Name: Deshmukh Mehmood Rehan

MIS No. : 612303050

SY Computer Science – Div 1

Question: Write following functions with suitable prototypes for ADT Circular Linked List :

init_CLL() // initiliazes doubly linked list

insert_beg( ) // to add an element in the end of the CLL.

insert_end( ) // to add an element in the beginning of the CLL.

insert_pos( ) // to add an element at the position specified by user of the CLL.

remove_beg() // deletes the first node of the CLL

remove_end() // deletes the last node of the CLL

remove_pos( ) // to delete an element at the position specified by user of the CLL.

sort() // sort the CLL

display() // to display all the elements of the list

You are free to include more functions.

Skeleton of function main() is given below, use same by replacing commented statements by actual function calls:

```
int main() {
  CLL L1;
  //call init() // call init for  list
  // call insert_beg( ) // call multiple times
  // insert_end( ) //   // call multiple times
  // call display()

 // call insert_pos( )

  // call remove_beg()

// call remove_end()


  // call remove_pos()


return 0;

}
```

header.h: This File includes the declarations of structures and function prototypes

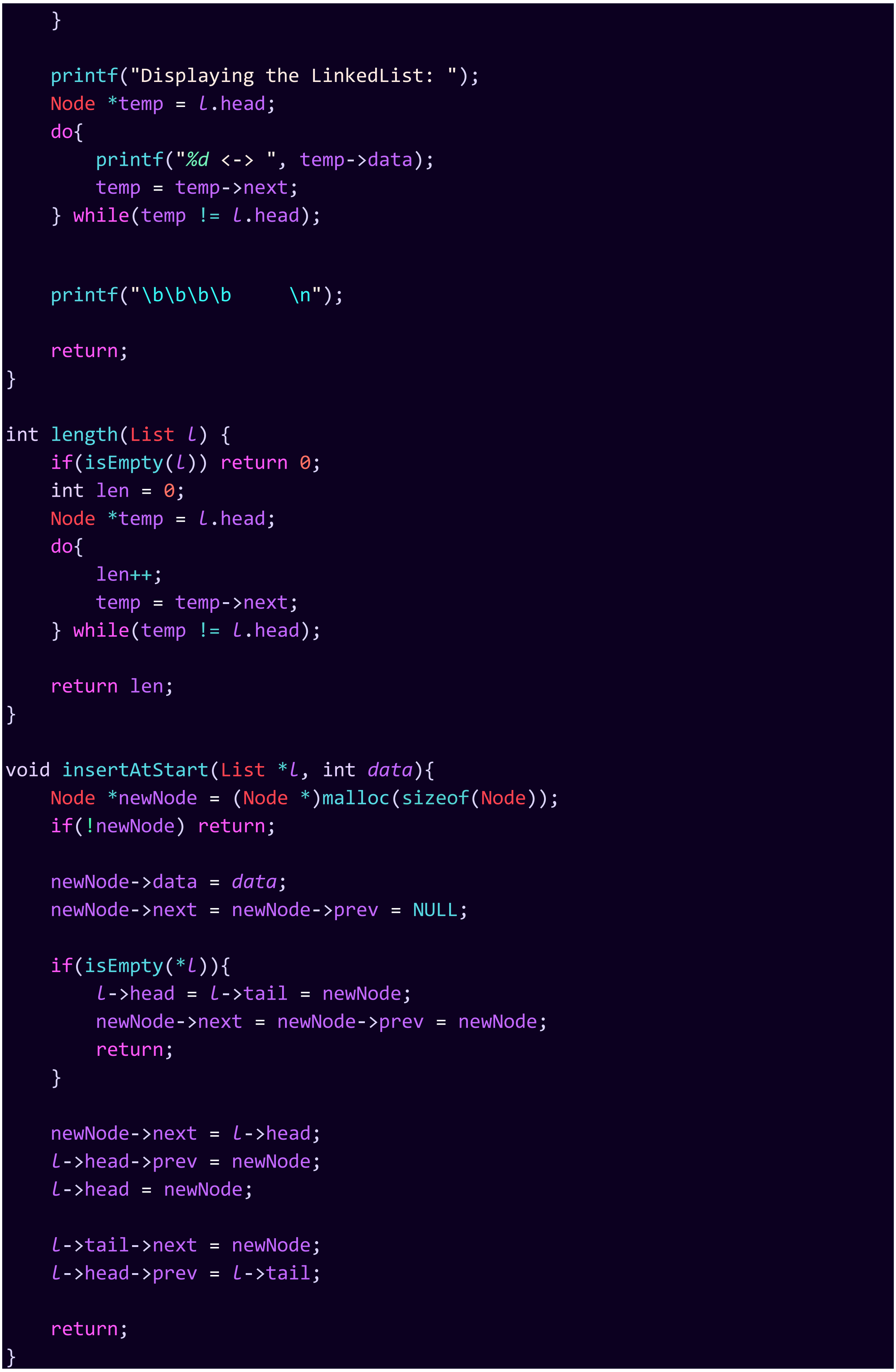
Note: The following is the implementation of Doubly Circular Linked List

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

typedef struct List{
    Node *head;
    Node *tail;
} List;

void init(List *l);
void append(List *l, int data);
void display(List l);
int length(List l);
void insertAtStart(List *l, int data);
void insertAtIndex(List *l, int data, int index);
int removeStart(List *l);
int removeAtIndex(List *l, int index);
int removeAtEnd(List *l);
void destroy(List *l);
void reverseList(List *l);
void fill(List *l, int number);
void sortList(List *l);
```

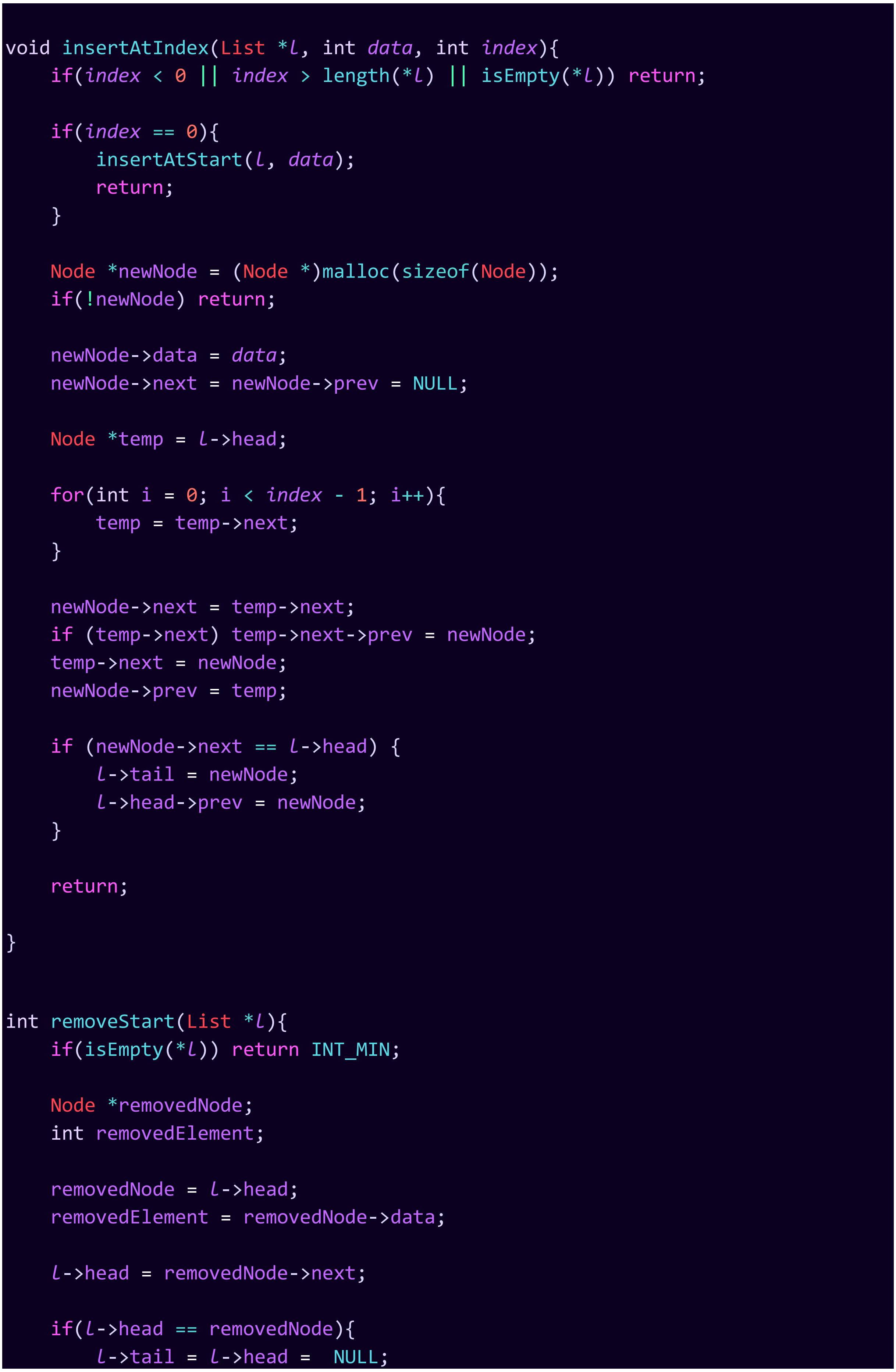
```
Node *getMid(List *l);
```

logic.c : contains the definition of all the functions declared in the header file along with some helper functions

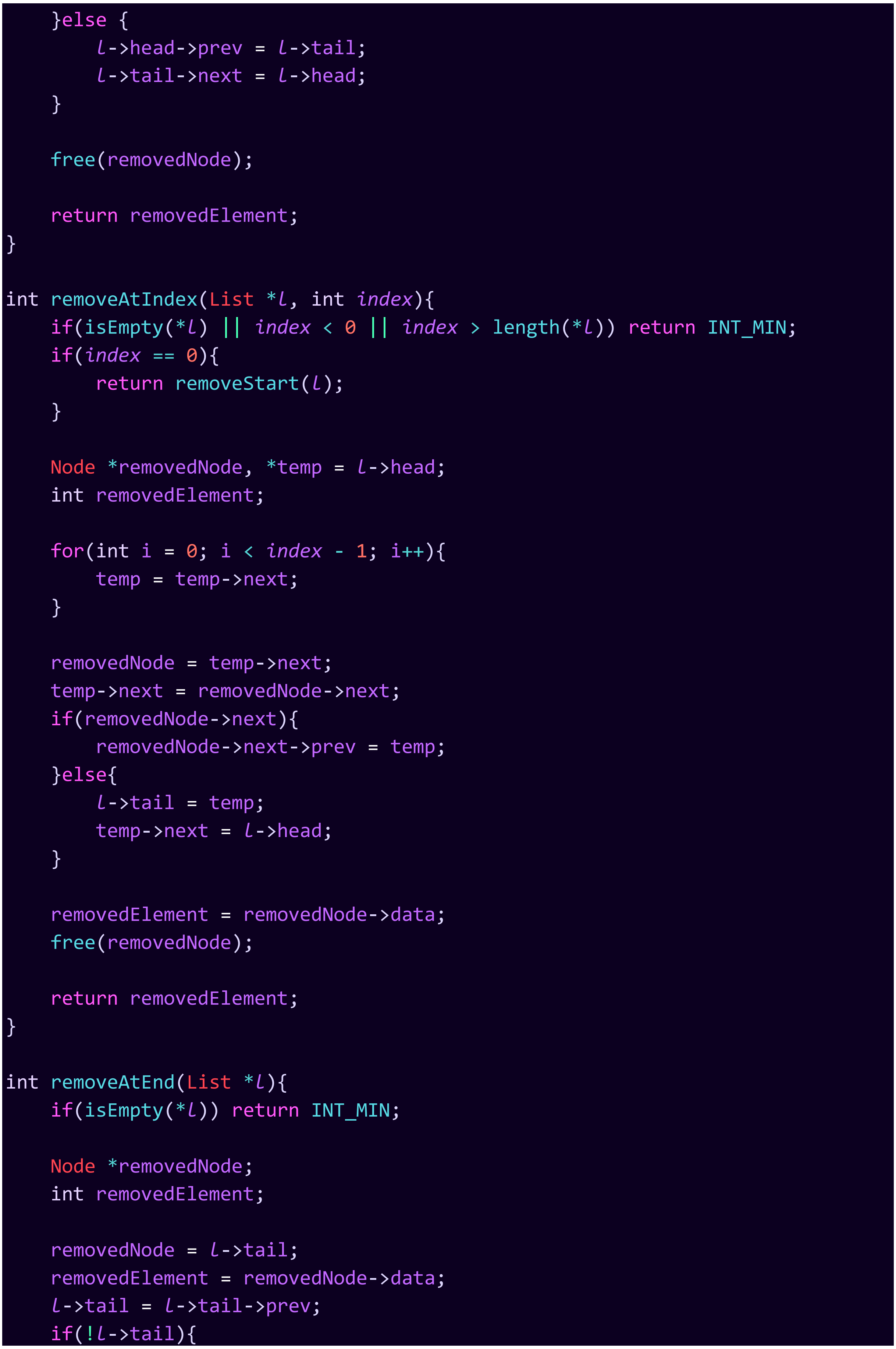
```
#include "header.h"
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

void init(List *l) {
    l->head = l->tail = NULL;
    return;
}

int isEmpty(List l) {
    return (!l.head);
}

void append(List *l, int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode)
        return;

    newNode->data = data;
    newNode->next = newNode->prev = NULL;

    if (isEmpty(*l)){
        l->head = l->tail = newNode;
        newNode->next = newNode->prev = newNode;
        return;
    }

    l->tail->next = newNode;
    newNode->prev = l->tail;
    l->tail = newNode;

    newNode->next = l->head;
    l->head->prev = newNode;

    return;
}

void display(List l) {
    if (isEmpty(l)){
        return;
```



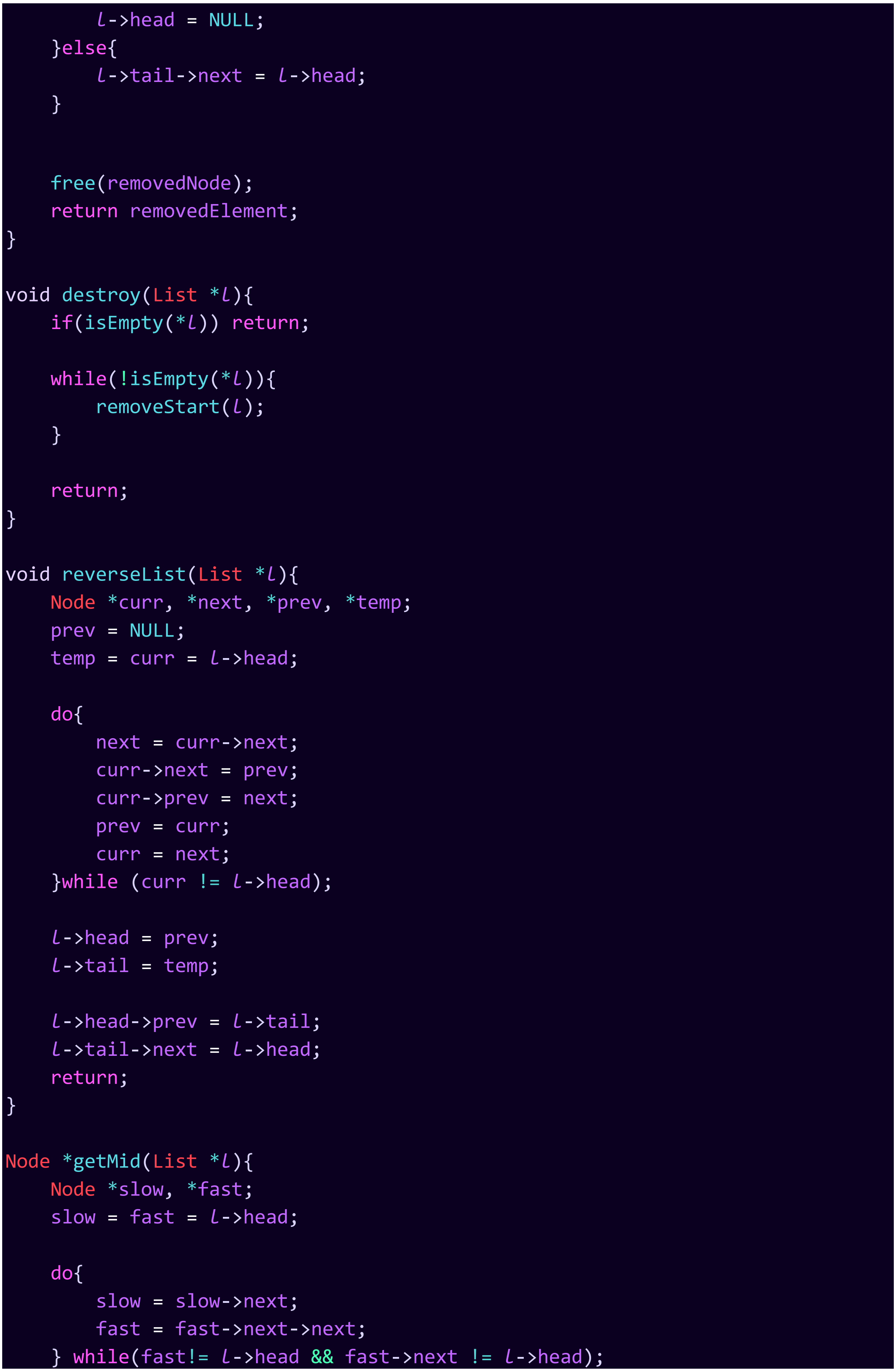
```
    }

    printf("Displaying the LinkedList: ");
    Node *temp = l.head;
    do{
        printf("%d <-> ", temp->data);
        temp = temp->next;
    } while(temp != l.head);


    printf("\b\b\b\b     \n");

    return;
}

int length(List l) {
    if(isEmpty(l)) return 0;
    int len = 0;
    Node *temp = l.head;
    do{
        len++;
        temp = temp->next;
    } while(temp != l.head);

    return len;
}

void insertAtStart(List *l, int data){
    Node *newNode = (Node *)malloc(sizeof(Node));
    if(!newNode) return;

    newNode->data = data;
    newNode->next = newNode->prev = NULL;

    if(isEmpty(*l)){
        l->head = l->tail = newNode;
        newNode->next = newNode->prev = newNode;
        return;
    }

    newNode->next = l->head;
    l->head->prev = newNode;
    l->head = newNode;

    l->tail->next = newNode;
    l->head->prev = l->tail;

    return;
}
```



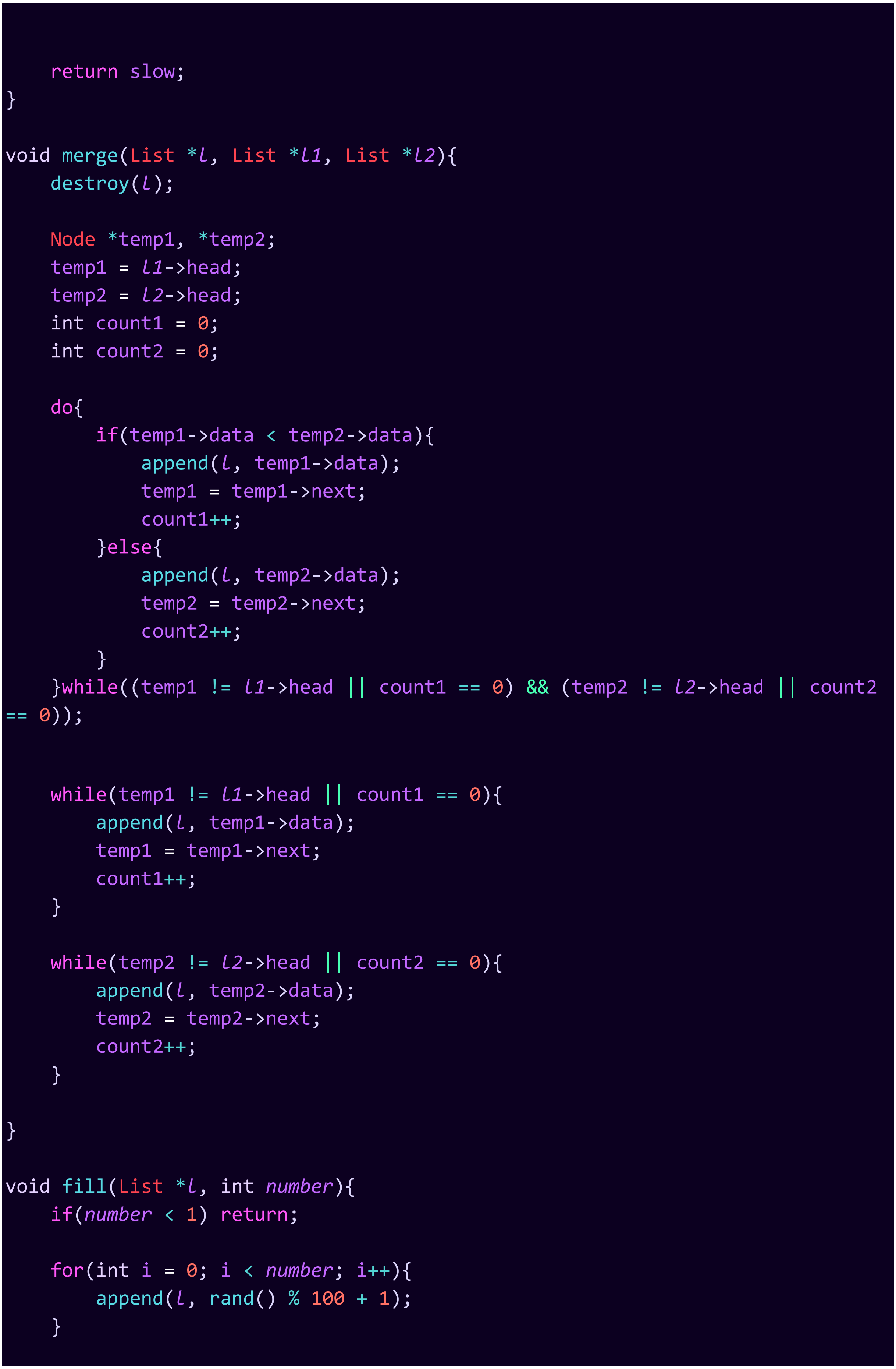
```
void insertAtIndex(List *l, int data, int index){
    if(index < 0 || index > length(*l) || isEmpty(*l)) return;

    if(index == 0){
        insertAtStart(l, data);
        return;
    }

    Node *newNode = (Node *)malloc(sizeof(Node));
    if(!newNode) return;

    newNode->data = data;
    newNode->next = newNode->prev = NULL;

    Node *temp = l->head;

    for(int i = 0; i < index - 1; i++){
        temp = temp->next;
    }

    newNode->next = temp->next;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
    newNode->prev = temp;

    if (newNode->next == l->head) {
        l->tail = newNode;
        l->head->prev = newNode;
    }

    return;

}


int removeStart(List *l){
    if(isEmpty(*l)) return INT_MIN;

    Node *removedNode;
    int removedElement;

    removedNode = l->head;
    removedElement = removedNode->data;

    l->head = removedNode->next;

    if(l->head == removedNode){
        l->tail = l->head =  NULL;
```



```
    }else {
        l->head->prev = l->tail;
        l->tail->next = l->head;
    }

    free(removedNode);

    return removedElement;
}

int removeAtIndex(List *l, int index){
    if(isEmpty(*l) || index < 0 || index > length(*l)) return INT_MIN;
    if(index == 0){
        return removeStart(l);
    }

    Node *removedNode, *temp = l->head;
    int removedElement;

    for(int i = 0; i < index - 1; i++){
        temp = temp->next;
    }

    removedNode = temp->next;
    temp->next = removedNode->next;
    if(removedNode->next){
        removedNode->next->prev = temp;
    }else{
        l->tail = temp;
        temp->next = l->head;
    }

    removedElement = removedNode->data;
    free(removedNode);

    return removedElement;
}

int removeAtEnd(List *l){
    if(isEmpty(*l)) return INT_MIN;

    Node *removedNode;
    int removedElement;

    removedNode = l->tail;
    removedElement = removedNode->data;
    l->tail = l->tail->prev;
    if(!l->tail){
```



```
        l->head = NULL;
    }else{
        l->tail->next = l->head;
    }


    free(removedNode);
    return removedElement;
}

void destroy(List *l){
    if(isEmpty(*l)) return;

    while(!isEmpty(*l)){
        removeStart(l);
    }

    return;
}

void reverseList(List *l){
    Node *curr, *next, *prev, *temp;
    prev = NULL;
    temp = curr = l->head;

    do{
        next = curr->next;
        curr->next = prev;
        curr->prev = next;
        prev = curr;
        curr = next;
    }while (curr != l->head);

    l->head = prev;
    l->tail = temp;

    l->head->prev = l->tail;
    l->tail->next = l->head;
    return;
}

Node *getMid(List *l){
    Node *slow, *fast;
    slow = fast = l->head;

    do{
        slow = slow->next;
        fast = fast->next->next;
    } while(fast!= l->head && fast->next != l->head);
```



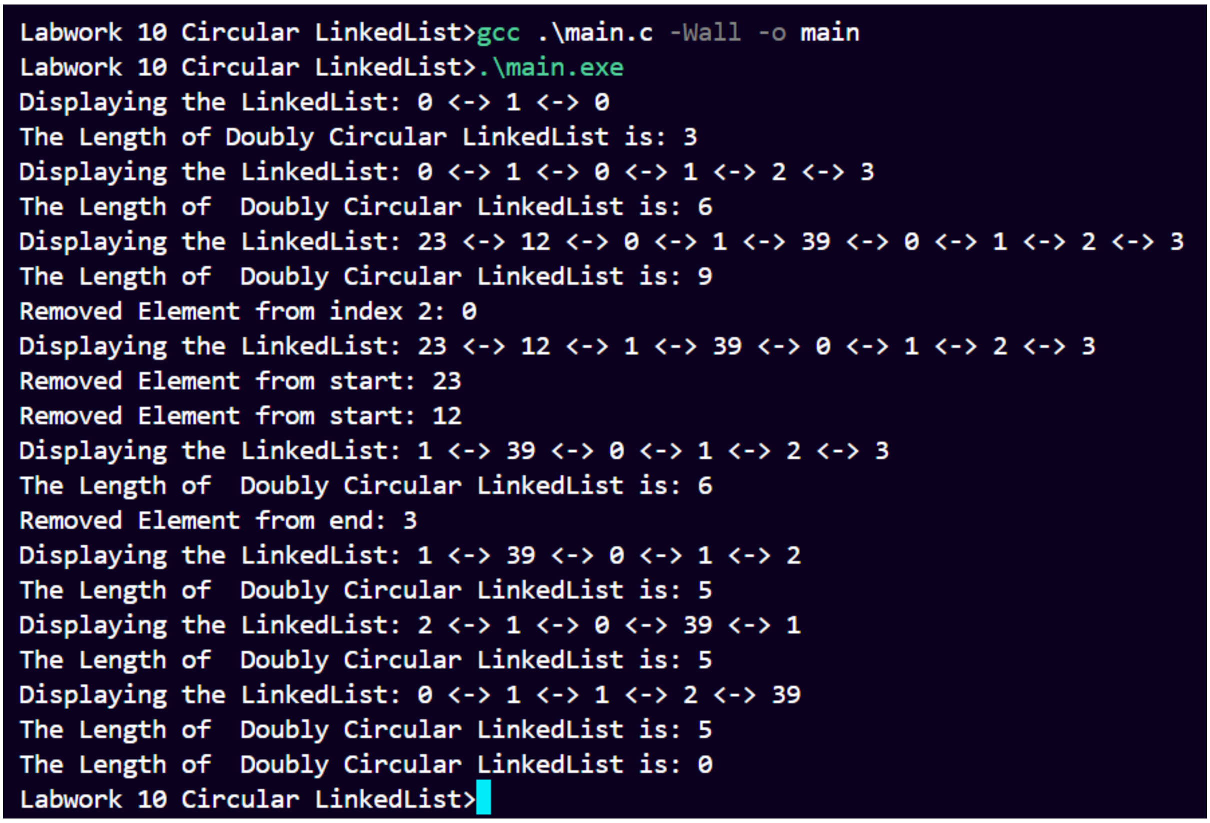
```
    return slow;
}

void merge(List *l, List *l1, List *l2){
    destroy(l);

    Node *temp1, *temp2;
    temp1 = l1->head;
    temp2 = l2->head;
    int count1 = 0;
    int count2 = 0;

    do{
        if(temp1->data < temp2->data){
            append(l, temp1->data);
            temp1 = temp1->next;
            count1++;
        }else{
            append(l, temp2->data);
            temp2 = temp2->next;
            count2++;
        }
    }while((temp1 != l1->head || count1 == 0) && (temp2 != l2->head || count2
== 0));


    while(temp1 != l1->head || count1 == 0){
        append(l, temp1->data);
        temp1 = temp1->next;
        count1++;
    }

    while(temp2 != l2->head || count2 == 0){
        append(l, temp2->data);
        temp2 = temp2->next;
        count2++;
    }

}

void fill(List *l, int number){
    if(number < 1) return;

    for(int i = 0; i < number; i++){
        append(l, rand() % 100 + 1);
    }
    return;
}

void sortList(List *l){
    if(l->head->next == l->head) return;

    Node *mid = getMid(l);

    List l1, l2;
    init(&l1);
    init(&l2);

    Node *temp = l->head;
    while(temp != mid){
        append(&l1, temp->data);
        temp = temp->next;
    }

    temp = mid;

    while(temp != l->head){
        append(&l2, temp->data);
        temp = temp->next;
    }

    sortList(&l1);
    sortList(&l2);


    merge(l, &l1, &l2);
    return;
}
```

main.c : This contains the code to test the implementation

```
#include "logic.c"

int main(){
    List l;
    init(&l);
    append(&l, 0);
    append(&l, 1);
    append(&l, 0);
    display(l);
    printf("The Length of Doubly Circular LinkedList is: %d\n", length(l));
    append(&l, 1);
    append(&l, 2);
    append(&l, 3);
```

```c
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
    insertAtStart(&l, 12);
    insertAtStart(&l, 23);
    insertAtIndex(&l, 39, 4);
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
    printf("Removed Element from index 2: %d\n", removeAtIndex(&l, 2));
    display(l);
    printf("Removed Element from start: %d\n", removeStart(&l));
    printf("Removed Element from start: %d\n", removeStart(&l));
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
    printf("Removed Element from end: %d\n", removeAtEnd(&l));
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
    reverseList(&l);
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
    sortList(&l);
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
    destroy(&l);
    display(l);
    printf("The Length of  Doubly Circular LinkedList is: %d\n", length(l));
}
```

Output:



```
Labwork 10 Circular LinkedList>gcc .\main.c -Wall -o main
Labwork 10 Circular LinkedList>.\main.exe
Displaying the LinkedList: 0 <-> 1 <-> 0
The Length of Doubly Circular LinkedList is: 3
Displaying the LinkedList: 0 <-> 1 <-> 0 <-> 1 <-> 2 <-> 3
The Length of  Doubly Circular LinkedList is: 6
Displaying the LinkedList: 23 <-> 12 <-> 0 <-> 1 <-> 39 <-> 0 <-> 1 <-> 2 <-> 3
The Length of  Doubly Circular LinkedList is: 9
Removed Element from index 2: 0
Displaying the LinkedList: 23 <-> 12 <-> 1 <-> 39 <-> 0 <-> 1 <-> 2 <-> 3
Removed Element from start: 23
Removed Element from start: 12
Displaying the LinkedList: 1 <-> 39 <-> 0 <-> 1 <-> 2 <-> 3
The Length of  Doubly Circular LinkedList is: 6
Removed Element from end: 3
Displaying the LinkedList: 1 <-> 39 <-> 0 <-> 1 <-> 2
The Length of  Doubly Circular LinkedList is: 5
Displaying the LinkedList: 2 <-> 1 <-> 0 <-> 39 <-> 1
The Length of  Doubly Circular LinkedList is: 5
Displaying the LinkedList: 0 <-> 1 <-> 1 <-> 2 <-> 39
The Length of  Doubly Circular LinkedList is: 5
The Length of  Doubly Circular LinkedList is: 0
Labwork 10 Circular LinkedList>
```