8.1.1

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
int data;
struct node* next;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory error\n");
        return NULL;
    }
    newNode->data = data;
    newNode->next = newNode;
    return newNode;
}
Node* insertNode(Node* head, int data, int position) {
    Node* newNode = createNode(data);
    if (position == 0) {
        if (head == NULL) {
            return newNode;
        }
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
```

```
        newNode->next = head;
        return newNode;
    }
    Node* prev = head;
    int i;
    for (i = 0; i < position - 1 && prev != NULL; i++) {
        prev = prev->next;
    }
    if (i != position - 1 || prev==NULL) {
        printf("Position out of range\n");
        free(newNode);
        return head;
    }
    newNode->next = prev->next;
    prev->next = newNode;
    return head;
}
Node* deleteNode(Node* head, int position) {
    if (head == NULL) {
        printf("List is empty\n");
        return NULL;
    }
    Node* prev = head, *curr = head->next;
    if (position == 0) {
        if (head->next == head) {
            free(head);
            return NULL;
        }
        while (prev->next != head) {
            prev = prev->next;
```

```
    }
    prev->next = curr;
    free(head);
    return curr;
  }
  int i;

  for (i = 0; i < position - 1 && curr != head; i++) {
    prev = curr;
    curr = curr->next;
  }
  if (i != position - 1 || curr == head) {
    printf("Position out of range\n");
    return head;
  }
  prev->next = curr->next;
  free(curr);
  return head;
}
int searchNode(Node* head, int key) {
  Node* temp = head;
  int position = 0;
  do {
    if (temp->data == key) {
      return position;
    }
    temp = temp->next;
    position++;
  } while (temp != head);
  return -1;
```

```
90      }
91      void traverseList(Node* head) {
92          Node* temp = head;
93          if (head != NULL) {
94              do {
95                  printf("%d --> ", temp->data);
96                  temp = temp->next;
97              } while (temp != head);
98          }
99          printf("\n");
100     }
101     int main() {
102         Node* head = NULL;
103         int option, position, element, result;
104         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             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);
```

```
            break;
        case 2:
            printf("The elements in CLL are: ");
            traverseList(head);
            break;
        case 3:
            printf("Enter search element: ");
            scanf("%d", &element);
            result = searchNode(head, element);
            if (result != -1) {
                printf("The given element %d is found at position: %d\n", element, result);
            } else {
                printf("Element not found\n");
            }
            break;
        case 4:
            printf("Enter position: ");
            scanf("%d", &position);
            head = deleteNode(head, position);
            if (head != NULL) {
                printf("The deleted element from CLL: %d\n", position);
            }
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid option\n");
    }
} while (1);
```

8.1.2

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    int data;
    struct node *next;

};
struct node *root = NULL;
void in_beg()
{
    struct node *p,*temp;
    temp = (struct node*)malloc(sizeof(struct node));
    printf("Enter an element: ");
    scanf("%d",&temp->data);
    temp->next = NULL;
    temp->prev = NULL;
    if(root == NULL)
    {
        root = temp;
    }
    else
    {

        temp->next = root;
        root->prev = temp;
        root = temp;
    }
```

```
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      {
67      printf("The given element %d is not found in the given DLL\n",kele);
68      }
69      }
70      void display()
71      {
72      struct node *p;
73      if(root==NULL)
74      {
75      printf("Double Linked List is empty\n");
76      }
77      else
78      {
79      p = root;
80      printf("The elements in DLL are: ");
81      while(p!=NULL)
82      {
83      printf("%d <--> ",p->data);
84      p = p->next;
85      }
```

```
86      printf("NULL\n");
87      }
88      }
89      int main()
90      {
91      int op;
92      while(1)
93      {
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      {
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

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node{
int data;
struct node* next;
} Node;

Node* createNode(int data) {
Node* newNode = (Node*)malloc(sizeof(Node));
if (!newNode) {
printf("Memory error\n");
return NULL;
}
newNode->data = data;
newNode->next = NULL;
return newNode;
}
Node* insertNode(Node* head, int data) {
Node* newNode = createNode(data);
if (!newNode) return head;
if (head == NULL) {
newNode->next = newNode;
head = newNode;
} else {
Node* current = head;
while (current->next != head) {
current = current->next;
}
```

```
30      current->next = newNode;
31      newNode->next = head;
32    }
33    return head;
34  }
35  void printList(Node* head) {
36    if (head == NULL) {
37      printf("List is empty\n");
38      return;
39    }
40    Node* current = head;
41    do {
42      printf("%d ", current->data);
43      current = current->next;
44    } while (current != head);
45    printf("\n");
46  }
47  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    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;
```

```
    head = prev;
    return head;
}
int main() {
    Node* head = NULL;

    int n, data;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements into the circular linked list: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        head = insertNode(head, data);
    }

    printf("The elements in the list before reversing are: ");
    printList(head);
    head = reverseList(head);
    printf("The elements in the list after reversing are: ");
    printList(head);

}
```

8.2.2

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    struct node* prev;
    struct node* next;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory error\n");
        return NULL;
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
Node* insertNode(Node* head, int data) {
    Node* newNode = createNode(data);
    if (!newNode) return head;

    if (head == NULL) {
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != NULL) {
```

```
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
    return head;
}
void printEvenPositionElements(Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    Node* current = head;
    int position = 1;
    while (current != NULL) {
        if (position % 2 == 0) {
            printf("%d ", current->data);
        }
        current = current->next;
        position++;
    }
    printf("\n");
}
int main() {
    Node* head = NULL;
    int n, data;
    printf("Enter the no.of elements in the Doubly linked list: ");
    scanf("%d", &n);
    printf("Enter the elements into Doubly linked list: ");
    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", &data);
        head = insertNode(head, data);
    }
    printf("Elements of Doubly linked list at even position are ");
    printEvenPositionElements(head);

}
```

8.2.3

```c
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head=NULL;

void create(int n)
{
    struct node *newnode,*ptr;
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=n;
    newnode->next=NULL;
    newnode->prev=NULL;
    if(head==NULL)
    {
        head=ptr=newnode;
    }
    else
    {
        ptr=head;
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
        }
```

```
        ptr->next=newnode;
        newnode->prev=ptr;
    }

    }

    void deleten()
{
    struct node *ptr,*temp;
    ptr=head;
    temp=NULL;
    int c=1;

    while(ptr!=NULL)
    {
        if(c%2==0)
        {
        temp=ptr;
        ptr=ptr->next;
        if(temp->prev!=NULL)
        {
            temp->prev->next=ptr;
        }
        else
        {
            head=ptr;
        }
        if(ptr!=NULL)
        {
            ptr->prev=temp->prev;
```

```
        }
        free(temp);
        }
        else
    {
        ptr=ptr->next;
        }
        c++;
        }
    }
    void print()
{
    struct node *ptr;
    ptr=head;
    while(ptr!=NULL)
{
    printf("%d ",ptr->data);
    ptr=ptr->next;
    }
    }
    int main()
{
    int l,i,n;
    printf("Enter the number of elements: ");
    scanf("%d",&l);
    printf("Enter the elements into the Doubly linked list: ");
    for(i=0;i<l;i++)
{
    scanf("%d",&n);
```

```
    create(n);
    }
    printf("Original Doubly linked list: ");
    print();
    printf("\n");
    printf("Elements after deletion are ");
    deleten();
    print();


    }
```