# AIM: IMPLEMENTATION OF CIRCULAR SINGLY LINKED LIST

## DESCRIPTION:-

A **Circular Singly Linked List** is a type of linked list in which all nodes are connected in a circular manner, meaning the last node points back to the first node instead of having a null reference. This structure allows for continuous traversal of the list without needing to start over from the head.

### Key Characteristics:

- **Head Pointer**: Points to the first node of the list.
- **Tail Pointer**: The last node's next pointer points to the head, creating a circular link.
- **Traversal**: Can start from any node and continue indefinitely, making it useful for applications where circular traversal is required, such as round-robin scheduling or managing playlists in media players.

### Operations:

1. **Insertion**:
    - **At the Beginning**: A new node is created, and its next pointer is set to the current head. The last node’s next pointer is updated to point to the new node, and the head pointer is updated.
    - **At the End**: Similar to insertion at the beginning, but the new node’s next pointer is set to point to the head, and the current last node’s next pointer is updated to point to the new node.
2. **Deletion**:
    - **From the Beginning**: The head pointer is moved to the next node, and the last node's next pointer is updated accordingly.
    - **From the End**: Requires traversal to find the second-to-last node to adjust its next pointer.
3. **Traversal**: A loop is established to traverse through the list until the starting point is reached again.

Circular singly linked lists are particularly useful in applications requiring repeated or circular access to elements, as they allow for efficient use of memory and resources without additional overhead of resetting pointers.

**CODE:-**

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

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      newNode->next = newNode;
   } else {
      struct Node* temp = *head;
      while (temp->next != *head) {
         temp = temp->next;
      }
      temp->next = newNode;
      newNode->next = *head;
      *head = newNode;
   }
}

void insertAtEnd(struct Node** head, int data) {
```

```
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
    }
}

void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = *head;
    if (temp->next == *head) {
        free(temp);
        *head = NULL;
    } else {
        struct Node* last = *head;
        while (last->next != *head) {
            last = last->next;
        }
        last->next = temp->next;
        *head = temp->next;
        free(temp);
    }
}

void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
```

```
        return;
    }

    struct Node* temp = *head;
    if (temp->next == *head) {
        free(temp);
        *head = NULL;
        return;
    }

    struct Node* secondLast = NULL;
    while (temp->next != *head) {
        secondLast = temp;
        temp = temp->next;
    }
    secondLast->next = *head;
    free(temp);
}

void display(struct Node* head) {
    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("(back to head)\n");
}

void circularLinkedListMenu() {
    struct Node* head = NULL;
    int choice, data;

    do {
```

```
        printf("\nCircular Singly Linked List Menu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Delete from Beginning\n");
        printf("4. Delete from End\n");
        printf("5. Display List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                deleteFromBeginning(&head);
                break;
            case 4:
                deleteFromEnd(&head);
                break;
            case 5:
                display(head);
                break;
            case 6:
                printf("Exiting Circular Singly Linked List Menu...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 6);
}
```

```
int main() {
   circularLinkedListMenu();
   return 0;
}
```

**OUTPUT:-**

```
Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 1
Enter data to insert at the beginning: 66

Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 1
Enter data to insert at the beginning: 55

Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 2
Enter data to insert at the end: 77
```

```
Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 2
Enter data to insert at the end: 99

Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 5
55 -> 66 -> 77 -> 99 -> (back to head)

Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 3
```

```
Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 4

Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 5
66 -> 77 -> (back to head)

Circular Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display List
6. Exit
Enter your choice: 6
Exiting Circular Singly Linked List Menu...
```