

```

1  #include <iostream>
2
3
4  using namespace std;
5
6
7  struct Node {
8      int data;
9      Node* prev = nullptr;
10     Node* next = nullptr;
11 };
12
13
14 Node* head = nullptr;
15
16
17 Node* newNode(int data) {
18     Node* node = new Node;
19     node->data = data;
20     return node;
21 }
22
23
24 void insertIntoEmptyList(int data) {
25     if( head != nullptr ) {
26         return;
27     }
28     Node* node = newNode(data);
29     head = node;
30     head->next = head;
31     head->prev = head;
32 }
33
34
35 void insertAtHead(int data) {
36     if(head == nullptr) {
37         insertIntoEmptyList(data);
38         return;
39     }
40     Node* node = newNode(data);
41     node->next = head;
42     node->prev = head->prev;
43     head->prev->next = node;
44     head->prev = node;
45     head = node;
46 }
47
48
49 void insertAtTail(int data) {
50     if(head == nullptr) {
51         insertIntoEmptyList(data);
52         return;
53     }
54     Node* node = newNode(data);
55     Node* tail = head->prev;
56     node->prev = tail;
57     node->next = head;
58     tail->next = node;
59     head->prev = node;
60 }
61
62
63
64 void insertAtMiddleAfter(int after, int data) {
65     if( head == nullptr ) {
66         return;
67     }
68     if(head != nullptr && head->prev != nullptr && head->prev->data == after) {
69         insertAtTail(data);
70         return;
71     }
72     Node* prev = head;
73     while(prev != head->prev) {
74         if(prev->data == after) {
75             Node* node = newNode(data);
76             node->next = prev->next;
77             node->prev = prev;
78             prev->next = node;
79             node->next->prev = node;
80             break;
81         }
82         prev = prev->next;
83     }
84 }
85
86
87
88 void deleteFromHead() {
89     if(head == nullptr) {
90         return;
91     }
92     Node* node = head;
93     head = node->next;
94     if (head != node) {
95         //list will not be empty after deleting head element
96         head->prev = node->prev;
97         node->prev->next = head;
98     }
99     else {
100        //list will be empty after deleting the only element.
101        head = nullptr;
102    }
103    delete node;
104 }
105
106

```

```

107 void deleteFromTail() {
108     if(head == nullptr) {
109         return;
110     }
111     Node* tail = head->prev;
112     head->prev = tail->prev;
113     if(head->prev != tail) {
114         //list will not be empty after deleting the tail element.
115         head->prev->next = head;
116     }
117     else {
118         //list will be empty after deleting the only element.
119         head = nullptr;
120     }
121     delete tail;
122 }
123
124
125
126 void deleteFromMiddle(int data) {
127     if(head == nullptr) {
128         return;
129     }
130     if (head->data == data) {
131         deleteFromHead();
132         return;
133     }
134     if(head->prev->data == data) {
135         deleteFromTail();
136         return;
137     }
138     Node* prev = head;
139     while(prev != head->prev) {
140         if(prev->next != nullptr && prev->next->data == data) {
141             Node* node = prev->next;
142             prev->next = node->next;
143             node->next->prev = prev;
144             delete node;
145             break;
146         }
147         prev = prev->next;
148     }
149 }
150
151
152
153 void showForward() {
154     if(head == nullptr) {
155         cout<<endl;
156         return;
157     }
158     Node* node = head;
159     while(node != head->prev) {
160         cout<<" "<<node->data;
161         node = node->next;
162     }
163     cout<<" "<<node->data<<endl;
164 }
165
166
167
168 void showBackward() {
169     if(head == nullptr) {
170         cout<<endl;
171         return;
172     }
173
174     Node* node = head->prev;
175     while(node != head) {
176         cout<<" "<<node->data;
177         node = node->prev;
178     }
179     cout<<" "<<node->data<<endl;
180 }
181
182
183
184 int main() {
185
186     insertAtHead(10);
187     insertAtHead(20);
188     insertAtHead(30);
189     insertAtHead(40);
190     insertAtHead(50);
191     showForward();
192     showBackward();
193
194     insertAtMiddleAfter(30, 60);
195     showForward();
196     showBackward();
197
198     insertAtMiddleAfter(10, 70);
199     showForward();
200     showBackward();
201
202     insertAtTail(80);
203     showForward();
204     showBackward();
205
206
207     deleteFromMiddle(60);
208     showForward();
209     showBackward();
210
211     deleteFromMiddle(70);
212     showForward();

```

```
213     showBackward();
214
215     deleteFromHead();
216     showForward();
217     showBackward();
218
219     while(head != nullptr) {
220         deleteFromTail();
221         showForward();
222         showBackward();
223     }
224     return 0;
225 }
```