNode;
49     }
50 }
51

```

```

52 // Function to insert data at a specific index position
53 void insertAtIndex(struct Node** head, int data, int position) {
54     if (position < 0) {
55         printf("Invalid position\n");
56         return;
57     }
58     if (position == 0) {
59         insertAtStart(head, data);
60         return;
61     }
62     struct Node* newNode = createNode(data);
63     if (*head == NULL) {
64         printf("List is empty. Insertion at the specified position is not possible.\n");
65         free(newNode);
66         return;
67     }
68     struct Node* current = *head;
69     int currentPos = 0;
70     while (current != *head && currentPos < position - 1) {
71         current = current->next;
72         currentPos++;
73     }
74     if (current == *head) {
75         printf("Invalid position\n");
76         free(newNode);
77     } else {
78         newNode->next = current->next;
79         newNode->prev = current;
80         current->next->prev = newNode;
81         current->next = newNode;
82     }
83 }
84

```

```

85 // Function to check if the circular doubly linked list is empty
86 int isEmpty(struct Node* head) {
87     return head == NULL;
88 }
89

```

```

90 // Function to delete data from the start of the circular doubly linked list
91 void deleteAtStart(struct Node** head) {
92     if (*head == NULL) {
93         printf("Circular doubly linked list is empty. Nothing to delete.\n");
94         return;
95     }
96     struct Node* tail = (*head)->prev;
97     if (*head == tail) {
98         free(*head);
99         *head = NULL;
100     } else {
101         *head = (*head)->next;
102         (*head)->prev = tail;
103         tail->next = *head;
104     }
105 }
106
107 // Function to delete data from the end of the circular doubly linked list
108 void deleteAtEnd(struct Node** head) {
109     if (*head == NULL) {
110         printf("Circular doubly linked list is empty. Nothing to delete.\n");
111         return;
112     }
113
114     struct Node* tail = (*head)->prev;
115     if (*head == tail) {
116         free(*head);
117         *head = NULL;
118     } else {
119         tail->prev->next = *head;
120         (*head)->prev = tail->prev;
121         free(tail);
122     }
123 }

```

```

125 // Function to delete data at a specific index position
126 void deleteAtIndex(struct Node** head, int position) {
127     if (position < 0) {
128         printf("Invalid position\n");
129         return;
130     }
131     if (position == 0) {
132         deleteAtStart(head);
133         return;
134     }
135     if (*head == NULL) {
136         printf("Circular doubly linked list is empty. Nothing to delete.\n");
137         return;
138     }
139     struct Node* current = *head;
140     int currentPos = 0;
141     while (current != *head && currentPos < position) {
142         current = current->next;
143         currentPos++;
144     }
145     if (current == *head) {
146         printf("Invalid position\n");
147     } else {
148         current->prev->next = current->next;
149         current->next->prev = current->prev;
150         free(current);
151     }
152 }
153

```



```

154 // Function to print the circular doubly linked list
155 void displayList(struct Node* head) {
156     if (head == NULL) {
157         printf("Circular doubly linked list is empty.\n");
158         return;
159     }
160
161     struct Node* current = head;
162     do {
163         printf("%d <-> ", current->data);
164         current = current->next;
165     } while (current != head);
166     printf("Head\n");
167 }
168

```

```

169 int main() {
170     struct Node* head = NULL;
171     int choice, data, position;
172
173     while (1) {
174         printf("1. Insert at the end\n");
175         printf("2. Insert at the beginning\n");
176         printf("3. Insert at a specific position\n");
177         printf("4. Check if the circular doubly linked list is empty\n");
178         printf("5. Delete from the beginning\n");
179         printf("6. Delete from the end\n");
180         printf("7. Delete from a specific position\n");
181         printf("8. Display the circular doubly linked list\n");
182         printf("9. Quit\n");
183         printf("Enter your choice: ");
184         scanf("%d", &choice);
185
186         switch (choice) {
187             case 1:
188                 printf("Enter data to insert at the end: ");
189                 scanf("%d", &data);
190                 insertAtEnd(&head, data);
191                 break;
192
193             case 2:
194                 printf("Enter data to insert at the beginning: ");
195                 scanf("%d", &data);
196                 insertAtStart(&head, data);
197                 break;
198

```

```

199             case 3:
200                 printf("Enter data to insert: ");
201                 scanf("%d", &data);
202                 printf("Enter the position to insert at: ");
203                 scanf("%d", &position);
204                 insertAtIndex(&head, data, position);
205                 break;
206

```

```

207         case 4:
208             if (isEmpty(head)) {
209                 printf("Circular doubly linked list is empty.\n");
210             } else {
211                 printf("Circular doubly linked list is not empty.\n");
212             }
213             break;
214         case 5:
215             deleteAtStart(&head);
216             break;
217         case 6:
218             deleteAtEnd(&head);
219             break;
220         case 7:
221             printf("Enter the position to delete from: ");
222             scanf("%d", &position);
223             deleteAtIndex(&head, position);
224             break;
225         case 8:
226             displayList(head);
227             break;
228         default:
229             printf("Invalid choice. Please try again.\n");
230     }
231 }
232 return 0;
233 }

```