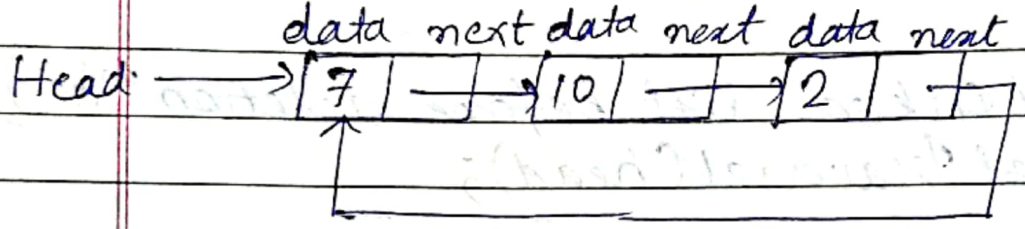
Circular Linked List

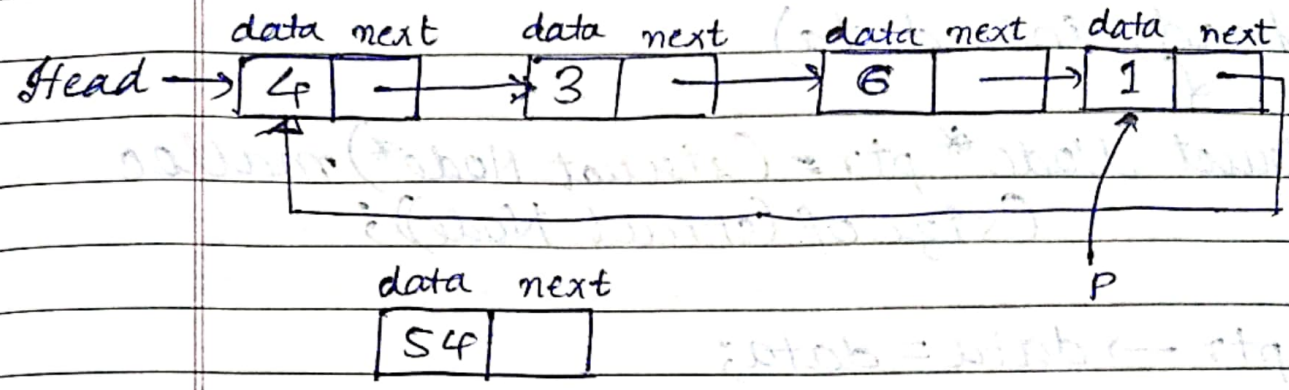
A circular linked list where the last element points to the first (head) hence forming a circular chain. There is no node pointing to NULL, indicating the absence of any end node. In circular linked lists, we have a head pointer but not starting of the list.



Operations:
Operations can be exactly performed like singly linked list. It's just that we have to maintain an extra pointer to check if we have gone through the linked list once.

Traversal:
→ It can be achieved by creating a new struct Node * pointer p, which starts from the head & goes through the list until it points again at the head
→ ~~Here~~ There is a head but no starting to this circular linked list, we have this head pointer just to start or except in this list & for our convenience.

Insertion in a Circular Linked List



E.g. Code

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

struct Node {

    int data;
    struct Node* next;

};

void LinkedList Traversal (struct Node* head)
{
struct Node* ptr = head;
do{
  printf("Element is %d \n", ptr -> data);
      ptr = ptr -> next;
  } while (ptr != head);

}
```

```
struct Node* insertAtFirst(struct Node*
  head, int data)
    {
struct Node* ptr = (struat Node*) malloc
          (sizeof(struct Node));

  ptr -> data = data;

  struct Node* p = head -> next;
  while (p -> next != head){
        p = p -> next;
    }
// at the point p points to the last
// node of the circular linkedList

    p -> next = ptr;
    ptr -> next = head;
    head = ptr;
    return head;

}

int main(){

struct Node* head;
--u ------* second;
-- 11 ------* third;
--11 ------* fourth;
```

```
head = (struct Node*)malloc(sizeof(struct Node));
second = --"--
third = --"--
fourth = --"--

head -> data = 4;
head -> next = second;

second -> data = 3;
second -> next = third;

third -> data = 6;
third -> next = fourth;

fourth -> data = 1;
fourth -> next = head;

pf("\nCircular Linked List before insertion\n");
LinkedListTraversal(head);
head = insertatFirst(head, 54);
--"-- (--"--, 58);
--"-- (--"--, 59);

pf("\nCircular Linked List after insertion\n");
LinkedListTraversal(head);

return 0;

}
```

O/P:

Circular Linked List before insertion

Element is 4

3 —— 11 ——

6 —— 11 ——

1 —— 11 ——

Circular Linked List after insertion

Element is 59

59 —— 11 ——

54 —— 11 ——

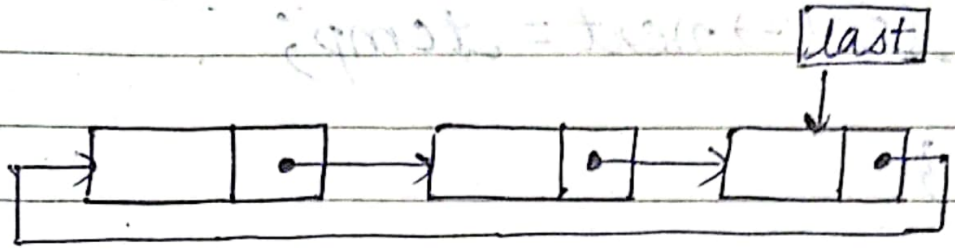
4 —— 11 ——

3 —— 11 ——

6 —— 11 ——

1 —— 11 ——

Circular LL operations:



Insert at Beginning:

```
struct node{
    int info;
    struct node* next;
        };

struct node* last=NULL;

void InsertatFront(int data){

    struct node* temp=(struct node*)
        malloc(sizeof(struct node));

    if(last==NULL){
        temp->info=data;
        temp->next=temp;
        last=temp;
            }
```

Else last node contains the ref. of the new node & new node contains the ref. of the previous first node

```
        else {
            temp -> info = data;
temp -> next = last -> next;
            last -> next = temp;

        }
    }
```

Insertion at end:

```
void add at last (int data)
{
struct node * temp = (struct node *)
        malloc (sizeof (struct node));

        if (last == NULL) {
            temp -> info = data;
            temp -> next = temp;
            last = temp;
        }

        else{
            temp -> info = data;
            temp -> next = last -> next;
            last -> next = temp;
            last = temp;
        }
    }
```

```
void insertafter()
{

int data, value;

struct node *temp, *n;

pf("\n Enter the no. after which
    you want to enter: \n");
scanf("%d", &value);

temp = last->next;

do{
    if (temp->info == value){

  n = (struct node*)malloc(sizeof(struct
                          node));

pf("\n Enter data to be inserted:\n");
scanf("%d", &data);

n->info = data;
n->next = temp->next;
temp->next = n;

        if(temp == last){
            last=n;
        break;

}
```

```
else {
    temp = temp -> next;
}
} while (temp != last -> next);
}
```

Function to delete a particular element

```
void deleteNode(struct node *head,
                int key)
{
    if (head == NULL)
        return;
struct node *curr = head, *prev;
while (curr -> data != key)
{
    if (curr -> next == head)
    {
        pf("Given node not found-\n");
        break;
    }
    prev = curr;
    curr = curr -> next;
}
```

```
// Check if node is only node
if(curr->next==head)
{
    head == NULL;
    free(curr);
    return;
}
// If more than one node, check if it's first node
if(curr==head)
{
    prev = head;
    while (prev->next != head){
        prev = prev->next;
    }
    head = curr->next;
    prev->next = head;
    free (curr);
}
// Check if last node
else if(curr->next==head &&
                curr==head)
{
    prev->next = head;
    free(curr);
}
else {
    prev->next = curr->next;
    free (curr);
}
}
```