LAB SUBMISSION – 5 <-link

Meher Shrishti Nigam – 20BRS1193

CODE FOR DOUBLY LINKED LIST – FOR QUESTIONS 1, 2

```
// DoublyLinkedList.c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

/*
    A doubly linked list here is defined by its "start".
    "start" is a node pointer that points to the first element.

    Here, start is a proper node with memory allocated to it.
    Note that start doesn't store any actual data.
    The next of the start node stores the location of the first element in
the doubly linked list.
    The prev of the start node is NULL and never accessed.
    The data of the start node is INT_MIN and never accessed.

    Thus, if we create a doubly linked list "dll", dll->start->next gives
us the first element.

    The first element's prev node is NULL. (Thus, it is not connected to
the start node.)

    "len" stores the number of elements in the linked list. This can be
easily calculated,
    however it is added to make things easier and for illustration. It can
be removed.
    Functions are written independent of len.

    Functions provided:

    insertAtStart
    insertAtEnd
    insertAtPosition

    display
    search

    deleteAtStart
    deleteAtEnd
```

```
    deleteAtPosition

*/

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

typedef struct DoublyLinkedList
{
    Node * start;
    int len;
}DoublyLinkedList;

Node * createNode(int item)
{
    Node * temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}

DoublyLinkedList * createDoublyLinkedList()
{
    DoublyLinkedList * dll = (DoublyLinkedList
*)malloc(sizeof(DoublyLinkedList));
    dll->len = 0;
    dll->start = (Node *)malloc(sizeof(Node));
    dll->start->data = INT_MIN; // Data in the start node is never to be
accessed. If INT_MIN is the data displayed, an error has possibly occured
    dll->start->next = NULL;
    dll->start->prev = NULL;
    printf("A new doubly linked list was created!\n");
    return dll;
}

/* Insertion */

// Inserts an element at the start of a linked list
void insertAtStart(DoublyLinkedList * dll, int item)
{
```

```
    Node * newnode = createNode(item);
    if(dll->start->next != NULL)
    {
        newnode->next = dll->start->next;
        dll->start->next->prev = newnode;
    }
    dll->start->next = newnode;
    printf("%d was inserted at the start of the doubly linked list!\n",
item);
    dll->len++;
}
// Inserts an element at the end of a doubly linked list
void insertAtEnd(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = newnode;
    if(ptr != dll->start)
        newnode->prev = ptr;
    printf("%d was inserted at the end of the doubly linked list!\n",
item);
    dll->len++;
}

// Inserts an element at a specified position.
// Here, position is determined by usual 1-base counting. If position = 5,
item will be the fifth element in the linked list
void insertAtPosition(DoublyLinkedList * dll, int item, int position)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    for(int i = 0; i < (position - 1); i++)
    {
        if(ptr->next == NULL && i + 1 < position)
        {
            printf("Invalid Location\n");
            return;
        }
        ptr = ptr->next;
    }
    if(position != 1) // If adding at beginning then newnode's prev will
remain NULL
```

```
        newnode->prev = ptr;
    if(dll->start->next != NULL) // When list is not empty
    {
        newnode->next = ptr->next;
        if(ptr->next != NULL)
            ptr->next->prev = newnode; // Skip this if adding at the end
    }
    ptr->next = newnode;
    printf("%d was inserted at position %d of the doubly linked list!\n",
item, position);
    dll->len++;
}

/* Traversal */
/* Display function provided for illustrative purposes.*/
void display(DoublyLinkedList * dll)
{
     if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to display.\n");
        return;
    }

    // Printing data
    Node * ptr = dll->start->next;
    printf("Start: %d \n", dll->start);
    while(ptr != NULL)
    {
        printf("%d       ", ptr->data);
        ptr = ptr->next;
    }
    // Printing the location
    Node * ptr2 = dll->start->next;
    printf("\n%d ", dll->start->next);
    while(ptr2 != NULL)
    {
        printf("%d ", ptr2->next);
        ptr2 = ptr2->next;
    }
    printf("\n");
}

// Returns the 1-base position of the first occurance of data.
int search(DoublyLinkedList * dll, int item)
```

```
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to search for.\n");
        return INT_MIN;
    }
    int count = 1;
    Node * ptr = dll->start->next;
    while(ptr->data != item)
    {
        if(ptr->data != item && ptr->next == NULL)
        {
            printf("%d not found in the doubly linked list.\n", item);
            return INT_MIN;
        }
        ptr = ptr->next;
        count++;
    }
    printf("%d found at %d position in the doubly linked list.\n", item,
count);
    return count;
}

/* Deletion */

// Deletion at start
int deleteAtStart(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete.\n");
        return INT_MIN;
    }
    Node * ptr = dll->start->next;
    dll->start->next = ptr->next;
    if(ptr->next != NULL)
        ptr->next->prev = NULL;
    int data = ptr->data;
    free(ptr);
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    dll->len--;
    return data;
}
```

```c
// Deletion at end
int deleteAtEnd(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
    }
    int data;
    Node * ptr = dll->start->next;
    while(ptr->next != NULL)
        ptr = ptr->next;
    if(ptr->prev == NULL) // Only one element is present
        dll->start->next = NULL;
    else // Other cases
        ptr->prev->next = NULL;
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    return data;
}
// Deletion of an element at a particular position
// Here, position is determined by usual 1-base counting.
int deleteAtPosition(DoublyLinkedList * dll, int position)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
    }
    Node * ptr = dll->start;
    int data;
    for(int i = 0; i < position; i++)
    {
        if(ptr->next == NULL && i < position)
        {
            printf("Invalid Location\n");
            return INT_MIN;
        }
        ptr = ptr->next;
    }
```

```
    if(ptr->prev == NULL) // If first element is to be deleted
        dll->start->next = ptr->next;
    else
        ptr->prev->next = ptr->next;

    if(ptr->next != NULL)
        ptr->next->prev = ptr->prev;
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from %d position of the doubly linked list.\n",
data, position);
    return data;
}

void introduction()
{
    printf("~ To display the DLL, enter 1.\n\n");
    printf("INSERTION\n~ To insert an element at the start DLL, enter 2
and the element.\n");
    printf("~ To insert an element at the end of the DLL, enter 3 and the
element.\n");
    printf("~ To insert an element at a specific position of the DLL,
enter 4, the element and the position.\n\n");
    printf("DELETION\n~ To delete an element from the start DLL, enter
5.\n");
    printf("~ To delete an element from the end of the DLL, enter 6.\n");
    printf("~ To delete an element from a specific position of the DLL,
enter 7 and the position.\n");
    printf("\n~ To search for an element, enter 8 and the element.\n\n");
    printf("~ To print this message again, enter 9.\n");
    printf("~ To exit, enter 0.\n\n");
}

int main()
{
    printf("\nWelcome to Doubly Linked List (DLL) Generator!\n\n");
    introduction();
    int x = -1;
    int item, position;
    DoublyLinkedList * dll = createDoublyLinkedList();
    printf("\n");
    while(x != 0)
```

```
    {
        scanf("%d", &x);
        switch (x)
        {
            case 0: printf("Exiting... Thank you!\n"); break;
            case 1: display(dll); break;

            case 2: printf("data: "); scanf("%d", &item);
insertAtStart(dll, item); break;
            case 3: printf("data: "); scanf("%d", &item); insertAtEnd(dll,
item); break;
            case 4: printf("data: "); scanf("%d", &item);
printf("position: "); scanf("%d", &position); insertAtPosition(dll, item,
position); break;

            case 5: deleteAtStart(dll); break;
            case 6: deleteAtEnd(dll); break;
            case 7: printf("position: "); scanf("%d", &position);
deleteAtPosition(dll, position); break;

            case 8: printf("data to be searched: "); scanf("%d", &item);
search(dll, item); break;

            case 9: introduction(); break;
            default: printf("Invalid value, try again.\n"); break;
        }
    }
}
```

Next Page..

The following cases were tested while writing the code for insertAtStart

1. Insert at start in an empty list
2. Insert at start in a non-empty list

The following cases were tested while writing the code for insertAtEnd

1. Insert at end in an empty list
2. Insert at end in a non-empty list

The following cases were tested while writing the code for insertAtPosition

1. Insertion in between two elements
2. Insertion at the end in a non-empty list
3. Insert at beginning in a non-empty list
4. Insertion when list is empty and position = 1
5. Insertion when list is empty but position > 1 (invalid case)
6. Insertion when position is too large (invalid)

The following cases were tested while writing the code for deleteAtStart

1. Delete at start when list is empty (invalid)
2. Delete at start when list has 1 element
3. Delete at start when list has more than 1 element

The following cases were tested while writing the code for deleteAtEnd

1. Delete at end when list is empty (invalid)
2. Delete at end when list has 1 element
3. Delete at end when list has more than 1 element

The following cases were tested while writing the code for deleteAtPosition

1. Deletion from empty list (invalid)
2. Deletion in between two elements
3. Deletion at end when list has more than 1 element
4. Deletion at start when list has more than 1 element
5. Deletion when 1 element is present and postion = 1
6. Deletion when 1 element is present but postion > 1 (invalid case).
7. Deletion when position is too large (invalid)

RUNNING THE DOUBLY LINKED LIST CODE TO SOLVE PROBLEM 1

1- Write a program (preferably in C) to search a given element in doubly linked list at the given location

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> gcc DoublyLinkedList.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> ./a.exe

Welcome to Doubly Linked List (DLL) Generator!

~ To display the DLL, enter 1.

INSERTION
~ To insert an element at the start DLL, enter 2 and the element.
~ To insert an element at the end of the DLL, enter 3 and the element.
~ To insert an element at a specific position of the DLL, enter 4, the element and the position.

DELETION
~ To delete an element from the start DLL, enter 5.
~ To delete an element from the end of the DLL, enter 6.
~ To delete an element from a specific position of the DLL, enter 7 and the position.

~ To search for an element, enter 8 and the element.

~ To print this message again, enter 9.
~ To exit, enter 0.

A new doubly linked list was created!

2
data: 45
45 was inserted at the start of the doubly linked list!
3
data: 87
87 was inserted at the end of the doubly linked list!
2
data: 57
57 was inserted at the start of the doubly linked list!
4
data: 1
position: 1
1 was inserted at position 1 of the doubly linked list!
4
data: 570
position: 4
570 was inserted at position 4 of the doubly linked list!
1
Start: 1840224
1       57      45      570     87
1840352 1840320 1840256 1840384 1840288 0
8
data to be searched: 57
57 found at 2 position in the doubly linked list.
8
data to be searched: 60
60 not found in the doubly linked list.
0
Exiting... Thank you!
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5>
```

RUNNING THE DOUBLY LINKED LIST CODE TO SOLVE PROBLEM 2(a)

2- Write a program for the various cases of insertion and deletion (you can implement all in single program itself) in doubly linked list and circular doubly linked list.

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> gcc DoublyLinkedList.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> ./a.exe

Welcome to Doubly Linked List (DLL) Generator!

~ To display the DLL, enter 1.

INSERTION
~ To insert an element at the start DLL, enter 2 and the element.
~ To insert an element at the end of the DLL, enter 3 and the element.
~ To insert an element at a specific position of the DLL, enter 4, the element and the position.

DELETION
~ To delete an element from the start DLL, enter 5.
~ To delete an element from the end of the DLL, enter 6.
~ To delete an element from a specific position of the DLL, enter 7 and the position.

~ To search for an element, enter 8 and the element.

~ To print this message again, enter 9.
~ To exit, enter 0.

A new doubly linked list was created!

2
data: 45
45 was inserted at the start of the doubly linked list!
5
45 was deleted from the start of the doubly linked list.
3
data: 54
54 was inserted at the end of the doubly linked list!
6
54 was deleted from the start of the doubly linked list.
4
data: 14
position: 1
14 was inserted at position 1 of the doubly linked list!
7
position: 43
Invalid Location
7
position: 1
14 was deleted from 1 position of the doubly linked list.
2
```

```
14 was deleted from 1 position of the doubly linked list.
2
data: 3
3 was inserted at the start of the doubly linked list!
3
data: 4
4 was inserted at the end of the doubly linked list!
2
data: 7
7 was inserted at the start of the doubly linked list!
3
data: 9
9 was inserted at the end of the doubly linked list!
4
data: 67
position: 4
67 was inserted at position 4 of the doubly linked list!
1
Start: 7672928
7       3       4       67      9
7673024 7672960 7672992 7673088 7673056 0
4
data: 64
position: 6
64 was inserted at position 6 of the doubly linked list!
1
Start: 7672928
7       3       4       67      9       64
7673024 7672960 7672992 7673088 7673056 7673120 0
5
7 was deleted from the start of the doubly linked list.
6
64 was deleted from the start of the doubly linked list.
7
position: 4
9 was deleted from 4 position of the doubly linked list.
7
position: 4
Invalid Location
7
position: 1
3 was deleted from 1 position of the doubly linked list.
5
```

```
3 was deleted from 1 position of the doubly linked list.
5
4 was deleted from the start of the doubly linked list.
6
67 was deleted from the start of the doubly linked list.
5
Doubly Linked List is empty, nothing to delete.
1
Empty Doubly Linked List, nothing to display.
0
Exiting... Thank you!
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5>
```

CODE FOR CIRCULAR DOUBLY LINKED LIST – FOR QUESTION 2

```
// CircularDoublyLinkedList.c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

/*
    A circular doubly linked list here is defined by its "start".
    "start" is a node pointer that points to the first element.

    Here, start is a proper node with memory allocated to it.
    Note that start doesn't store any actual data.
    The next of the start node stores the location of the first element in
the doubly linked list.
    The prev of the start node is NULL and never accessed.
    The data of the start node is INT_MIN and never accessed.

    Thus, if we create a circular doubly linked list "dcll", dcll->start-
>next gives us the first element.

    The first element's prev node is the address of the last node. (Thus,
it is not connected to the start node.)
    The last elements's next node is the address of the first node. Thus
the linked list is circular.

    "len" stores the number of elements in the linked list. This can be
easily calculated,
    however it is added to make things easier and for illustration. It can
be removed.
    Functions are written independent of len.

    Functions provided:

    insertAtStart
    insertAtEnd
    insertAtPosition

    display
    search

    deleteAtStart
    deleteAtEnd
    deleteAtPosition -> not done
```

```
*/

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

typedef struct CircularDoublyLinkedList
{
    Node * start;
    int len;
}CircularDoublyLinkedList;

Node * createNode(int item)
{
    Node * temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}

CircularDoublyLinkedList * createCircularDoublyLinkedList()
{
    CircularDoublyLinkedList * dcll = (CircularDoublyLinkedList
*)malloc(sizeof(CircularDoublyLinkedList));
    dcll->len = 0;
    dcll->start = (Node *)malloc(sizeof(Node));
    dcll->start->data = INT_MIN; // Data in the start node is never to be
accessed. If INT_MIN is the data displayed, an error has possibly occured
    dcll->start->next = NULL;
    dcll->start->prev = NULL;
    printf("A new circular doubly linked list was created!\n");
    return dcll;
}

/* Insertion */

// Inserts an element at the start of a linked list
void insertAtStart(CircularDoublyLinkedList * dcll, int item)
{
    Node * newnode = createNode(item);
    if(dcll->start->next == NULL) // If dcll is empty
```

```
    {
        dcll->start->next = newnode;
        newnode->next = newnode;
        newnode->prev = newnode;
    }
    else
    {
        newnode->next = dcll->start->next;
        newnode->prev = dcll->start->next->prev;
        dcll->start->next->prev->next = newnode;
        dcll->start->next->prev = newnode;
        dcll->start->next = newnode;
    }
    printf("%d was inserted at the start of the circular doubly linked
list!\n", item);
    dcll->len++;
}

// Inserts an element at the end of a linked list
void insertAtEnd(CircularDoublyLinkedList * dcll, int item)
{
    Node * newnode = createNode(item);
    if(dcll->start->next == NULL) // If dcll is empty
    {
        dcll->start->next = newnode;
        newnode->next = newnode;
        newnode->prev = newnode;
    }
    else
    {
        newnode->next = dcll->start->next;
        newnode->prev = dcll->start->next->prev;
        dcll->start->next->prev->next = newnode;
        dcll->start->next->prev = newnode;
    }
    printf("%d was inserted at the end of the circular doubly linked
list!\n", item);
    dcll->len++;
}

// Inserts an element at a specified position.
// Here, position is determined by usual 1-base counting. If position = 5,
item will be the fifth element in the linked list
```

```c
void insertAtPosition(CircularDoublyLinkedList * dcll, int item, int
position)
{
    Node * newnode = createNode(item);
    Node * ptr = dcll->start;
    for(int i = 0; i < (position - 1); i++)
    {
        if(ptr->next == dcll->start->next && i + 1 < position && ptr->data
!= INT_MIN)
        {
            printf("Invalid Location\n");
            return;
        }
        ptr = ptr->next;
    }
    if(dcll->start->next == NULL) // If dcll is empty
    {
        if(position != 1)
        {
            printf("Invalid Location\n");
            return;
        }
        newnode->next = newnode;
        newnode->prev = newnode;
    }
    else
    {
        newnode->next = ptr->next;
        newnode->prev = ptr->next->prev;
        if(position == 1)
            ptr->next->prev->next = newnode; // for 1st position
        ptr->next->prev = newnode;
    }
    ptr->next = newnode;
    printf("%d was inserted at position %d of the circular doubly linked
list!\n", item, position);
    dcll->len++;
}

/* Traversal */

// Display function provided for illustrative purposes.
void display(CircularDoublyLinkedList * dcll)
{
```

```
    if(dcll->start->next == NULL)
    {
        printf("Circular doubly linked list is empty, nothing to
display.\n");
        return;
    }
    // Printing data
    Node * ptr = dcll->start->next;
    if(ptr == ptr->next)
        printf("%d      ", ptr->data);
    else
    {
        while(ptr->next != dcll->start->next)
        {
            printf("%d       ", ptr->data);
            ptr = ptr->next;
        }
        printf("%d      ", ptr->data);
    }
    // Printing the location
    Node * ptr2 = dcll->start->next;
    printf("\n%d ", ptr->next);
    while(ptr2->next != dcll->start->next)
    {
        printf("%d ", ptr2->next);
        ptr2 = ptr2->next;
    }
    printf("%d ", ptr2->next);  // -> to check circular nature
    printf("\n");
}

// Returns the 1-base position of the first occurance of data.
int search(CircularDoublyLinkedList * dcll, int item)
{
    if(dcll->start->next == NULL)
    {
        printf("Empty circular doubly linked list, nothing to search
for.\n");
        return INT_MIN;
    }
    int count = 1;
    Node * ptr = dcll->start->next;
    while(ptr->data != item)
    {
```

```
        if(ptr->data != item && ptr->next == dcll->start->next)
        {
            printf("%d not found in the circular doubly linked list.\n",
item);
            return INT_MIN;
        }
        ptr = ptr->next;
        count++;
    }
    printf("%d found at %d position in the circular doubly linked
list.\n", item, count);
    return count;
}

/* Deletion */

// Deletion at start
int deleteAtStart(CircularDoublyLinkedList * dcll)
{
    if(dcll->start->next == NULL)
    {
        printf("Circular Doubly Linked List is empty, nothing to
delete\n");
        return INT_MIN;
    }
    Node * ptr = dcll->start->next;
    if(ptr->next == ptr)
        dcll->start->next = NULL;
    else
    {
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        dcll->start->next = ptr->next;
    }
    int data = ptr->data;
    free(ptr);
    printf("%d was deleted from the start of the circular doubly linked
list.\n", data);
    dcll->len--;
    return data;
}

// Deletion at end
int deleteAtEnd(CircularDoublyLinkedList * dcll)
```

```
{
    if(dcll->start->next == NULL)
    {
        printf("Circular Doubly Linked List is empty, nothing to
delete\n");
        return INT_MIN;
    }
    Node * ptr = dcll->start->next;
    if(ptr->next == ptr)
        dcll->start->next = NULL;
    else
    {
        ptr = ptr->prev;
        dcll->start->next->prev = ptr->prev;
        ptr->prev->next = dcll->start->next;
    }
    int data = ptr->data;
    free(ptr);
    printf("%d was deleted from the end of the circular doubly linked
list.\n", data);
    dcll->len--;
    return data;
}

// Deletion of an element at a particular position
// Here, position is determined by usual 1-base counting.
int deleteAtPosition(CircularDoublyLinkedList * dcll, int position)
{
    if(dcll->start->next == NULL)
    {
        printf("Circular doubly linked list is empty, nothing to
delete\n");
        return INT_MIN;
    }
    Node * ptr = dcll->start;
    for(int i = 0; i < position; i++)
    {
        if(ptr->next == dcll->start->next && i < position && ptr->data !=
INT_MIN)
        {
            printf("Invalid Location\n");
            return INT_MIN;
        }
        ptr = ptr->next;
```

```
    }
    if(ptr->next == ptr)
        dcll->start->next = NULL;
    else
    {
        if(ptr->next == dcll->start->next) // deletion at end
        {
            dcll->start->next->prev = ptr->prev;
            ptr->prev->next = dcll->start->next;
        }
        else
        {
            ptr->prev->next = ptr->next;
            ptr->next->prev = ptr->prev;
            if(position == 1) // deletion at beginning
                dcll->start->next = ptr->next;
        }
    }
    int data = ptr->data;
    free(ptr);
    dcll->len--;
    printf("%d was deleted from %d position of the circular doubly linked
list.\n", data, position);
    return data;
}

void introduction()
{
    printf("~ To display the DCLL, enter 1.\n\n");
    printf("INSERTION\n~ To insert an element at the start DCLL, enter 2
and the element.\n");
    printf("~ To insert an element at the end of the DCLL, enter 3 and the
element.\n");
    printf("~ To insert an element at a specific position of the DCLL,
enter 4, the element and the position.\n\n");
    printf("DELETION\n~ To delete an element from the start DCLL, enter
5.\n");
    printf("~ To delete an element from the end of the DCLL, enter 6.\n");
    printf("~ To delete an element from a specific position of the DCLL,
enter 7 and the position.\n");
    printf("\n~ To search for an element, enter 8 and the element.\n\n");
    printf("~ To print this message again, enter 9.\n");
    printf("~ To exit, enter 0.\n\n");
}
```

```c
int main()
{
    printf("\nWelcome to Circular Doubly Linked List (DCLL)
Generator!\n\n");
    introduction();
    int x = -1;
    int item, position;
    CircularDoublyLinkedList * dcll = createCircularDoublyLinkedList();
    printf("\n");
    while(x != 0)
    {
        scanf("%d", &x);
        switch (x)
        {
            case 0: printf("Exiting... Thank you!\n"); break;
            case 1: display(dcll); break;

            case 2: printf("data: "); scanf("%d", &item);
insertAtStart(dcll, item); break;
            case 3: printf("data: "); scanf("%d", &item);
insertAtEnd(dcll, item); break;
            case 4: printf("data: "); scanf("%d", &item);
printf("position: "); scanf("%d", &position); insertAtPosition(dcll, item,
position); break;

            case 5: deleteAtStart(dcll); break;
            case 6: deleteAtEnd(dcll); break;
            case 7: printf("position: "); scanf("%d", &position);
deleteAtPosition(dcll, position); break;

            case 8: printf("data to be searched: "); scanf("%d", &item);
search(dcll, item); break;

            case 9: introduction(); break;
            default: printf("Invalid value, try again.\n"); break;
        }
    }
}
```

RUNNING THE CIRCULAR DOUBLY LINKED LIST CODE TO SOLVE PROBLEM 2(b)

2- Write a program for the various cases of insertion and deletion (you can implement all in single program itself) in doubly linked list and circular doubly linked list.

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> gcc CircularDoublyLinkedList.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> ./a.exe

Welcome to Circular Doubly Linked List (DCLL) Generator!

~ To display the DCLL, enter 1.

INSERTION
~ To insert an element at the start DCLL, enter 2 and the element.
~ To insert an element at the end of the DCLL, enter 3 and the element.
~ To insert an element at a specific position of the DCLL, enter 4, the element and the position.

DELETION
~ To delete an element from the start DCLL, enter 5.
~ To delete an element from the end of the DCLL, enter 6.
~ To delete an element from a specific position of the DCLL, enter 7 and the position.

~ To search for an element, enter 8 and the element.

~ To print this message again, enter 9.
~ To exit, enter 0.

A new circular doubly linked list was created!
```

```
A new circular doubly linked list was created!

2
data: 3
3 was inserted at the start of the circular doubly linked list!
5
3 was deleted from the start of the doubly linked list.
3
data: 7
7 was inserted at the end of the circular doubly linked list!
6
7 was deleted from the end of the doubly linked list.
4
data: 6
position: 1
6 was inserted at position 1 of the circular doubly linked list!
8
data to be searched: 3
3 not found in the circular doubly linked list.
8
data to be searched: 6
6 found at 1 position in the circular doubly linked list.
7
position: 1
6 was deleted from 1 position of the circular doubly linked list.
2
data: 5
5 was inserted at the start of the circular doubly linked list!
3
data: 6
6 was inserted at the end of the circular doubly linked list!
2
data: 8
8 was inserted at the start of the circular doubly linked list!
3
data: 90
90 was inserted at the end of the circular doubly linked list!
2
data: 39
39 was inserted at the start of the circular doubly linked list!
3
```

```
39 was inserted at the start of the circular doubly linked list!
3
data: 48
48 was inserted at the end of the circular doubly linked list!
1
39       8       5       6       90       48
11474176 11474112 11474048 11474080 11474144 11474208 11474176
8
data to be searched: 90
90 found at 5 position in the circular doubly linked list.
6
48 was deleted from the end of the doubly linked list.
5
39 was deleted from the start of the doubly linked list.
7
position: 4
90 was deleted from 4 position of the circular doubly linked list.
7
position: 4
Invalid Location
7
position: 3
6 was deleted from 3 position of the circular doubly linked list.
7
position: 1
8 was deleted from 1 position of the circular doubly linked list.
6
5 was deleted from the end of the doubly linked list.
5
Circular Doubly Linked List is empty, nothing to delete
1
Circular doubly linked list is empty, nothing to display.
0
Exiting... Thank you!
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5>
```

3- Consider a scenario where n elements are present in the list. User wants to separate the list in two using one value x such that all the nodes greater than x comes at one end whereas nodes smaller than x comes to another end.

CODE FOR NODE SEPARATION:

Code is same as doubly linked list code, and has separateByX function at the end –

```c
// NodeSeparation.c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
/*  Question -
*   Consider a scenario where n elements are present in the list. User wants to separate the list in two
*   using one value x such that all the nodes greater than x comes at one end whereas nodes smaller than
*   x comes to another end
*/

/*  Soultion -
*   Here the question is solved for doubly linked lists.
*   We traverse through the linked list. If we come across a node that is greater than x, it is moved to
*   front of the linked list. The nodes less than x will automatically end up towards the end.
*   Here we have to keep in mind to move the node itself, we are not creating a newnode with the same
*   data and moving it in the front, we are changing the place of the node in the list.
*
*   Refer to the doubly linked list program (DoublyLinkedList.c).
*
*
*/

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

typedef struct DoublyLinkedList
```

```
{
    Node * start;
    int len;
}DoublyLinkedList;

Node * createNode(int item)
{
    Node * temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}

DoublyLinkedList * createDoublyLinkedList()
{
    DoublyLinkedList * dll = (DoublyLinkedList
*)malloc(sizeof(DoublyLinkedList));
    dll->len = 0;
    dll->start = (Node *)malloc(sizeof(Node));
    dll->start->data = INT_MIN; // Data in the start node is never to be
accessed. If INT_MIN is the data displayed, an error has possibly occured
    dll->start->next = NULL;
    dll->start->prev = NULL;
    printf("A new doubly linked list was created!\n");
    return dll;
}

/* Insertion */

// Inserts an element at the start of a linked list
void insertAtStart(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    if(dll->start->next != NULL)
    {
        newnode->next = dll->start->next;
        dll->start->next->prev = newnode;
    }
    dll->start->next = newnode;
    printf("%d was inserted at the start of the doubly linked list!\n",
item);
    dll->len++;
}
```

```c
// Inserts an element at the end of a doubly linked list
void insertAtEnd(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = newnode;
    if(ptr != dll->start)
        newnode->prev = ptr;
    printf("%d was inserted at the end of the doubly linked list!\n",
item);
    dll->len++;
}

// Inserts an element at a specified position.
// Here, position is determined by usual 1-base counting. If position = 5,
item will be the fifth element in the linked list
void insertAtPosition(DoublyLinkedList * dll, int item, int position)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    for(int i = 0; i < (position - 1); i++)
    {
        if(ptr->next == NULL && i + 1 < position)
        {
            printf("Invalid Location\n");
            return;
        }
        ptr = ptr->next;
    }
    if(position != 1) // If adding at beginning then newnode's prev will
remain NULL
        newnode->prev = ptr;
    if(dll->start->next != NULL) // When list is not empty
    {
        newnode->next = ptr->next;
        if(ptr->next != NULL)
            ptr->next->prev = newnode; // Skip this if adding at the end
    }
    ptr->next = newnode;
    printf("%d was inserted at position %d of the doubly linked list!\n",
item, position);
    dll->len++;
```

```c
}

/* Traversal */
/* Display function provided for illustrative purposes.*/
void display(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to display.\n");
        return;
    }
    // Printing data
    Node * ptr = dll->start->next;
    printf("Start: %d \n", dll->start);
    while(ptr != NULL)
    {
        printf("%d       ", ptr->data);
        ptr = ptr->next;
    }
    // Printing the location
    Node * ptr2 = dll->start->next;
    printf("\n%d ", dll->start->next);
    while(ptr2 != NULL)
    {
        printf("%d ", ptr2->next);
        ptr2 = ptr2->next;
    }
    printf("\n");
}

// Returns the 1-base position of the first occurance of data.
int search(DoublyLinkedList * dll, int item)
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to search for.\n");
        return INT_MIN;
    }
    int count = 1;
    Node * ptr = dll->start->next;
    while(ptr->data != item)
    {
        if(ptr->data != item && ptr->next == NULL)
        {
```

```
            printf("%d not found in the doubly linked list.\n", item);
            return INT_MIN;
        }
        ptr = ptr->next;
        count++;
    }
    printf("%d found at %d position in the doubly linked list.\n", item,
count);
    return count;
}

/* Deletion */

// Deletion at start
int deleteAtStart(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete.\n");
        return INT_MIN;
    }
    Node * ptr = dll->start->next;
    dll->start->next = ptr->next;
    if(ptr->next != NULL)
        ptr->next->prev = NULL;
    int data = ptr->data;
    free(ptr);
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    dll->len--;
    return data;
}

// Deletion at end
int deleteAtEnd(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
    }
    int data;
    Node * ptr = dll->start->next;
    while(ptr->next != NULL)
```

```
        ptr = ptr->next;
    if(ptr->prev == NULL) // Only one element is present
        dll->start->next = NULL;
    else // Other cases
        ptr->prev->next = NULL;
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    return data;
}
// Deletion of an element at a particular position
// Here, position is determined by usual 1-base counting.
int deleteAtPosition(DoublyLinkedList * dll, int position)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
    }
    Node * ptr = dll->start;
    int data;
    for(int i = 0; i < position; i++)
    {
        if(ptr->next == NULL && i < position)
        {
            printf("Invalid Location\n");
            return INT_MIN;
        }
        ptr = ptr->next;
    }

    if(ptr->prev == NULL) // If first element is to be deleted
        dll->start->next = ptr->next;
    else
        ptr->prev->next = ptr->next;

    if(ptr->next != NULL)
        ptr->next->prev = ptr->prev;
    data = ptr->data;
    free(ptr);
    dll->len--;
```

```
    printf("%d was deleted from %d position of the doubly linked list.\n",
data, position);
    return data;
}

void separateByX(DoublyLinkedList * dll, int x)
{
    printf("Separating Nodes by %d\n", x);
    Node * ptr = dll->start->next;
    Node * ptr1;
    while(ptr != NULL)
    {
        ptr1 = ptr->next;
        if(ptr->data > x)
        {
            if(ptr->prev != NULL)
            {
                ptr->prev->next = ptr->next;
                ptr->next->prev = ptr->prev;
                ptr->next = dll->start->next;
                ptr->prev = NULL;
                dll->start->next->prev = ptr;
                dll->start->next = ptr;
            }
        }
        ptr = ptr1;
    }
}

int main()
{
    DoublyLinkedList * dll = createDoublyLinkedList();
    insertAtEnd(dll, 30);
    insertAtEnd(dll, 25);
    insertAtEnd(dll, 65);
    insertAtEnd(dll, 20);
    insertAtEnd(dll, 5);
    insertAtEnd(dll, 75);
    insertAtEnd(dll, 10);
    insertAtEnd(dll, 20);
    display(dll);
    separateByX(dll, 25);
    display(dll);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> gcc NodeSeparation.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> ./a.exe
A new doubly linked list was created!
30 was inserted at the end of the doubly linked list!
25 was inserted at the end of the doubly linked list!
65 was inserted at the end of the doubly linked list!
20 was inserted at the end of the doubly linked list!
5 was inserted at the end of the doubly linked list!
75 was inserted at the end of the doubly linked list!
10 was inserted at the end of the doubly linked list!
20 was inserted at the end of the doubly linked list!
Start: 10163296
30      25      65      20      5       75      10      20
10163328 10163360 10163392 10163424 10163456 10163488 10163520 10163552 0
Separating Nodes by 25
Start: 10163296
75      65      30      25      20      5       10      20
10163488 10163392 10163328 10163360 10163424 10163456 10163520 10163552 0
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5>
```

4- Consider one scenario where two lists is given and needs to be merged in a single list. Write a program for the given scenario.

CODE FOR MERGING DOUBLY LINKED LISTS:

Code is same as doubly linked list code, and has join function at the end –

```c
// MergeDoublyLinkedList.c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
/*  Question -
*   Consider one scenario where two lists is given and needs to be merged
in a single list. Write a
*   program for the given scenario.
*/

/*  Soultion -
*   We join the last element to the first element of the second list and
delete the second list's start.
*   Refer to the doubly linked list program (DoublyLinkedList.c).
*
*
*/

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

```
    struct Node * next;
}Node;

typedef struct DoublyLinkedList
{
    Node * start;
    int len;
}DoublyLinkedList;

Node * createNode(int item)
{
    Node * temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}

DoublyLinkedList * createDoublyLinkedList()
{
    DoublyLinkedList * dll = (DoublyLinkedList
*)malloc(sizeof(DoublyLinkedList));
    dll->len = 0;
    dll->start = (Node *)malloc(sizeof(Node));
    dll->start->data = INT_MIN; // Data in the start node is never to be
accessed. If INT_MIN is the data displayed, an error has possibly occured
    dll->start->next = NULL;
    dll->start->prev = NULL;
    printf("A new doubly linked list was created!\n");
    return dll;
}

/* Insertion */

// Inserts an element at the start of a linked list
void insertAtStart(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    if(dll->start->next != NULL)
    {
        newnode->next = dll->start->next;
        dll->start->next->prev = newnode;
    }
    dll->start->next = newnode;
```

```
    printf("%d was inserted at the start of the doubly linked list!\n",
item);
    dll->len++;
}
// Inserts an element at the end of a doubly linked list
void insertAtEnd(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = newnode;
    if(ptr != dll->start)
        newnode->prev = ptr;
    printf("%d was inserted at the end of the doubly linked list!\n",
item);
    dll->len++;
}

// Inserts an element at a specified position.
// Here, position is determined by usual 1-base counting. If position = 5,
item will be the fifth element in the linked list
void insertAtPosition(DoublyLinkedList * dll, int item, int position)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    for(int i = 0; i < (position - 1); i++)
    {
        if(ptr->next == NULL && i + 1 < position)
        {
            printf("Invalid Location\n");
            return;
        }
        ptr = ptr->next;
    }
    if(position != 1) // If adding at beginning then newnode's prev will
remain NULL
        newnode->prev = ptr;
    if(dll->start->next != NULL) // When list is not empty
    {
        newnode->next = ptr->next;
        if(ptr->next != NULL)
            ptr->next->prev = newnode; // Skip this if adding at the end
    }
```

```
    ptr->next = newnode;
    printf("%d was inserted at position %d of the doubly linked list!\n",
item, position);
    dll->len++;
}

/* Traversal */
/* Display function provided for illustrative purposes.*/
void display(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to display.\n");
        return;
    }
    // Printing data
    Node * ptr = dll->start->next;
    printf("Start: %d \n", dll->start);
    while(ptr != NULL)
    {
        printf("%d       ", ptr->data);
        ptr = ptr->next;
    }
    // Printing the location
    Node * ptr2 = dll->start->next;
    printf("\n%d ", dll->start->next);
    while(ptr2 != NULL)
    {
        printf("%d ", ptr2->next);
        ptr2 = ptr2->next;
    }
    printf("\n");
}

// Returns the 1-base position of the first occurance of data.
int search(DoublyLinkedList * dll, int item)
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to search for.\n");
        return INT_MIN;
    }
    int count = 1;
    Node * ptr = dll->start->next;
```

```
    while(ptr->data != item)
    {
        if(ptr->data != item && ptr->next == NULL)
        {
            printf("%d not found in the doubly linked list.\n", item);
            return INT_MIN;
        }
        ptr = ptr->next;
        count++;
    }
    printf("%d found at %d position in the doubly linked list.\n", item,
count);
    return count;
}

/* Deletion */

// Deletion at start
int deleteAtStart(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete.\n");
        return INT_MIN;
    }
    Node * ptr = dll->start->next;
    dll->start->next = ptr->next;
    if(ptr->next != NULL)
        ptr->next->prev = NULL;
    int data = ptr->data;
    free(ptr);
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    dll->len--;
    return data;
}

// Deletion at end
int deleteAtEnd(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
```

```
    }
    int data;
    Node * ptr = dll->start->next;
    while(ptr->next != NULL)
        ptr = ptr->next;
    if(ptr->prev == NULL) // Only one element is present
        dll->start->next = NULL;
    else // Other cases
        ptr->prev->next = NULL;
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    return data;
}
// Deletion of an element at a particular position
// Here, position is determined by usual 1-base counting.
int deleteAtPosition(DoublyLinkedList * dll, int position)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
    }
    Node * ptr = dll->start;
    int data;
    for(int i = 0; i < position; i++)
    {
        if(ptr->next == NULL && i < position)
        {
            printf("Invalid Location\n");
            return INT_MIN;
        }
        ptr = ptr->next;
    }

    if(ptr->prev == NULL) // If first element is to be deleted
        dll->start->next = ptr->next;
    else
        ptr->prev->next = ptr->next;

    if(ptr->next != NULL)
        ptr->next->prev = ptr->prev;
```

```
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from %d position of the doubly linked list.\n",
data, position);
    return data;
}

// Join two doubly linked lists
void joinDoublyLinkedLists(DoublyLinkedList * dll1, DoublyLinkedList *
dll2)
{
    printf("Joining doubly linked lists...\n");
    Node * ptr = dll1->start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = dll2->start->next;
    dll2->start->next->prev = ptr;
    free(dll2->start);
    free(dll2);
}

int main()
{
    DoublyLinkedList * dll = createDoublyLinkedList();
    insertAtEnd(dll, 30);
    insertAtEnd(dll, 25);
    insertAtEnd(dll, 65);
    insertAtEnd(dll, 20);
    insertAtEnd(dll, 5);
    insertAtEnd(dll, 75);
    insertAtEnd(dll, 10);
    insertAtEnd(dll, 20);
    display(dll);

    DoublyLinkedList * dll1 = createDoublyLinkedList();
    insertAtEnd(dll1, 20);
    insertAtEnd(dll1, 85);
    insertAtEnd(dll1, 61);
    insertAtEnd(dll1, 20);
    insertAtEnd(dll1, 56);
    insertAtEnd(dll1, 75);
    insertAtEnd(dll1, 17);
    insertAtEnd(dll1, 90);
```

```
    display(dll1);

    joinDoublyLinkedLists(dll, dll1);
    display(dll);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> gcc MergeDoublyLinkedList.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> ./a.exe
A new doubly linked list was created!
30 was inserted at the end of the doubly linked list!
25 was inserted at the end of the doubly linked list!
65 was inserted at the end of the doubly linked list!
20 was inserted at the end of the doubly linked list!
5 was inserted at the end of the doubly linked list!
75 was inserted at the end of the doubly linked list!
10 was inserted at the end of the doubly linked list!
20 was inserted at the end of the doubly linked list!
Start: 11408480
30      25      65      20      5       75      10      20
11408512 11408544 11408576 11408608 11408640 11408672 11408704 11408736 0
A new doubly linked list was created!
20 was inserted at the end of the doubly linked list!
85 was inserted at the end of the doubly linked list!
61 was inserted at the end of the doubly linked list!
20 was inserted at the end of the doubly linked list!
56 was inserted at the end of the doubly linked list!
75 was inserted at the end of the doubly linked list!
17 was inserted at the end of the doubly linked list!
90 was inserted at the end of the doubly linked list!
Start: 11408800
20      85      61      20      56      75      17      90
11408832 11408864 11409712 11409616 11409488 11409360 11409008 11409232 0
Joining doubly linked lists...
Start: 11408480
30      25      65      20      5       75      10      20      20      85      61      20      56      75      17      90
11408512 11408544 11408576 11408608 11408640 11408672 11408704 11408736 11408832 11408864 11409712 11409616 11409488 11409360 11409008 11409232 0
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5>
```

5- There are few elements is given in the list. Given elements are not in a sorted way. Write one function to reverse the given list (using recursion and without using recursion).

CODE FOR REVERSING DOUBLY LINKED LISTS:

Code is same as reversing linked list code, and has reverse functions at the end(iterative method and recursive method) –

```
// ReverseDoublyLinkedList.c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
/*  Question -
*   There are few elements is given in the list. Given elements are not in
a sorted way. Write one
*   function to reverse the given list (using recursion and without using
recursion)
*/
typedef struct Node{
    int data;
    struct Node * prev;
    struct Node * next;
```

```c
}Node;

typedef struct DoublyLinkedList
{
    Node * start;
    int len;
}DoublyLinkedList;

Node * createNode(int item)
{
    Node * temp = (Node *)malloc(sizeof(Node));
    temp->data = item;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}

DoublyLinkedList * createDoublyLinkedList()
{
    DoublyLinkedList * dll = (DoublyLinkedList
*)malloc(sizeof(DoublyLinkedList));
    dll->len = 0;
    dll->start = (Node *)malloc(sizeof(Node));
    dll->start->data = INT_MIN; // Data in the start node is never to be
accessed. If INT_MIN is the data displayed, an error has possibly occured
    dll->start->next = NULL;
    dll->start->prev = NULL;
    printf("A new doubly linked list was created!\n");
    return dll;
}

/* Insertion */

// Inserts an element at the start of a linked list
void insertAtStart(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    if(dll->start->next != NULL)
    {
        newnode->next = dll->start->next;
        dll->start->next->prev = newnode;
    }
    dll->start->next = newnode;
```

```
    printf("%d was inserted at the start of the doubly linked list!\n",
item);
    dll->len++;
}
// Inserts an element at the end of a doubly linked list
void insertAtEnd(DoublyLinkedList * dll, int item)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = newnode;
    if(ptr != dll->start)
        newnode->prev = ptr;
    printf("%d was inserted at the end of the doubly linked list!\n",
item);
    dll->len++;
}

// Inserts an element at a specified position.
// Here, position is determined by usual 1-base counting. If position = 5,
item will be the fifth element in the linked list
void insertAtPosition(DoublyLinkedList * dll, int item, int position)
{
    Node * newnode = createNode(item);
    Node * ptr = dll->start;
    for(int i = 0; i < (position - 1); i++)
    {
        if(ptr->next == NULL && i + 1 < position)
        {
            printf("Invalid Location\n");
            return;
        }
        ptr = ptr->next;
    }
    if(position != 1) // If adding at beginning then newnode's prev will
remain NULL
        newnode->prev = ptr;
    if(dll->start->next != NULL) // When list is not empty
    {
        newnode->next = ptr->next;
        if(ptr->next != NULL)
            ptr->next->prev = newnode; // Skip this if adding at the end
    }
```

```
    ptr->next = newnode;
    printf("%d was inserted at position %d of the doubly linked list!\n",
item, position);
    dll->len++;
}

/* Traversal */
/* Display function provided for illustrative purposes.*/
void display(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to display.\n");
        return;
    }
    // Printing data
    Node * ptr = dll->start->next;
    printf("Start: %d \n", dll->start);
    while(ptr != NULL)
    {
        printf("%d       ", ptr->data);
        ptr = ptr->next;
    }
    // Printing the location
    Node * ptr2 = dll->start->next;
    printf("\n%d ", dll->start->next);
    while(ptr2 != NULL)
    {
        printf("%d ", ptr2->next);
        ptr2 = ptr2->next;
    }
    printf("\n");
}

// Returns the 1-base position of the first occurance of data.
int search(DoublyLinkedList * dll, int item)
{
    if(dll->start->next == NULL)
    {
        printf("Empty Doubly Linked List, nothing to search for.\n");
        return INT_MIN;
    }
    int count = 1;
    Node * ptr = dll->start->next;
```

```
    while(ptr->data != item)
    {
        if(ptr->data != item && ptr->next == NULL)
        {
            printf("%d not found in the doubly linked list.\n", item);
            return INT_MIN;
        }
        ptr = ptr->next;
        count++;
    }
    printf("%d found at %d position in the doubly linked list.\n", item,
count);
    return count;
}

/* Deletion */

// Deletion at start
int deleteAtStart(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete.\n");
        return INT_MIN;
    }
    Node * ptr = dll->start->next;
    dll->start->next = ptr->next;
    if(ptr->next != NULL)
        ptr->next->prev = NULL;
    int data = ptr->data;
    free(ptr);
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    dll->len--;
    return data;
}

// Deletion at end
int deleteAtEnd(DoublyLinkedList * dll)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
```

```
    }
    int data;
    Node * ptr = dll->start->next;
    while(ptr->next != NULL)
        ptr = ptr->next;
    if(ptr->prev == NULL) // Only one element is present
        dll->start->next = NULL;
    else // Other cases
        ptr->prev->next = NULL;
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from the start of the doubly linked list.\n",
data);
    return data;
}
// Deletion of an element at a particular position
// Here, position is determined by usual 1-base counting.
int deleteAtPosition(DoublyLinkedList * dll, int position)
{
    if(dll->start->next == NULL)
    {
        printf("Doubly Linked List is empty, nothing to delete\n");
        return INT_MIN;
    }
    Node * ptr = dll->start;
    int data;
    for(int i = 0; i < position; i++)
    {
        if(ptr->next == NULL && i < position)
        {
            printf("Invalid Location\n");
            return INT_MIN;
        }
        ptr = ptr->next;
    }

    if(ptr->prev == NULL) // If first element is to be deleted
        dll->start->next = ptr->next;
    else
        ptr->prev->next = ptr->next;

    if(ptr->next != NULL)
        ptr->next->prev = ptr->prev;
```

```
    data = ptr->data;
    free(ptr);
    dll->len--;
    printf("%d was deleted from %d position of the doubly linked list.\n",
data, position);
    return data;
}


// Reverse a doubly linked list by iterative method
void reverse_1(DoublyLinkedList * dll)
{
    printf("Reversing doubly linked list by iterative method\n");
    Node * ptr = dll->start->next->next;
    Node * ptr1;
    while(ptr != NULL)
    {
        ptr1 = ptr->next;
        if(ptr->prev != NULL)
        {
            ptr->prev->next = ptr->next;
            if(ptr->next != NULL)
                ptr->next->prev = ptr->prev;
            ptr->next = dll->start->next;
            ptr->prev = NULL;
            dll->start->next->prev = ptr;
            dll->start->next = ptr;
        }
        ptr = ptr1;
    }
}

// Reverse a doubly linked list with recursion
void reverse_2(Node * ptr, Node * start)
{
    if(ptr->next == NULL)
    {
        printf("Reversing doubly linked list using recursion\n");
        start->next = ptr;
        return;
    }
    reverse_2(ptr->next, start);
    ptr->next->prev = ptr->next->next;
    ptr->next->next = ptr;
```

```c
    if(ptr->prev == NULL)
    {
        ptr->prev = ptr->next;
        ptr->next = NULL;
    }
}
int main()
{
    DoublyLinkedList * dll = createDoublyLinkedList();
    insertAtEnd(dll, 30);
    insertAtEnd(dll, 25);
    insertAtEnd(dll, 65);
    insertAtEnd(dll, 20);
    insertAtEnd(dll, 57);
    insertAtEnd(dll, 75);
    insertAtEnd(dll, 10);
    insertAtEnd(dll, 29);
    display(dll);
    // Reversing using iterative method
    reverse_1(dll);
    display(dll);
    // Reversing using recursive method
    reverse_2(dll->start->next, dll->start);
    display(dll);
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> gcc ReverseDoublyLinkedList.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5> ./a.exe
A new doubly linked list was created!
30 was inserted at the end of the doubly linked list!
25 was inserted at the end of the doubly linked list!
65 was inserted at the end of the doubly linked list!
20 was inserted at the end of the doubly linked list!
57 was inserted at the end of the doubly linked list!
75 was inserted at the end of the doubly linked list!
10 was inserted at the end of the doubly linked list!
29 was inserted at the end of the doubly linked list!
Start: 10556512
30      25      65      20      57      75      10      29
10556544 10556576 10556608 10556640 10556672 10556704 10556736 10556768 0
Reversing doubly linked list by iterative method
Start: 10556512
29      10      75      57      20      65      25      30
10556768 10556736 10556704 10556672 10556640 10556608 10556576 10556544 0
Reversing doubly linked list using recursion
Start: 10556512
30      25      65      20      57      75      10      29
10556544 10556576 10556608 10556640 10556672 10556704 10556736 10556768 0
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_5>
```