**Batch: C3 Roll No.: 16010123217**

**Experiment No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:** Implementation of various types of LL- doubly LL, circular LL, circular doubly LL |

**Objective:** To understand the use of linked lists as data structures for various applications.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO 2** | Apply linear and non-linear data structure in application development. |

**Books/ Journals/ Websites referred:**

[**https://youtube.com/playlist?list=PLdo5W4Nhv31bbKJzrsKfMpo\_grxuLl8LU&si=dBEVB0nzG6cmiRfJ**](https://youtube.com/playlist?list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&si=dBEVB0nzG6cmiRfJ)

**Swati Mali Mam notes**

**Introduction:**

Define Linked List

Ans.

Definition:

* An ordered collection of homogenous data items
* Where elements can be added anywhere and removed from anywhere

Operations:

1. Create an empty linked List
2. check if it is empty
3. Insert: add an element at the desired position
4. Delete: remove the desired element
5. Destroy : remove all the elements one by one and destroy the data structure

**Types of linked list:**

There are three Types of Linked List

1. Circular Linked List : In this Type of Linked List, the last node is connected to first node.

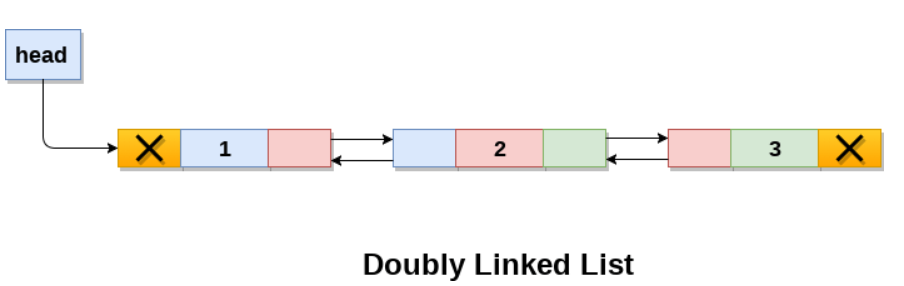


Used in Music Playlist

1. Doubly Linked List:

In a Doubly Linked List each item points to both its predecessor and successor

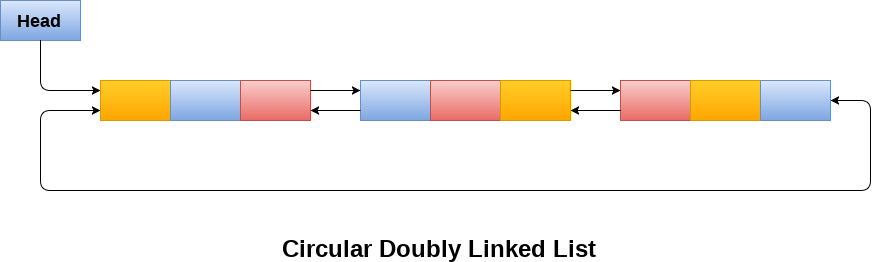
* prev points to the predecessor
* next points to the successor



* Doubly linked lists are used in web page navigation in both forward and backward directions.

1. Circular Doubly Linked List:

Ans. Circular Doubly Linked List is mixture of doubly Linked List and Circular Linked List. As doubly linked list contains prev and next it also contains prev and next and last node next is connected to first node prev.



It requires more space per node and more time complexity for basic operations

(CLL, DLL, CDLL)

**Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:**

**Assigned CLL Unsorted**

struct node{

int data;

struct node \*next;

}

Creation:

Algorithm CLLType CreateCLL(){

createNode(head);

head = NULL;

}

Insertion:

//Insert at Beginning

Algorithm CLLType InsertBeg( ElementType ele){

createNode(newNode);

newNode -> data = ele;

newNode ->next = NULL;

if( head == NULL){

head = newNode;

newNode->next = head;

}

else {

createNode(temp);

while (temp->next != head) {

temp = temp->next;

}

newNode->next = head;

temp->next = newNode;

head = newNode;

}

print(“Inserted at beginning");

}

//Insert at End

Algorithm CLLType InsertEnd(ElementType element) {

createNode(newNode);

newNode->data = element;

if (head == NULL) {

head = newNode;

newNode->next = head;

} else {

createNode(temp);

temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

print(“Inserted at end");

}

// Insertion in the middle (after a given position)

Algorithm CLLType Insertmid( ElementType element, int position) {

createNode(newNode);

newNode->data = element;

if (position == 0 || head == NULL) {

insertbeg(element);

return;

}

struct node \*temp = head;

for (int i = 0; i < position - 1 && temp->next != head; i++) {

temp = temp->next;

}

newNode->next = temp->next;

temp->next = newNode;

print(“Inserted at position %d\n", position);

}

Deletion:

// Deletion at the beginning

void deletebeg() {

if (head == NULL) {

printf("List is empty.\n");

return;

}

struct node \*temp = head;

if (head->next == head) {

free(head);

head = NULL;

} else {

struct node \*last = head;

while (last->next != head) {

last = last->next;

}

head = head->next;

last->next = head;

free(temp);

}

printf("Element deleted from the beginning.\n");

}

// Deletion at the end

Algorithm ElementType deleteEnd() {

if (head == NULL) {

printf("List is empty.\n");

return;

}

createNode(temp);

temp = head;

if (head->next == head) {

free(head);

head = NULL;

} else {

createNode(prev);

while (temp->next != head) {

prev = temp;

temp = temp->next;

}

prev->next = head;

free(temp);

}

printf("Element deleted from the end.\n");

}

// Deletion in the middle (by position)

Algorithm ElementType deleteMid(int position) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

if (position == 0) {

deletebeg();

return;

}

createNode(temp);

temp = head;

createNode(prev);

for (int i = 0; i < position && temp->next != head; i++) {

prev = temp;

temp = temp->next;

}

if (temp == head) {

printf("Invalid position.\n");

} else {

prev->next = temp->next;

free(temp);

printf("Element deleted at position %d.\n", position);

}

}

Transversal :

Algorithm CLLType Transverse(){

createNode(temp);

temp = head;

while(temp->next != head){

print(temp->data);

temp = temp ->next;

}

print(temp->data); //Last element printing

}

Searching:

Algorithm Search( CLLType cll, ElementType ele ){

createNode(temp);

temp = head;

do{

if(temp-> data == ele){

print(“Element found”);

return;

}

temp = temp ->next;

}

While(temp != head);

Print(“Element not found”);

}

**Program**

#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node \*next;

};

struct node \*head = NULL;

*// Insertion at the beginning*

void insertbeg(int element) {

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->data = element;

    if (head == NULL) {

        head = newNode;

        newNode->next = head;

    } else {

        struct node \*temp = head;

        while (temp->next != head) {

            temp = temp->next;

        }

        newNode->next = head;

        temp->next = newNode;

        head = newNode;

    }

    printf("%d inserted at beginning\n", element);

}

*// Insertion at the end*

void insertend(int element) {

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->data = element;

    if (head == NULL) {

        head = newNode;

        newNode->next = head;

    } else {

        struct node \*temp = head;

        while (temp->next != head) {

            temp = temp->next;

        }

        temp->next = newNode;

        newNode->next = head;

    }

    printf("%d inserted at end\n", element);

}

*// Insertion at a specific position*

void insertmid(int element, int position) {

    struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

    newNode->data = element;

    if (position == 1 || head == NULL) {

        insertbeg(element);

        return;

    }

    struct node \*temp = head;

    for (int i = 1; i < position - 1 && temp->next != head; i++) {

        temp = temp->next;

    }

    if (temp->next == head && position > 1) {

        printf("Position out of bounds, inserting at the end.\n");

        insertend(element);

        return;

    }

    newNode->next = temp->next;

    temp->next = newNode;

    printf("%d inserted at position %d\n", element, position);

}

*// Traversing the circular linked list*

void transverse() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct node \*temp = head;

    do {

        printf("%d -> ", temp->data);

        temp = temp->next;

    } while (temp != head);

    printf("HEAD\n");

}

*// Searching for an element in the circular linked list*

void search(int element) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct node \*temp = head;

    int index = 1;

    do {

        if (temp->data == element) {

            printf("Element %d found at index %d\n", element, index);

            return;

        }

        temp = temp->next;

        index++;

    } while (temp != head);

    printf("Element %d not found.\n", element);

}

*// Deletion at the beginning*

void deletebeg() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct node \*temp = head;

    if (head->next == head) {

        free(head);

        head = NULL;

    } else {

        struct node \*last = head;

        while (last->next != head) {

            last = last->next;

        }

        head = head->next;

        last->next = head;

        free(temp);

    }

    printf("Element deleted from the beginning.\n");

}

*// Deletion at the end*

void deleteend() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct node \*temp = head;

    if (head->next == head) {

        free(head);

        head = NULL;

    } else {

        struct node \*prev = NULL;

        while (temp->next != head) {

            prev = temp;

            temp = temp->next;

        }

        prev->next = head;

        free(temp);

    }

    printf("Element deleted from the end.\n");

}

*// Deletion at a specific position*

void deletemid(int position) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    if (position == 1) {

        deletebeg();

        return;

    }

    struct node \*temp = head;

    struct node \*prev = NULL;

    for (int i = 1; i < position && temp->next != head; i++) {

        prev = temp;

        temp = temp->next;

    }

    if (temp == head || temp->next == head && position > 1) {

        printf("Invalid position.\n");

    } else {

        prev->next = temp->next;

        free(temp);

        printf("Element deleted at position %d.\n", position);

    }

}

*// Menu-driven program*

int main() {

    int choice, element, position;

    while (1) {

        printf("\nCircular Linked List Menu:\n");

        printf("1. Insert at beginning\n");

        printf("2. Insert at end\n");

        printf("3. Insert at position\n");

        printf("4. Delete from beginning\n");

        printf("5. Delete from end\n");

        printf("6. Delete from position\n");

        printf("7. Traverse list\n");

        printf("8. Search element\n");

        printf("9. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter element to insert at beginning: ");

                scanf("%d", &element);

                insertbeg(element);

                break;

            case 2:

                printf("Enter element to insert at end: ");

                scanf("%d", &element);

                insertend(element);

                break;

            case 3:

                printf("Enter element to insert: ");

                scanf("%d", &element);

                printf("Enter position : ");

                scanf("%d", &position);

                insertmid(element, position);

                break;

            case 4:

                deletebeg();

                break;

            case 5:

                deleteend();

                break;

            case 6:

                printf("Enter position to delete: ");

                scanf("%d", &position);

                deletemid(position);

                break;

            case 7:

                transverse();

                break;

            case 8:

                printf("Enter element to search: ");

                scanf("%d", &element);

                search(element);

                break;

            case 9:

                printf("Exiting...\n");

                exit(0);

            default:

                printf("Invalid choice! Please try again.\n");

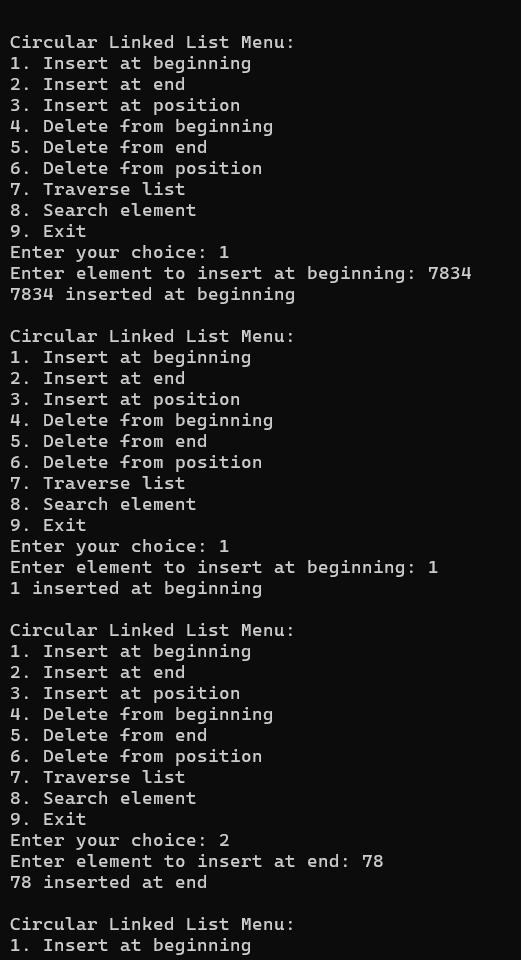
        }

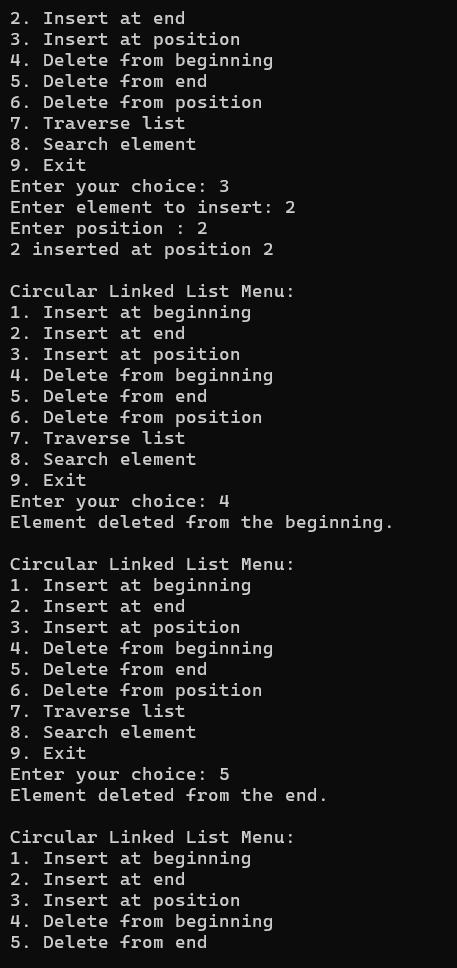
    }

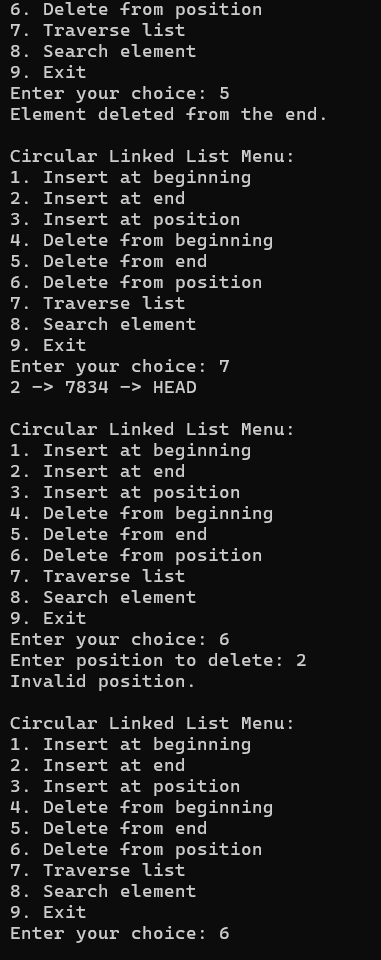
    return 0;

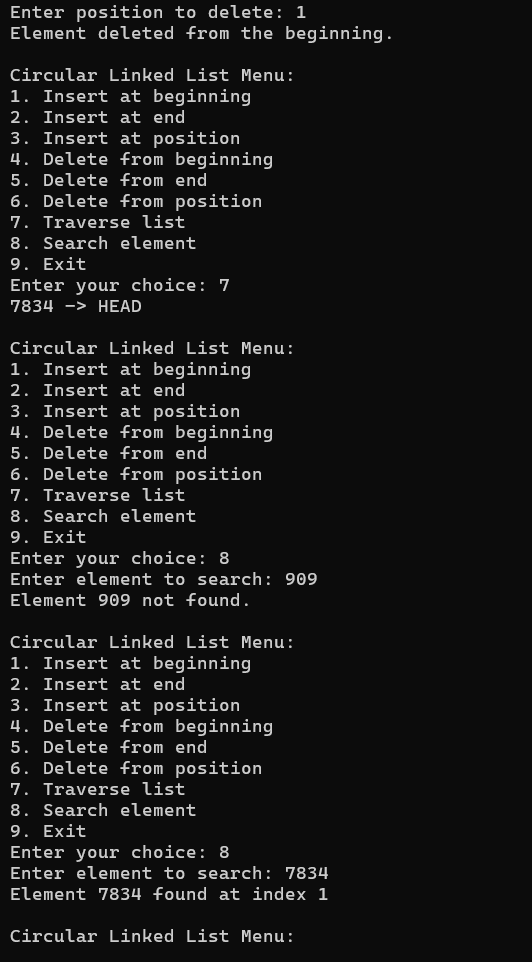
}

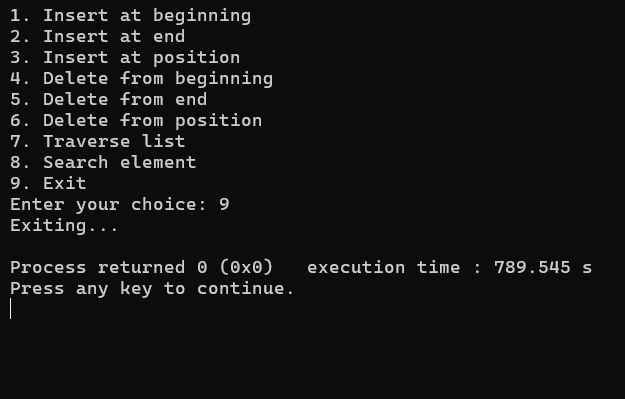
**Output:-**











**Conclusion:-**

In this experiment we learnt about different types of linked lists their uses and specific features.

**Post lab questions:**

1. Compare and contrast SLL and DLL.

Ans.Singly Linked List (SLL) Doubly Linked List (DLL)

|  |  |
| --- | --- |
| Node contains 2 field – data field and next link field. | Node contains 3 fields – data field, previous link field and next link field. |
| Transversal is possible only in one direction. | Transversal is possible in both directions (forward and backward). |
| Occupies less space than DLL. | Occupies more space than SLL. |
| Deletion takes O(n) time complexity because the previous node needs to known. | Deletion takes O(1) because the previous node can be accessed easily. |
| Less used in practice due to limited number of operations. | More used in libraries due to wider number of operations. |

2. List any 3 scenarios where circular linked lists are preferable over linear linked lists?

Ans.

Data streaming : To overwrite old data with new data when capacity is reached.

Token Passing in Networks : A circular linked list is used to represent the nodes in the network with each node passing a token to the next node in a continuous loop.

Repeating Playlist: In media players, when we loop the playlist it gets automatically to the start when it reaches the end.

3. How would you implement a function to reverse a doubly linked list?

Ans. To reverse a linked list

1. Take 2 pointers ptr1 and ptr 2
2. ptr1 = head and ptr2 = ptr1-> next;
3. ptr1 ->next = NULL; and ptr1->prev = ptr2; //first node to make it last
4. while(ptr2 != NULL)

{

ptr2->prev = ptr2->next;

ptr2->next = ptr1;

ptr1 = ptr2;

ptr2 = ptr2->prev;

}

head = ptr1;

return head;

1. What are some practical applications of circular doubly linked lists in real-world systems? How does their structure provide advantages in these scenarios?

Ans. Music or Media Playlists

Advantage: A circular doubly linked list allows seamless forward and backward navigation between songs or media items. When the end is reached, it loops back to the start, making playlist management simple and efficient without special checks for start or end conditions.

Gaming (For example UNO)

Advantage: A circular doubly linked list provides a circular path without stopping at the last player and the game continues. As UNO has reverse cards so reversing is also possible in doubly linked list

Carousel Displays in User Interfaces

Advantage: Circular doubly linked lists naturally handle the circular navigation in these displays, allowing both forward and backward scrolling without requiring additional logic to restart at the beginning or end of the list.