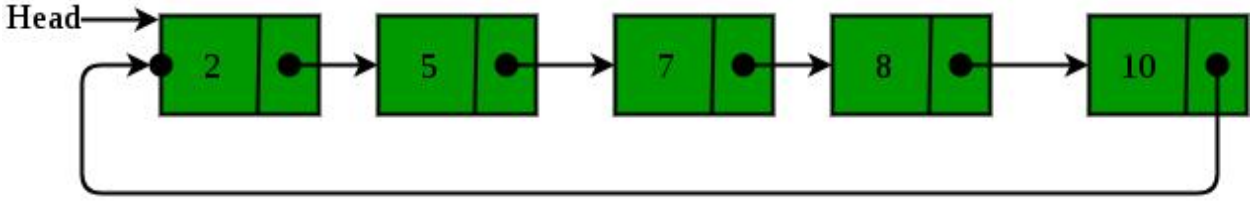
Experiment no 7:

Name: Kaushal Tare, Roll no:50, Batch: C , SEM: III

Aim: Implementation of Circular Linked List ADT

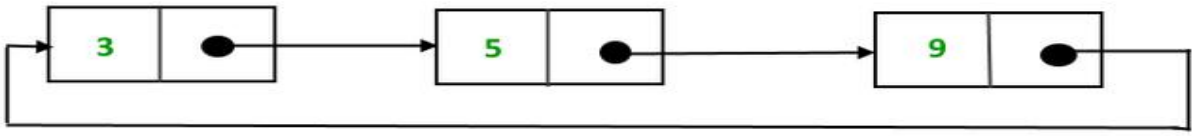
Objective: Circular Linked List can be used to manage the Computing Resources of the computer. Data Structure such as stacks and queue are implemented with the help of circular linked list

Theory:
The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.



There are generally two types of circular linked lists:

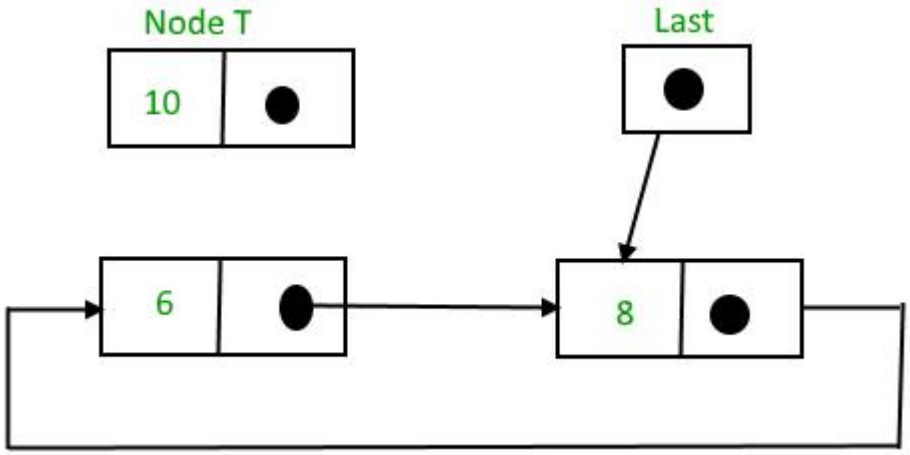
- Circular singly linked list: In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We traverse the circular singly linked list until we reach the same node where we started. The circular singly linked list has no beginning or end. No null value is present in the next part of any of the nodes.



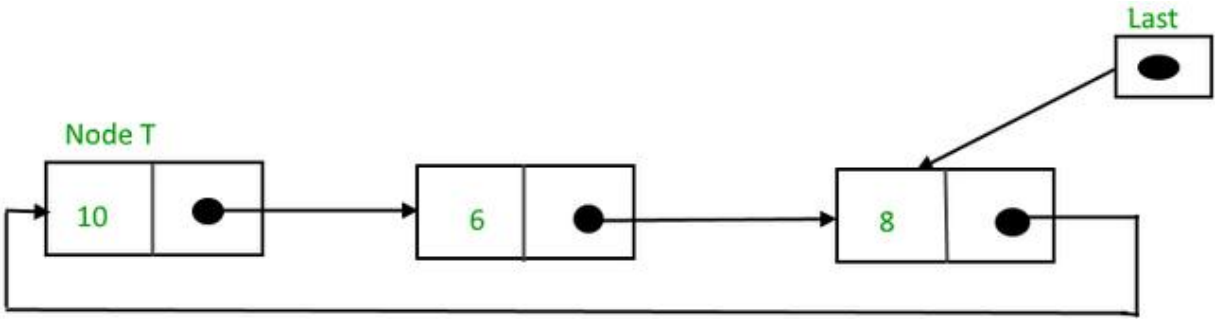
Algorithm:
1) Insertion at the beginning of the list: To insert a node at the beginning of the list, follow these steps:

- Create a node, say T.
- Make T -> next = last -> next.

- last -> next = T.



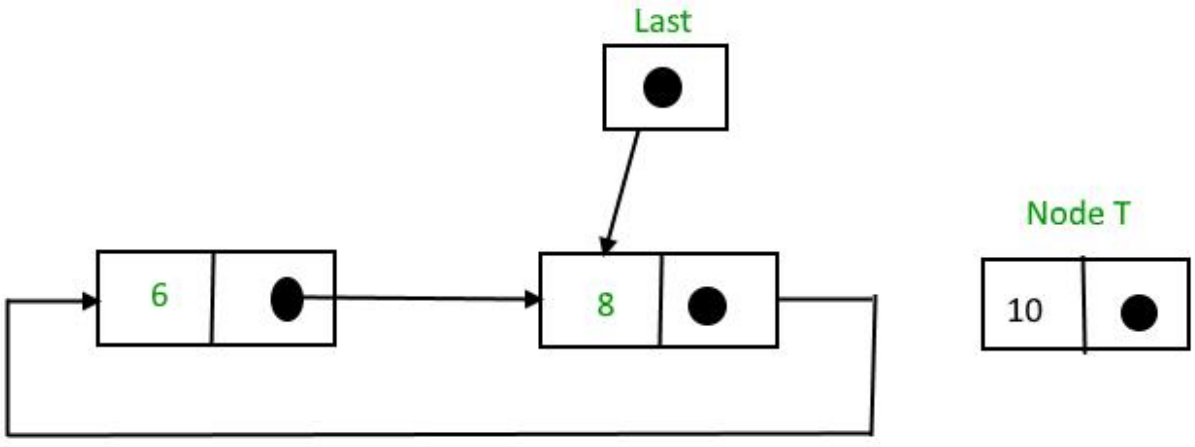
And then,



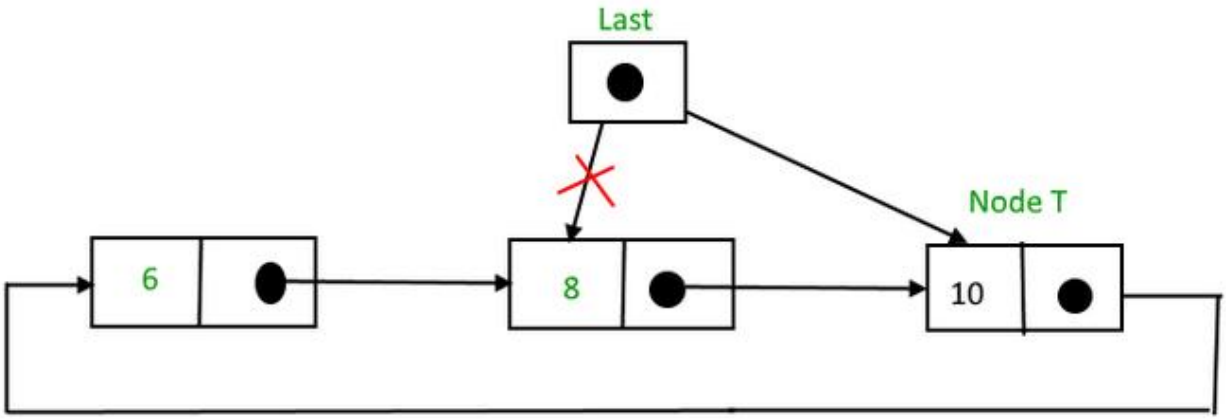
2) Insertion at the end of the list: To insert a node at the end of the list, follow these steps:

- Create a node, say T.
- Make T -> next = last -> next;
- last -> next = T.
- last = T.

Before insertion,



After insertion,



Circular linked list after insertion of node at the end

3) Insertion in between the nodes: To insert a node in between the two nodes, follow these steps:

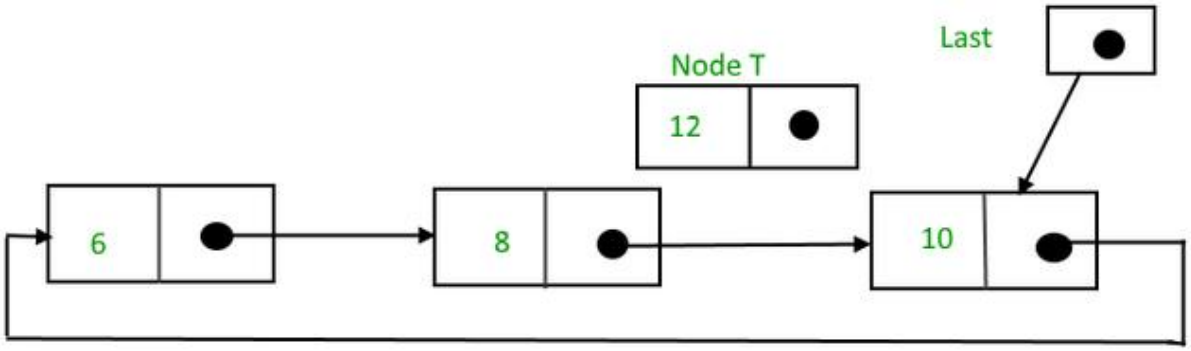
Create a node, say T.

Search for the node after which T needs to be inserted, say that node is P.

Make T -> next = P -> next;

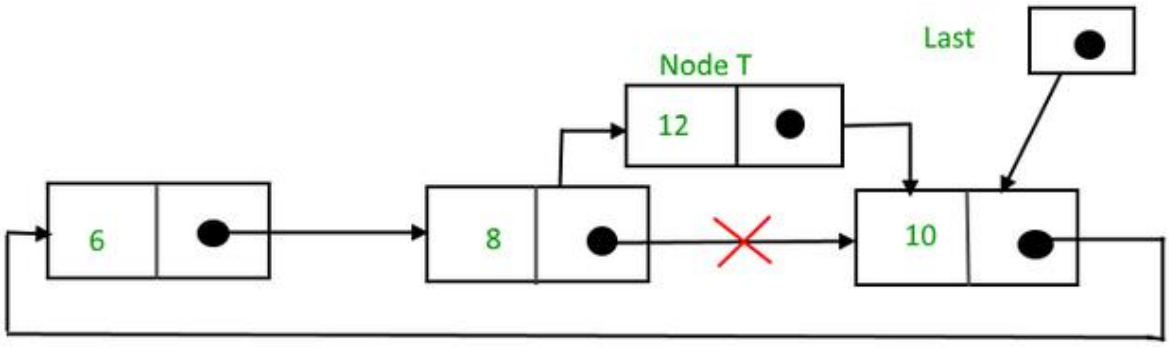
P -> next = T.

Suppose 12 needs to be inserted after the node has the value 10,



Circular linked list before insertion

After searching and insertion,



Circular linked list after insertion

2. Deletion in a circular linked list:

1) Delete the node only if it is the only node in the circular linked list:

Free the node's memory

The last value should be NULL A node always points to another node, so NULL assignment is not necessary.
Any node can be set as the starting point.
Nodes are traversed quickly from the first to the last.

2) Deletion of the last node:

Locate the node before the last node (let it be temp)

Keep the address of the node next to the last node in temp

Delete the last memory

Put temp at the end

3) Delete any node from the circular linked list: We will be given a node and our task is to delete that node from the circular linked list.

Algorithm:

Case 1: List is empty.

If the list is empty we will simply return.

Case 2:List is not empty

If the list is not empty then we define two pointers curr and prev and initialize the pointer curr with the head node.

Traverse the list using curr to find the node to be deleted and before moving to curr to the next node, every time set prev = curr.

If the node is found, check if it is the only node in the list. If yes, set head = NULL and free(curr).

If the list has more than one node, check if it is the first node of the list. Condition to check this(curr == head). If yes, then move prev until it reaches the last node. After prev reaches the last node, set head = head -> next and prev -> next = head. Delete curr.

If curr is not the first node, we check if it is the last node in the list. Condition to check this is (curr -> next == head).

If curr is the last node. Set prev -> next = head and delete the node curr by free(curr).

If the node to be deleted is neither the first node nor the last node, then set prev -> next = curr -> next and delete curr.

If the node is not present in the list return head and don't do anything.

Code :

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

// Structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to insert a node at the
// beginning of a Circular linked list
void push(struct Node** head_ref, int data)
{

    // Create a new node and make head
    // as next of it.
    struct Node* ptr1 = (struct Node*)malloc(sizeof(struct Node));
    ptr1->data = data;
    ptr1->next = *head_ref;

    // If linked list is not NULL then
    // set the next of last node
    if (*head_ref != NULL) {

        // Find the node before head and
        // update next of it.
        struct Node* temp = *head_ref;
```

```
        while (temp->next != *head_ref)
            temp = temp->next;
        temp->next = ptr1;
    }
    else

        // For the first node
        ptr1->next = ptr1;

    *head_ref = ptr1;
}

// Function to print nodes in a given
// circular linked list
void printList(struct Node* head)
{
    struct Node* temp = head;
    if (head != NULL) {
        do {
            printf("%d ", temp->data);
            temp = temp->next;
        } while (temp != head);
    }

    printf("\n");
}

// Function to delete a given node
// from the list
void deleteNode(struct Node** head, int key)
{

    // If linked list is empty
    if (*head == NULL)
        return;

    // If the list contains only a
    // single node
    if ((*head)->data == key && (*head)->next == *head) {
        free(*head);
        *head = NULL;
        return;
    }
```

```
    struct Node *last = *head, *d;

    // If head is to be deleted
    if ((*head)->data == key) {

        // Find the last node of the list
        while (last->next != *head)
            last = last->next;

        // Point last node to the next of
        // head i.e. the second node
        // of the list
        last->next = (*head)->next;
        free(*head);
        *head = last->next;
        return;
    }

    // Either the node to be deleted is
    // not found or the end of list
    // is not reached
    while (last->next != *head && last->next->data != key) {
        last = last->next;
    }

    // If node to be deleted was found
    if (last->next->data == key) {
        d = last->next;
        last->next = d->next;
        free(d);
    }
    else
        printf("Given node is not found in the list!!!\n");
}

// Driver code
int main()
{
    // Initialize lists as empty
    struct Node* head = NULL;

    // Created linked list will be
```

```
    // 2->5->7->8->10
    push(&head, 2);
    push(&head, 5);
    push(&head, 7);
    push(&head, 8);
    push(&head, 10);

    printf("List Before Deletion: ");
    printList(head);

    deleteNode(&head, 7);

    printf("List After Deletion: ");
    printList(head);

    return 0;
}
```

Output:

```
List Before Deletion: 10 8 7 5 2
List After Deletion: 10 8 5 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion : In a circular linked list experiment, we observed efficient traversal and insertion due to the circular structure. However, careful management of pointers and handling edge cases is crucial to prevent infinite loops and ensure proper functioning.