

8.1.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  v typedef struct node {
5      int data;
6      struct node* next;
7  } Node;
8
9  v Node* createNode(int data) {
10     —> Node* newNode = (Node*) malloc(sizeof(Node));
11     v —> if (!newNode) {
12         —> —> printf("Memory error\n");
13         —> —> return NULL;
14     —> }
15     —> newNode->data = data;
16     —> newNode->next = NULL;
17     —> return newNode;
18 }
19 v Node* insertNode(Node* head, int data, int position) {
20     —> Node* newNode = createNode(data);
21     v —> if (position == 0) {
22     v —> —> if (head == NULL) {
23         —> —> —> return newNode;
24     —> —> }
25     —> —> Node* temp = head;
26     v —> —> while (temp->next != NULL) {
27         —> —> —> temp = temp->next;
28     —> —> }
29     —> —> temp->next = newNode;
```

```

30     —>|—>|newNode->next == head;
31     —>|—>|return newNode;
32     —>|}
33     —>|Node* prev == head;
34     —>|int i;
35 v —>|for (i == 0; i < position - 1 && prev != NULL; i++) {
36     —>|—>|prev == prev->next;
37     —>|}
38 v —>|if (i != position - 1 || prev == NULL) {
39     —>|—>|printf("Position out of range\n");
40     —>|—>|free(newNode);
41     —>|—>|return head;
42     —>|}
43     —>|newNode->next == prev->next;
44     —>|prev->next == newNode;
45     —>|return head;
46     }
47 v Node* deleteNode(Node* head, int position) {
48 v —>|if (head == NULL) {
49     —>|—>|printf("List is empty\n");
50     —>|—>|return NULL;
51     —>|}
52     —>|Node* prev == head, *curr == head->next;
53 v —>|if (position == 0) {
54 v —>|—>|if (head->next == head) {
55     —>|—>|—>|free(head);
56     —>|—>|—>|return NULL;
57     —>|—>|}
58 v —>|—>|while (prev->next != head) {
59     —>|—>|—>|prev == prev->next;

```

```

60     —> —> }
61     —> —> prev->next = curr;
62     —> —> free(head);
63     —> —> return curr;
64     —> }
65     —> int i;
66     —>
67 v —> —> for (i = 0; i < position - 1 && curr != head; i++) {
68     —> —> prev = curr;
69     —> —> curr = curr->next;
70     —> }
71 v —> —> if (i != position - 1 || curr == head) {
72     —> —> printf("Position out of range\n");
73     —> —> return head;
74     —> }
75     —> prev->next = curr->next;
76     —> free(curr);
77     —> return head;
78 }
79 v int searchNode(Node* head, int key) {
80     —> Node* temp = head;
81     —> int position = 0;
82 v —> do {
83 v —> —> if (temp->data == key) {
84     —> —> —> return position;
85     —> —> }
86     —> —> temp = temp->next;
87     —> —> position++;
88     —> } while (temp != head);
89     —> return -1;

```

```

90     }
91     v void traverseList(Node* head) {
92         —>Node* temp = head;
93     v —> if (head != NULL) {
94     v —> —>do {
95         —> —> —>printf("%d --> ", temp->data);
96         —> —> —>temp = temp->next;
97         —> —>} while (temp != head);
98         —>}
99         —>printf("\n");
100     }
101     v int main() {
102         —>Node* head = NULL;
103         —>int option, position, element, result;
104     v —>do {
105         —> —>printf("1. Insert At specified position\n");
106         —> —>printf("2. Traverse the List\n");
107         —> —>printf("3. Search\n");
108         —> —>printf("4. Delete\n");
109         —> —>printf("5. Exit\n");
110         —> —>printf("Enter your option: ");
111         —> —>scanf("%d", &option);
112     v —> —>switch (option) {
113         —> —> —>case 1:
114             —> —> —>printf("Enter a position: ");
115             —> —> —>scanf("%d", &position);
116             —> —> —>printf("Enter an element: ");
117             —> —> —>scanf("%d", &element);
118             —> —> —>head = insertNode(head, element, position);

```

```

119     break;
120     case 2:
121         printf("The elements in CLL are: ");
122         traverseList(head);
123         break;
124     case 3:
125         printf("Enter search element: ");
126         scanf("%d", &element);
127         result = searchNode(head, element);
128         v if (result != -1) {
129             printf("The given element %d is found at position: %d\n", element, result);
130         } else {
131             printf("Element not found\n");
132         }
133         break;
134     case 4:
135         printf("Enter position: ");
136         scanf("%d", &position);
137         head = deleteNode(head, position);
138         v if (head != NULL) {
139             printf("The deleted element from CLL: %d\n", position);
140         }
141         break;
142     case 5:
143         exit(0);
144     default:
145         printf("Invalid option\n");
146     }
147 } while (1);

```

8.1.2

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  struct node
4  v {
5      —>struct node *prev;
6      —>int data;
7      —>struct node *next;
8      —>
9  };
10 struct node *root = NULL;
11 void in_beg()
12 v {
13     —>struct node *p,*temp;
14     —>temp = (struct node*)malloc(sizeof(struct node));
15     —>printf("Enter an element: ");
16     —>scanf("%d",&temp->data);
17     —>temp->next = NULL;
18     —>temp->prev = NULL;
19     —>if(root == NULL)
20 v —>{
21     —>—>root = temp;
22     —>}
23     —>else
24 v —>{
25     —>—>
26     —>—>temp->next = root;
27     —>—>root->prev = temp;
28     —>—>root = temp;
29     —>}

```

```

30     }
31     void del_beg()
32     v {
33         —> struct node *p,*q;
34
35         —> if(root==NULL)
36     v —> {
37         —> —> printf("Double Linked List is empty so deletion is not
           possible\n");
38         —> }
39         —> else
40     v —> {
41         —> —> p=.root;
42         —> —> root=.root->next;
43         —> —> printf("The deleted element from DLL : %d\n",p->data);
44         —> —> free(p);
45         —> —>
46         —> }
47     }
48     void search()
49     v {
50         —> struct node *p,*q;
51         —> int kele,pos=1,flag=0;
52         —> printf("Enter search element: ");
53         —> scanf("%d",&kele);
54         —> p=.root;
55         —> while(p!=NULL)
56     v —> {
57         —> —> if(p->data==kele)
58     v —> —> {

```

```

59     —> —> —> printf("The given element %d is found at position : :
    %d\n", kele, pos);
60     —> —> —> flag = 1;
61     —> —> }
62     —> —> p = p->next;
63     —> —> pos++;
64     —> }
65     —> if(flag==0)
66 v —> {
67     —> —> printf("The given element %d is not found in the given
    DLL\n", kele);
68     —> }
69     }
70     void display()
71 v {
72     —> struct node *p;
73     —> if(root==NULL)
74 v —> {
75     —> —> printf("Double Linked List is empty\n");
76     —> }
77     —> else
78 v —> {
79     —> —> p = root;
80     —> —> printf("The elements in DLL are : ");
81     —> —> while(p!=NULL)
82 v —> —> {
83     —> —> —> printf("%d<--> ", p->data);
84     —> —> —> p = p->next;
85     —> —> }

```



```

86     —>—>—>printf("NULL\n");
87     —>}
88 }
89 int main()
90 v {
91     —>int op;
92     —>while(1)
93 v —>{
94     —>—>—>printf("1.Insert At Begin\n2.Delete at Begin\n3.Search an element
Position\n4.Traverse the List\n5.Exit\n");
95     —>—>—>printf("Enter your option : ");
96     —>—>—>scanf("%d",&op);
97     —>—>—>switch(op)
98 v —>—>—>{
99     —>—>—>—>case 1:
100     —>—>—>—>in_beg();
101     —>—>—>—>break;
102     —>—>—>—>
103     —>—>—>—>case 2:
104     —>—>—>—>del_beg();
105     —>—>—>—>break;
106     —>—>—>—>
107     —>—>—>—>case 3:
108     —>—>—>—>search();
109     —>—>—>—>break;
110     —>—>—>—>
111     —>—>—>—>case 4:
112     —>—>—>—>display();
113     —>—>—>—>break;
114     —>—>—>—>

```

```

115     —>—>—>—>case 5:
116     —>—>—>—>exit(0);
117     —>—>—>—>break;
118     —>—>—>—>
119     —>—>—>—>
120     —>—>—>}
121     —>}
122 }
123
124

```

8.2.1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  v —>—> typedef struct node{
5      —>int data;
6      —>struct node* next;
7  } Node;
8
9  v Node* createNode(int data){
10     —>Node* newNode = (Node*)malloc(sizeof(Node));
11  v —>if (!newNode){
12     —>—>printf("Memory error\n");
13     —>—>return NULL;
14     —>}
15     —>newNode->data = data;
16     —>newNode->next = NULL;
17     —>return newNode;
18 }
19 v Node* insertNode(Node* head, int data){
20     —>Node* newNode = createNode(data);
21     —>if (!newNode) return head;
22  v —>if (head == NULL){
23     —>—>newNode->next = newNode;
24     —>—>head = newNode;
25  v —>} else {
26     —>—>Node* current = head;
27  v —>—>while (current->next != head){
28     —>—>—>current = current->next;
29     —>—>}

```

```

30     —>—>current->next := newNode;
31     —>—>newNode->next := head;
32     —>}
33     —>return head;
34 }
35 v void printList(Node* head) {
36 v —>—>if (head == NULL) {
37     —>—>printf("List is empty\n");
38     —>—>return;
39     —>}
40     —>Node* current := head;
41 v —>do {
42     —>—>printf("%d ", current->data);
43     —>—>current := current->next;
44     —>} while (current != head);
45     —>printf("\n");
46 }
47 v Node* reverseList(Node* head) {
48     —>—>if (head == NULL || head->next == head) return head;
49     —>Node* prev := head;
50     —>Node* current := head->next;
51     —>Node* next := current->next;
52 v —>while (current != head) {
53     —>—>current->next := prev;
54     —>—>prev := current;
55     —>—>current := next;
56     —>—>next := (next->next == head) ? head : next->next;
57     —>}
58     —>head->next := prev;

```

```

59     —> head = prev;
60     —> return head;
61 }
62 v int main() {
63     —> Node* head = NULL;
64
65     —> int n, data;
66     —> printf("Enter the number of elements: ");
67     —> scanf("%d", &n);
68     —> printf("Enter the elements into the circular linked list: ");
69 v —> for (int i = 0; i < n; i++) {
70     —> —> scanf("%d", &data);
71     —> —> head = insertNode(head, data);
72     —> }
73
74     —> printf("The elements in the list before reversing are: ");
75     —> printList(head);
76     —> head = reverseList(head);
77     —> printf("The elements in the list after reversing are: ");
78     —> printList(head);
79
80 }

```

8.2.2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  v typedef struct node {
5      —>int data;
6      —>struct node* prev;
7      —>struct node* next;
8  } Node;
9
10 v Node* createNode(int data) {
11     —>Node* newNode = (Node*)malloc(sizeof(Node));
12     v —>if (!newNode) {
13         —>—>printf("Memory error\n");
14         —>—>return NULL;
15     }
16     —>newNode->data = data;
17     —>newNode->prev = NULL;
18     —>newNode->next = NULL;
19     —>return newNode;
20 }
21 v Node* insertNode(Node* head, int data) {
22     —>Node* newNode = createNode(data);
23     —>if (!newNode) return head;
24
25     v —>if (head == NULL) {
26         —>—>head = newNode;
27     v —>} else {
28         —>—>Node* current = head;
29     v —>—>while (current->next != NULL) {

```

```

30     —> —> —>current = current->next;
31     —> —> }
32     —> —> current->next = newNode;
33     —> —> newNode->prev = current;
34     —> }
35     —> return head;
36 }
37 v void printEvenPositionElements(Node* head) {
38 v     —> if (head == NULL) {
39     —> —> printf("List is empty\n");
40     —> —> return;
41     —> }
42     —> Node* current = head;
43     —> int position = 1;
44 v     —> while (current != NULL) {
45 v     —> —> if (position % 2 == 0) {
46     —> —> —> printf("%d ", current->data);
47     —> —> }
48     —> —> current = current->next;
49     —> —> position++;
50     —> }
51     —> printf("\n");
52 }
53 v int main() {
54     —> Node* head = NULL;
55     —> int n, data;
56     —> printf("Enter the no. of elements in the Doubly linked list: ");
57     —> scanf("%d", &n);
58     —> printf("Enter the elements into Doubly linked list: ");
59 v     —> for (int i = 0; i < n; i++) {
60         —> —> scanf("%d", &data);
61         —> —> head = insertNode(head, data);
62         —> }
63     —> printf("Elements of Doubly linked list at even position are:");
64     —> printEvenPositionElements(head);
65
66 }

```

8.2.3

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int data;
7      struct node *next;
8      struct node *prev;
9  };
10 struct node *head=NULL;
11
12 void create(int n)
13 {
14     struct node *newnode,*ptr;
15     newnode=(struct node*)malloc(sizeof(struct node));
16     newnode->data=n;
17     newnode->next=NULL;
18     newnode->prev=NULL;
19     if(head==NULL)
20     {
21         head=ptr=newnode;
22     }
23     else
24     {
25         ptr=head;
26         while(ptr->next!=NULL)
27         {
28             ptr=ptr->next;
29         }

```

```

30     —> —> —> ptr->next=newnode;
31     —> —> —> newnode->prev=ptr;
32     —> —> }
33     —> —>
34     }
35
36     void deleten()
37     v {
38     —> —> —> struct node *ptr,*temp;
39     —> —> —> ptr=head;
40     —> —> —> temp=NULL;
41     —> —> —> int c=1;
42     —> —>
43     —> —> —> while(ptr!=NULL)
44     v —> —> {
45     —> —> —> if(c%2==0)
46     v —> —> —> {
47     —> —> —> —> temp=ptr;
48     —> —> —> —> ptr=ptr->next;
49     —> —> —> —> if(temp->prev!=NULL)
50     v —> —> —> —> {
51     —> —> —> —> —> temp->prev->next=ptr;
52     —> —> —> —> }
53     —> —> —> else
54     v —> —> —> {
55     —> —> —> —> head=ptr;
56     —> —> —> —> }
57     —> —> —> if(ptr!=NULL)
58     v —> —> —> {
59     —> —> —> —> ptr->prev=temp->prev;

```



```

60     —> —> }
61     —> —> free(temp);
62     —> —> }
63     —> —> else
64 v —> —> {
65     —> —> —> ptr=ptr->next;
66     —> —> }
67     —> —> c++;
68     —> —> }
69     }
70     void print()
71 v {
72     —> struct node *ptr;
73     —> ptr=head;
74     —> while(ptr!=NULL)
75 v —> {
76     —> —> printf("%d.",ptr->data);
77     —> —> ptr=ptr->next;
78     —> }
79     }
80     int main()
81 v {
82     —> int l,i,n;
83     —> printf("Enter the number of elements: ");
84     —> scanf("%d",&l);
85     —> printf("Enter the elements into the Doubly linked list: ");
86     —> for(i=0;i<l;i++)
87 v —> {
88     —> —> scanf("%d",&n);

```

```

89     —> —> create(n);
90     —> }
91     —> printf("Original Doubly linked list: ");
92     —> print();
93     —> printf("\n");
94     —> printf("Elements after deletion are: ");
95     —> deleten();
96     —> print();
97     —>
98
99     }

```