# EXPERIMENT - 15

# Aim: Write a menu driven program to show Circular linked list implementation. It should consist of delete from head , delete from end & delete from middle.

## Pseudo code

Initialize head pointer to NULL

Function InsertAtTail(head, val):

Create a new node n with data = val

If head is NULL:

Set head = n

Set n->next = n

Else:

Initialize temp = head

While temp->next is not equal to head:

Move temp to the next node

Set temp->next = n

Set n->next = head

Function InsertAtHead(head, val):

Create a new node n with data = val

If head is NULL:

Set head = n

Set n->next = n

Else:

Initialize temp = head

While temp->next is not equal to head:

Move temp to the next node

Set temp->next = n

Set n->next = head

Set head = n

Function InsertAtMiddle(head, val, position):

Create a new node n with data = val

If head is NULL or position is 1:

Call InsertAtHead(head, val)

Else:

Initialize temp = head

Initialize count = 1

While temp->next->data is not equal to val and count < position - 1:

Move temp to the next node

Increment count

Set n->next = temp->next

Set temp->next = n

Function DeleteAtEnd(head):

If head is NULL:

Return

Initialize temp = head

While temp->next->next is not equal to head:

Move temp to the next node

Delete the last node (temp->next)

Set temp->next = head

Function DeleteAtHead(head):

If head is NULL:

Return

Initialize temp = head

While temp->next is not equal to head:

Move temp to the next node

Set head = head->next

Free the memory of the original head node

Set temp->next = head

Function DeleteAtMiddle(head, val):

If head is NULL or head->next is NULL:

Call DeleteAtHead(head)

Else:

Initialize temp = head

While temp->next->data is not equal to val:

Move temp to the next node

Set temp->next = temp->next->next

Function Display(head):

If head is NULL:

Return

Initialize temp = head

Repeat:

Print temp->data

Move temp to the next node

Until temp is equal to head

Print a newline

## Source code:

//CIRCULAR LINKED LIST

#include<iostream>

using namespace std;

struct node{

    int data;

    node\* next;

    node(int val){

        data = val;

        next = NULL;

    }

};

void insertattail(node\* &head, int val){

    node\* n = new node(val);

    if (head==NULL){                     // FUNCTION

        head=n;                          // TO INSERT

        n->next=n;                       // AT TAIL

        return;

    }

    node\* temp = head;

    while(temp->next!=head){

        temp=temp->next;

    }

    temp->next=n;

    n->next=head;

}

void insertathead(node\* &head,int val){

    node\* n = new node(val);

    if (head==NULL){                           // FUNCTION

        head=n;                                // TO INSERT

        n->next=n;                             //   HEAD

        return;

    }

    node\* temp = head;

    while(temp->next!=head){

        temp=temp->next;

    }

    temp->next=n;

    n->next=head;

    head=n;

}

void insertatmiddle(node\* &head,int val,int position){

    int count=1;

    node\* n = new node(val);

    if (head==NULL ){                          // FUNCTION

        head=n;                                // TO INSERT

        n->next=n;                             // AT MIDDLE

        return;

    }

    if (position == 1){

        insertathead(head,val);

        return;

    }

    node\* temp =head;

    while(temp->data!=val && count < position-1){

        temp=temp->next;

        count++;

    }

    n->next=temp->next;

    temp->next=n;

}

// DELETION

void deleteatend(node\* head ){

    if (head==NULL){

        return;

    }

    node\*temp = head;                  //   FUNCTION

    while(temp->next->next!=head){     //  DELETE FROM

        temp=temp->next;               //      END

    }

    temp->next=head;

}

void deleteathead(node\* &head){

    node\* temp= head;

    while(temp->next!=head){            //  FUNCTION

        temp=temp->next;                // DELETE FROM

    }                                   //    HEAD

    head=head->next;

    free(temp->next);

    temp->next=head;

}

void deleteatmiddle(node\* &head,int val){       //  FUNCTION

    if (head ==NULL || head->next==NULL){       // DELETE FROM

        deleteathead(head);                     //   MIDDLE

        return;

    }

    node\* temp = head;

    while(temp->next->data!=val){

        temp=temp->next;

    }

    temp->next=temp->next->next;

}

void display(node\* head){

    node\* temp =head;

    do {

        cout<<temp->data<<" ";

        temp=temp->next;

    }

    while (temp!=head);

    cout<<endl;

}

int main(){

    node\* head = NULL;

    insertattail(head,2);

    insertattail(head,3);

    insertattail(head,4);

    insertattail(head,5);

    //display(head);

    insertathead(head,1);

    //display(head);

   // insertatmiddle(head,9,3);

    //display(head);

    deleteatend(head);

    display(head);

    deleteathead(head);

    display(head);

    deleteatmiddle(head,3);

    display(head);

}

## Output:

**1 2 3 4**

**2 3 4**

**2 4**

## Learning from experiment

* Circular linked list insertion and deletion implemented.
* Proper handling of head and tail nodes.