

SEM 3\Exp4\CDLL.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Node structure for the circular doubly linked list
5  struct Node
6  {
7      int data;
8      struct Node *next;
9      struct Node *prev;
10 };
11
12 // Insert a node at the end of the circular doubly linked list
13 void insert(struct Node **head_ref, int new_data)
14 {
15     struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
16     new_node->data = new_data;
17
18     if (*head_ref == NULL)
19     {
20         new_node->next = new_node;
21         new_node->prev = new_node;
22         *head_ref = new_node;
23         return;
24     }
25
26     struct Node *last = (*head_ref)->prev;
27
28     new_node->next = *head_ref;
29     (*head_ref)->prev = new_node;
30     new_node->prev = last;
31     last->next = new_node;
32 }
33
34 // Display the circular doubly linked list
35 void display(struct Node *head)
36 {
37     if (head == NULL)
38     {
39         printf("List is empty.\n");
40         return;
41     }
42
43     struct Node *temp = head;
44     printf("Traversal in forward direction:\n");
45     do
46     {
47         printf("%d ", temp->data);
48         temp = temp->next;
49     } while (temp != head);
50     printf("\n");
51 }
```

```
52     printf("Traversal in reverse direction:\n");
53     temp = head->prev;
54     do
55     {
56         printf("%d ", temp->data);
57         temp = temp->prev;
58     } while (temp->next != head);
59     printf("\n");
60 }
61
62 // Delete a node from the circular doubly linked list
63 void deleteNode(struct Node **head_ref, int key)
64 {
65     if (*head_ref == NULL)
66         return;
67
68     struct Node *current = *head_ref;
69
70     while (current->data != key)
71     {
72         current = current->next;
73         if (current == *head_ref)
74         {
75             printf("Element %d not found in the list.\n", key);
76             return;
77         }
78     }
79
80     if (current->next == *head_ref && current->prev == *head_ref)
81     {
82         *head_ref = NULL;
83         free(current);
84         return;
85     }
86
87     if (current == *head_ref)
88     {
89         struct Node *last = (*head_ref)->prev;
90         *head_ref = current->next;
91         last->next = *head_ref;
92         (*head_ref)->prev = last;
93         free(current);
94         return;
95     }
96
97     current->prev->next = current->next;
98     current->next->prev = current->prev;
99
100     free(current);
101 }
102
103 void search(struct Node *head, int key)
104 {
105     if (head == NULL)
```

```
106     {
107         printf("List is empty.\n");
108         return;
109     }
110
111     struct Node *temp = head;
112     int pos = 0;
113
114     do
115     {
116         if (temp->data == key)
117         {
118             printf("Element %d found at position %d\n", key, pos);
119             return;
120         }
121         temp = temp->next;
122         pos++;
123     } while (temp != head);
124
125     printf("Element %d not found in the list\n", key);
126 }
127
128 int count(struct Node *head)
129 {
130     if (head == NULL)
131         return 0;
132
133     struct Node *temp = head;
134     int count = 0;
135
136     do
137     {
138         count++;
139         temp = temp->next;
140     } while (temp != head);
141
142     return count;
143 }
144
145 int main()
146 {
147     struct Node *head = NULL;
148     int choice, value, key;
149
150     printf("\nCircular Doubly Linked List Operations:\n");
151     printf("1. Insert\n");
152     printf("2. Display\n");
153     printf("3. Delete\n");
154     printf("4. Search\n");
155     printf("5. Count\n");
156     printf("6. Exit\n");
157
158     while (1)
159     {
```

```
160
161     printf("Enter your choice: ");
162     scanf("%d", &choice);
163
164     switch (choice)
165     {
166     case 1:
167         printf("Enter the value to insert: ");
168         scanf("%d", &value);
169         insert(&head, value);
170         printf("\n");
171         break;
172     case 2:
173         display(head);
174         printf("\n");
175         break;
176     case 3:
177         printf("Enter the value to delete: ");
178         scanf("%d", &key);
179         deleteNode(&head, key);
180         printf("\n");
181         break;
182     case 4:
183         printf("Enter the value to search: ");
184         scanf("%d", &key);
185         search(head, key);
186         printf("\n");
187         break;
188     case 5:
189         printf("The number of nodes in the list: %d\n", count(head));
190         printf("\n");
191         break;
192     case 6:
193         exit(0);
194     default:
195         printf("Invalid choice!\n");
196     }
197 }
198
199 return 0;
200 }
201
```