Name: Deshmukh Mehmood Rehan

MIS No. : 612303050

SY Computer Science – Div 1

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

init_DLL() // initiliazes doubly linked list

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

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

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

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

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

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

sort() // sort the DLL

displayRL() // to display all the elements of the list starting from right element to left.

displayLR() // to display all the elements of the list starting from left element to right.

is_palindrome() /*The functions determines the given DLL is a palindrome or not. For example, if the list is: {1, 2, 3, 2, 1} is a palindrome. For example, if the DLL is: {1, 2, 3, 1, 2, 1} is a not a palindrome.*/

remove_duplicates() /* The functions removes a duplicate node keeping only one node so that elements of node are unique.

For example, if the DLL is: {1, 2, 3, 2, 1} after the function is invoked the DLL changes to: {1, 2, 3 } */

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() {
  DLL L1;
  //call init() // call init for  list
  // call insert_beg( )  // call multiple times
  // insert_end( )  //   // call multiple times
  // call displayLR()
 // call insert_pos( )
  // call displayRL()
 // call is_palindrome()
 // call remove_beg()
 // call displayLR()
// call remove_end()
 // call displayLR()
  // call remove_pos()
 // call displayLR()
return 0;
}
/* Call all the functions in main */
```

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

```
/*
this structure will be the node of our Doubly Linked  List;
For it to be a "Doubly" LinkedList we need the node to have two pointers
A previous pointer "prev" which will point to the previous element
and a Next pointer "next" which will point to the next element

in case of first Node the prev will point to NULL and
in case of last element the next will point to NULL
```

```
*/


typedef struct Node{
    int data;  // the data (in our case it is an Integer)
    struct Node *next; // the next pointer
    struct Node *prev; // the prev pointer
} Node;

/*
This will be our Data Structure or more specifically the ADT
for that we will need two pointers, first one to the beginning of the list
and other at the end of the list
we will typedef both of them into ADT list
*/
typedef struct List{
    Node *head; //pointer to the first Node
    Node *tail; //pointer to the last Node
} List;

void init(List *l); // this function initializes the list
void insertAtStart(List *l, int data); // this function inserts an element at
the start of the list
void append(List *l, int data); // this function inserts an element at the end
of the list
void insertAtIndex(List *l, int data, int index); // this function inserts an
element at a given index in the list
int removeStart(List *l); // this function removes element from the start of
the list
int removeAtEnd(List *l); // this function removes element from the end of the
list
int removeAtIndex(List *l, int index); // this function removes element from a
given index of the list
void sortList(List *l); // this function sorts the list
void displayLR(List l); // this function displays the list from left to right
void displayRL(List l); // this function displays the list from right to left
void removeDuplicates(List *l); // this function removes duplicates from the
list
int isPalindrome(List l); // this function checks wether the list is
palindrome


// some extra functions
int length(List l);
void destroy(List *l);
void reverseList(List *l);
void fill(List *l, int number);
```

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

```c
#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;
        return;
    }

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

    return;
}

void displayLR(List l) {
    if (isEmpty(l))
        return;

    printf("Displaying the LinkedList: ");

    for (Node *temp = l.head; temp; temp = temp->next)
    {
        printf("%d <-> ", temp->data);
    }
```

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

    return;
}

void displayRL(List l) {
    if (isEmpty(l))
        return;

    printf("Displaying the LinkedList: ");

    for (Node *temp = l.tail; temp; temp = temp->prev)
    {
        printf("%d <-> ", temp->data);
    }
    printf("\b\b\b\b     \n");

    return;
}

int length(List l) {
    int len = 0;
    for (Node *temp = l.head; temp; temp = temp->next)
    {
        len++;
    }

    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;
        return;
    }

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

    return;
}
```

```c
void insertAtIndex(List *l, int data, int index){
    if(index < 0 || index > length(*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 && temp; i++){
        temp = temp->next;
    }

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

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

```
    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;
    }

    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 = NULL;
    }
```

```
    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;

    while (curr){
        next = curr->next;
        curr->next = prev;
        curr->prev = next;
        prev = curr;
        curr = next;
    }

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

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

    while(fast && fast->next){
        slow = slow->next;
        fast = fast->next->next;
    }

    return slow;
}

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

    if (l1->head == NULL) {
```

```
        *l = *l2;
        return;
    }
    if (l2->head == NULL) {
        *l = *l1;
        return;
    }


    Node *temp1, *temp2;
    temp1 = l1->head;
    temp2 = l2->head;

    while(temp1 && temp2){
        if(temp1->data < temp2->data){
            append(l, temp1->data);
            temp1 = temp1->next;
        }else{
            append(l, temp2->data);
            temp2 = temp2->next;
        }
    }

    while(temp1){
        append(l, temp1->data);
        temp1 = temp1->next;
    }

    while(temp2){
        append(l, temp2->data);
        temp2 = temp2->next;
    }


}

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


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


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



void removeDuplicates(List *l){
    int len = length(*l);

    int * arr = (int *) calloc(len, sizeof(int));
    if (!arr) return;

    int newlen = 0;
    Node * p = l->head;

    arr[newlen++] = p->data;

    while (p->next)
    {
        int data = p->next->data;
        int duplicate = 0;

        for (int i = 0; i < newlen; i++)
        {
            if (arr[i] == data){
```

```
                duplicate = 1;
                break;
            }
        }

        if (duplicate)
        {
            Node * q = p->next;
            p->next = p->next->next;
            p->next->prev = p;
            free(q);
        }
        else{
            arr[newlen++] = data;
            p = p->next;
        }
    }

}


int isPalindrome(List l){
    Node *temp1 = l.head;
    Node *temp2 = l.tail;

    while(temp1 && temp2){
        if(temp1->data != temp2->data) return 0;
        temp1 = temp1->next;
        temp2 = temp2->prev;
    }

    return 1;

}
```

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);
    displayLR(l);
    displayRL(l);
```

```
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    append(&l, 1);
    append(&l, 2);
    append(&l, 3);
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    insertAtStart(&l, 12);
    insertAtStart(&l, 23);
    insertAtIndex(&l, 39, 4);
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    printf("Removed Element from index 2: %d\n", removeAtIndex(&l, 2));
    printf("Removed Element from start: %d\n", removeStart(&l));
    printf("Removed Element from start: %d\n", removeStart(&l));
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    printf("Removed Element from end: %d\n", removeAtEnd(&l));
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    reverseList(&l);
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    sortList(&l);
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    removeDuplicates(&l);
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));
    destroy(&l);
    displayLR(l);
    printf("The Length of Doubly LinkedList is: %d\n", length(l));

    List palindrome1, palindrome2;
    init(&palindrome1);
    init(&palindrome2);

    append(&palindrome1, 1);
    append(&palindrome1, 2);
    append(&palindrome1, 3);
    append(&palindrome1, 2);
    append(&palindrome1, 1);

    displayLR(palindrome1);
    printf("The List is Palindrome : %s\n", isPalindrome(palindrome1) ? "True"
: "False");

    append(&palindrome2, 1);
    append(&palindrome2, 2);
```

```
    append(&palindrome2, 3);
    append(&palindrome2, 1);
    append(&palindrome2, 2);
    append(&palindrome2, 1);

    displayLR(palindrome2);
    printf("The List is Palindrome : %s\n", isPalindrome(palindrome2) ? "True"
: "False");
}
```

Output:

```
Labwork 9 Doubly LinkedList>gcc .\main.c -Wall -o main
Labwork 9 Doubly LinkedList>.\main.exe
Displaying the LinkedList: 0 <-> 1 <-> 0
Displaying the LinkedList: 0 <-> 1 <-> 0
The Length of Doubly LinkedList is: 3
Displaying the LinkedList: 0 <-> 1 <-> 0 <-> 1 <-> 2 <-> 3
The Length of Doubly LinkedList is: 6
Displaying the LinkedList: 23 <-> 12 <-> 0 <-> 1 <-> 39 <-> 0 <-> 1 <-> 2 <-> 3
The Length of Doubly LinkedList is: 9
Removed Element from index 2: 0
Removed Element from start: 23
Removed Element from start: 12
Displaying the LinkedList: 1 <-> 39 <-> 0 <-> 1 <-> 2 <-> 3
The Length of Doubly LinkedList is: 6
Removed Element from end: 3
Displaying the LinkedList: 1 <-> 39 <-> 0 <-> 1 <-> 2
The Length of Doubly LinkedList is: 5
Displaying the LinkedList: 2 <-> 1 <-> 0 <-> 39 <-> 1
The Length of Doubly LinkedList is: 5
Displaying the LinkedList: 0 <-> 1 <-> 1 <-> 2 <-> 39
The Length of Doubly LinkedList is: 5
Displaying the LinkedList: 0 <-> 1 <-> 2 <-> 39
The Length of Doubly LinkedList is: 4
The Length of Doubly LinkedList is: 0
Displaying the LinkedList: 1 <-> 2 <-> 3 <-> 2 <-> 1
The List is Palindrome : True
Displaying the LinkedList: 1 <-> 2 <-> 3 <-> 1 <-> 2 <-> 1
The List is Palindrome : False
Labwork 9 Doubly LinkedList>
```