

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 typedef union {
7     int    iValue;
8     char   cValue;
9     char*  cp;
10    char   cbuff[20];
11 } NodeData;    // linked_node data element
12
13 typedef struct {
14     void *prev;
15     void *next;
16 } LinkInfo;    // has info on links to and from the node
17
18 typedef struct {
19     NodeData data;
20     LinkInfo linkTo;
21 } Node;    // linked list node element
22
23 Node *lhead = NULL;
24 Node *ltail = NULL;
25
26 void create();
27 void insert();
28 void ndelete();
29 void search();
30 void ndisplay();
31 void find();
32
33 int main()
34 {
35     int choice, n;
36
37     printf("\n Enter the initial no. of inputs:");
38     scanf("%d",&n);
39
40     int i;
41     for (i=1;i<=n;i++)
42         create();
43     printf("\n1.Insert\n2.Delete\n3.Display\n"
44           "4.Search\n5.Find\n6.Exit");
45
46
47     do
48     {
49         printf("\n Enter your choice: ");
50         scanf("%d",&choice);
51         switch (choice)
52         {
53             {
54                 case 1:insert();break;
55                 case 2:ndelete();break;
56                 case 3:ndisplay();break;
```

```
57     case 4:search();break;
58     case 5:find();break;
59     case 6:exit(0);break;
60 }
61 } while (choice !=6);
62
63 return 0;
64 }
65
66
67 Node* createNode(int val) {
68     Node* new = malloc(sizeof(Node));
69     new->data.iValue = val;
70     new->linkTo.prev = NULL;
71     new->linkTo.next = NULL;
72
73     return new;
74 }
75
76 int getValue() {
77     int val;
78     printf("\n Enter the data : ");
79     scanf("%d",&val);
80     printf("%d", val);
81     return val;
82 }
83
84 void create()
85 {
86     int val = getValue();
87
88     Node* new = createNode(val);
89
90     if (lhead == NULL) {
91         lhead = new;
92         ltail = lhead;
93     }
94     else {
95         new->linkTo.prev = ltail;
96         ltail->linkTo.next = new;
97         ltail = new;
98     }
99
100 }
101
102 Node* traverse(Node* np, int count)
103 {
104     for (; count > 0 && np != NULL; count--) {
105         // printf ("count:%d value:%d", count, np->data.iValue);
106         np = np->linkTo.next;
107     }
108     // printf ("Returning np: %p", np);
109     return np;
110
111 }
112
113
```

```
114 int getPosition() {
115     int pos;
116     printf("\n Enter the position: ");
117     scanf("%d",&pos);
118     printf("%d", pos);
119     pos = pos-1;
120     return pos < 0 ? 0 : pos; // return a non-negative number
121 }
122
123 void insert()
124 {
125     int pos = getPosition();
126     int val = getValue();
127
128     Node* iter = traverse(lhead, pos);
129     // printf ("iter: %p ", iter);
130     if (iter == NULL) {
131         // something is wrong, report it and quit
132         printf("\n No such position");
133     } else {
134         Node* new = createNode(val);
135
136         // insert the new node at the 'iter' location
137         // printf ("value at node: %d ", iter->data.iValue);
138         Node* locatedNode = iter;
139         Node* prevNode = locatedNode->linkTo.prev;
140
141         new->linkTo.next = locatedNode; // `located` comes after `new`
142         locatedNode->linkTo.prev = new; // and vice-versa
143
144         new->linkTo.prev = prevNode; // connect with prev
145         prevNode->linkTo.next = new; // and vice-versa
146
147     }
148 }
149
150 }
151
152 void ndelete()
153 {
154     int pos = getPosition();
155
156     Node *nodeToDelete, *prevNode, *nextNode;
157
158     Node* iter = traverse(lhead, pos);
159     if (iter == NULL) {
160         printf("\n No such position");
161         return;
162     }
163     prevNode = iter->linkTo.prev;
164     nextNode = iter->linkTo.next;
165
166     if (iter == lhead) {
167         if (nextNode != NULL) {
168             lhead->data.iValue = nextNode->data.iValue;
169             lhead->linkTo.next = nextNode->linkTo.next;
170             free(nextNode);
```

```
171     }
172     else {
173         free(lhead);
174         printf("\nHead is Free! List is empty!\n");
175         lhead = NULL;
176         ltail = NULL;
177     }
178
179 } else {
180     nodeToDelete = iter;
181     prevNode->linkTo.next = nextNode;
182     if (nextNode != NULL) {
183         nextNode->linkTo.prev = prevNode;
184     }
185
186     printf("\n The deleted data is %d",nodeToDelete->data.iValue);
187     free(nodeToDelete);
188 }
189 }
190
191
192 void ndisplay()
193 {
194     Node* iter = lhead;
195     // printf ("Inside display...%p\n", iter);
196
197     if (iter == NULL) printf("\nThe List is empty");
198     else {
199         printf("\n  The list contains : ");
200         for (; iter != NULL; iter = iter->linkTo.next)
201             printf("\t %d", iter->data.iValue);
202     }
203
204 }
205
206
207 void find()
208 {
209     int pos = getPosition();
210
211     Node* iter = traverse(lhead, pos);
212     if (iter == NULL)
213     {
214         printf("\n No such position");
215     }
216     else {
217         Node* prevNode = iter->linkTo.prev;
218         Node* nextNode = iter->linkTo.next;
219
220         int iterValue = iter->data.iValue;
221         int prevValue = (prevNode != NULL)? prevNode->data.iValue : -1;
222         int nextValue = (nextNode != NULL)? nextNode->data.iValue : -1;
223
224         printf("\nThe data around the position are "
225             "%d %d %d\n", prevValue, iterValue, nextValue);
226     }
227 }
```

```
228  
229 void search() {  
230  
231 }
```