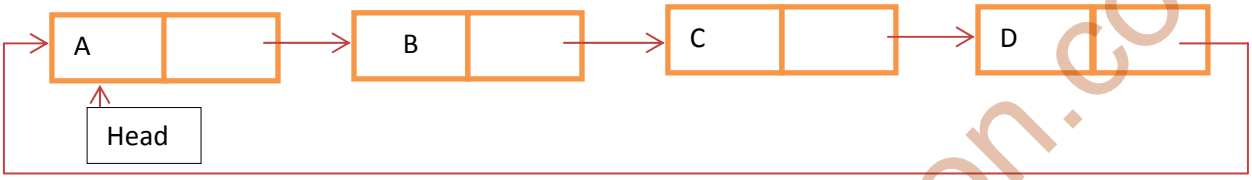
Circular Linked List

A circular linked list is a linked list in which tail element next pointer points to head element. While traversing the circular linked list we should be careful because circular linked lists don't have end where as in singly linked lists and doubly linked lists the end of lists are indicated with null value. Unlike linked lists, the circular linked lists can be used to add data items at any point without starting from the beginning point.



The type declaration for a circular linked list is as follows:

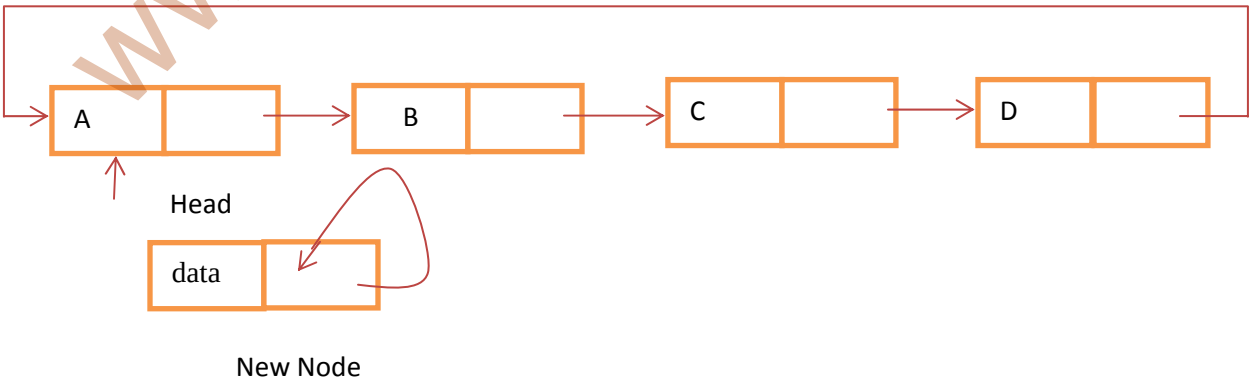
```
struct CLL{
        int data;
        struct CLL *next;
};
```

In the circular linked lists, we access the elements using the head node (similar to singular linked lists.).

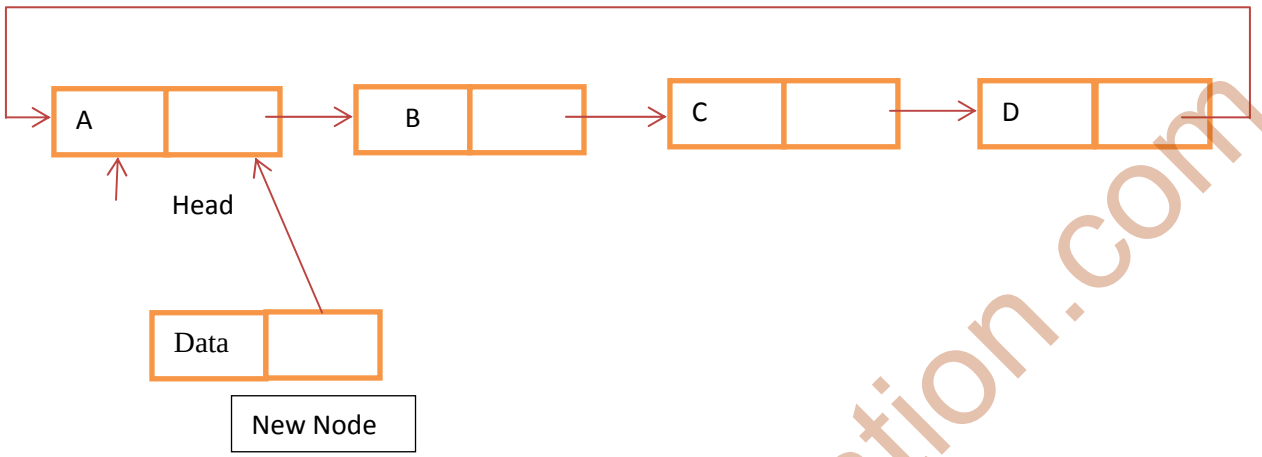
CLL Operations

1. Inserting a New Node at Front of a CLL: After inserting a new node at front of the list, we just need to update the pointers.

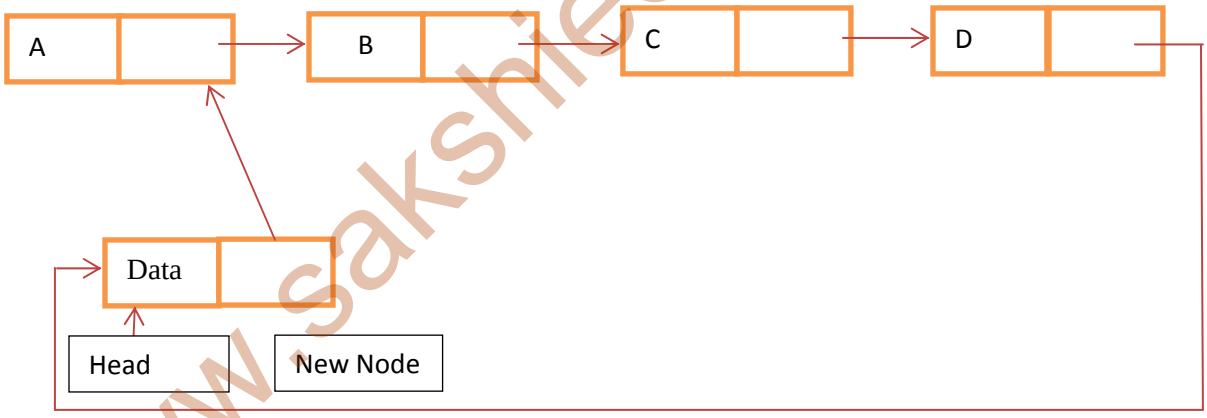
- Create a new node initially its next pointer points to itself.



- Update the next pointer of new node with head node. Traverse the list till reach tail node (This will stop at which the node points to previous node in the list).



- Update the tail node that points to the previous node to new node and also make the new node as head.

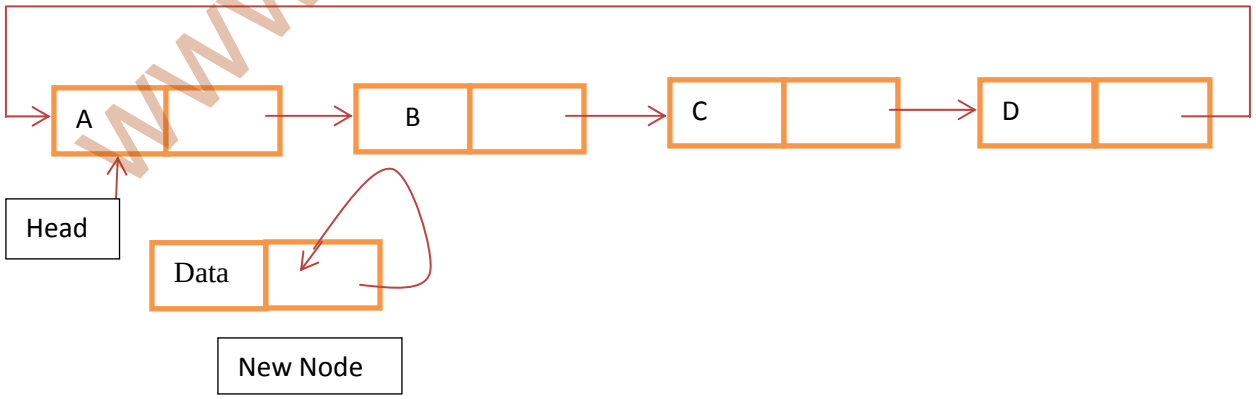



```
Void insertatFront(structCLLNode* head){
    StructCLLNode *temp;
    StructCLLNode *newNode=(structCLLNode*)malloc(sizeof(structCLLNode*));
    if(!newNode){
        printf("Memory Error");
        return;
```

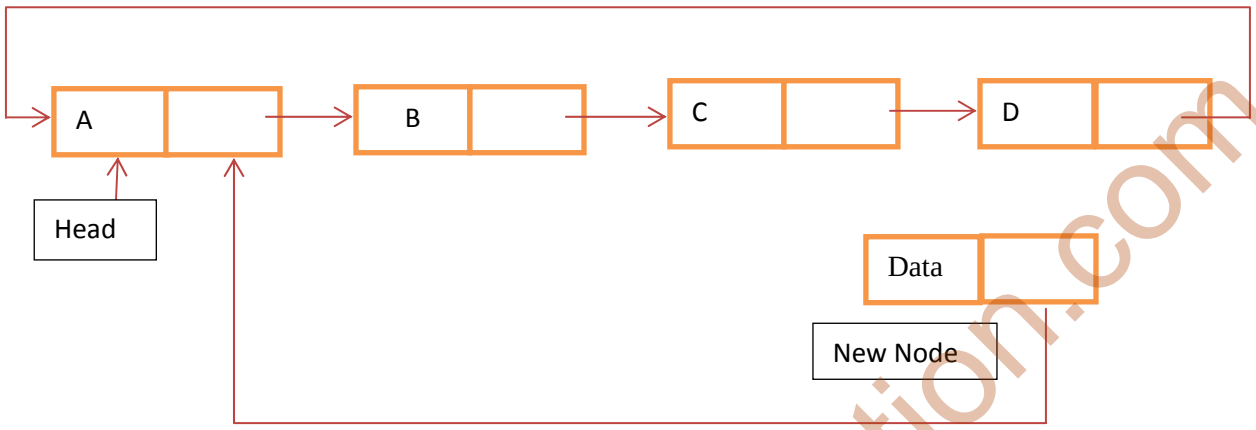
```
        }
        printf("\nEnter data to insert into list:");
        scanf("%d",&newNode->data);
        newNode->next=newNode;
        temp=head;
        if(head==NULL)
                head=newNode;
        else{
                while(temp->next!=head){
                        temp=temp->next;
                }
                newNode->next=head;
                current->next=newNode;
                head=newNode;
        }
}
```

2. Inserting a New Node at end of a CLL: Inserting a new node at the front of the list and the end of list is same, the only difference is we just need to update the pointers.

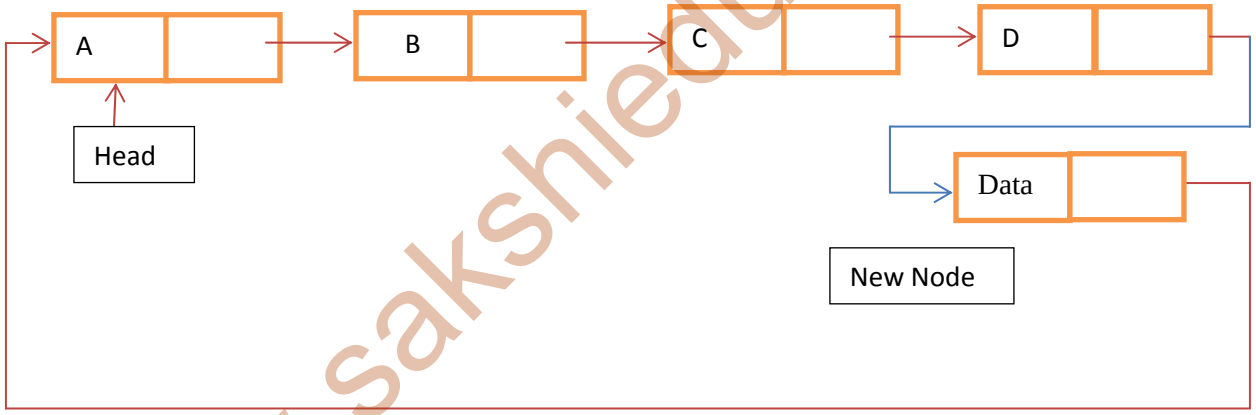
- Create a new node initially its next pointer points to itself.



- Update the next pointer of new node with head node. Traverse the list till reach tail node (This will stop at which, the node points to previous node in the list).



- Update the next pointer of previous node and make it point the new node.

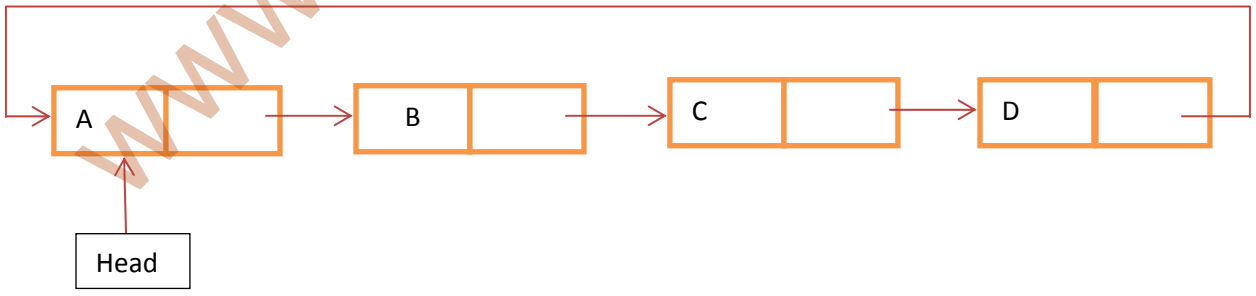



```
Void insertatEnd(structCLLNode* head){
    StructCLLNode *temp;
    StructCLLNode *newNode=(structCLLNode*)malloc(sizeof(structCLLNode*));
    if(!newNode){
        printf("Memory Error");
        return;
    }
```

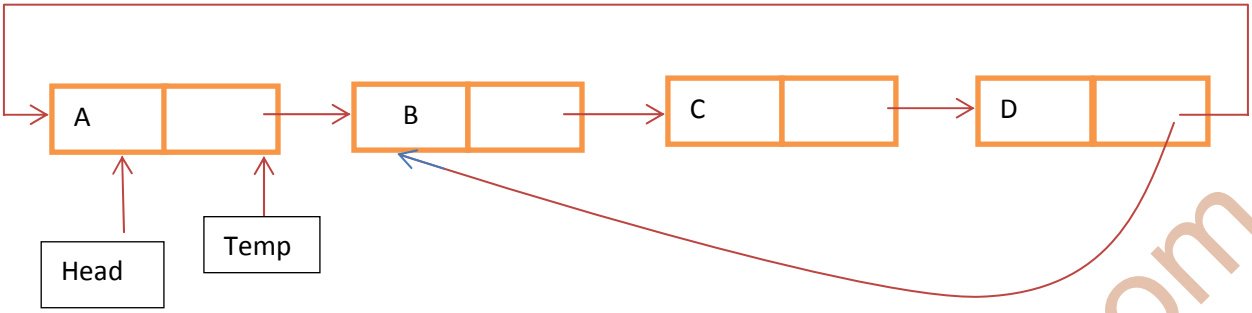
```
        printf("\nEnter data to insert into list:");
        scanf("%d",&newNode->data);
        newNode->next=newNode;
        temp=head;
        if(head==NULL)
                head=newNode;
        else{
                while(temp->next!=head){
                        temp=temp->next;
                }
                newNode->next=head;
                current->next=newNode;
        }
}
```

3. Deleting the first node in the list: The first node can be deleted by simply replacing the next field of the tail node with the next field of first or head node. Then copy the head node into temporary node and update the head node to point to the next filed of head and release the memory the temporary node.

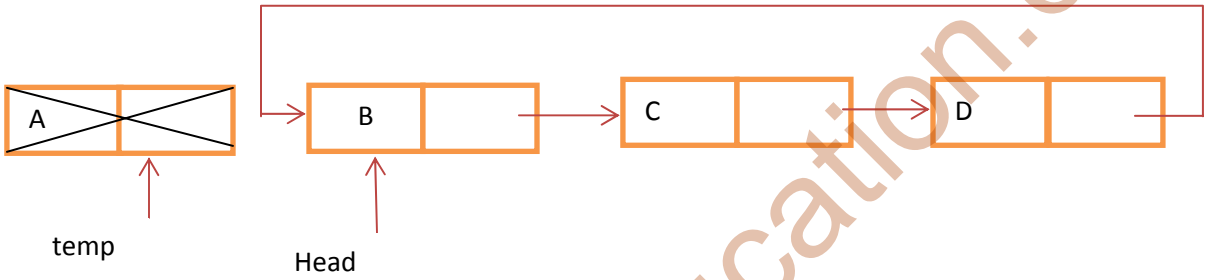
- Traverse the list until the tail node points to previous node (The node that has to be deleted)



- Create the temp node and copy the node to be deleted (head) into the temp and update the next pointer of tail with the next pointer of the head.



- Update the head and free the temp.

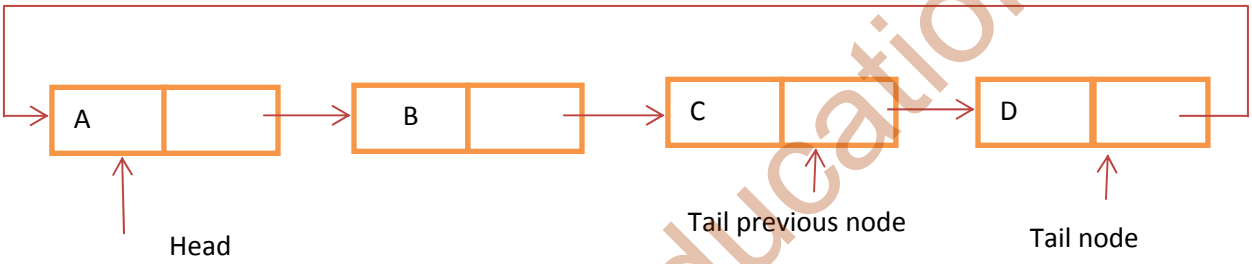



```
Void DeleteatFront(structCLLNode* head){
        StructCLLNode *temp;
        StructCLLNode *p;
        p=temp=head;
        if(head==NULL)
                printf("List if empty\n");
        else{
                while(p->next!=head){
                        p=p->next;
                }
                p->next=head->next;
                head=head->next;
                free(temp);
```

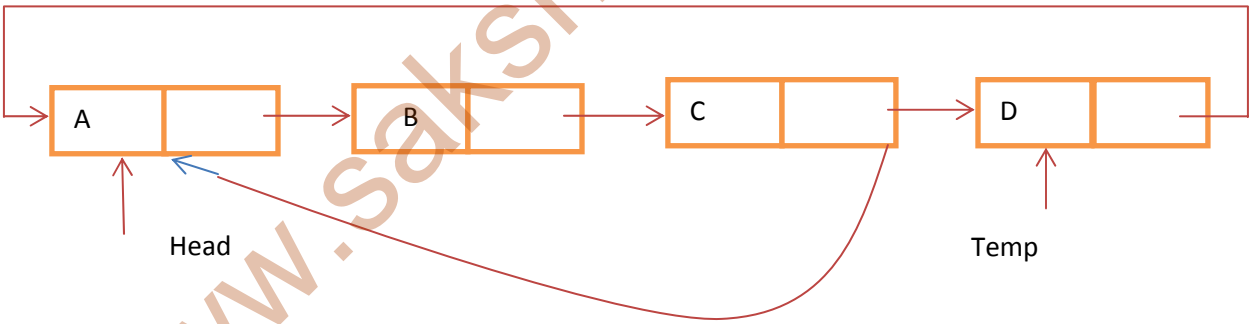
```
            temp=NULL;
        return;
        }
}
```

4. Deleting the last node in the list: The last node can be deleted by simply replacing the next field of the tail previous node with the head node. Copy the tail node into the temporary node and release the memory the temporary node.

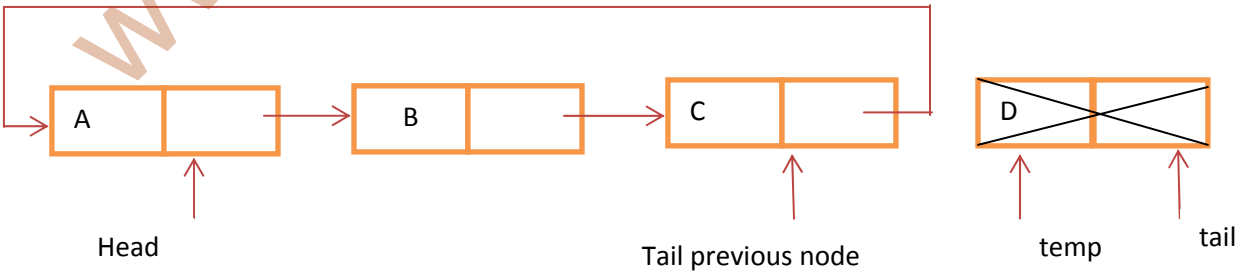
- Traverse the list until find tail node and tail previous node



- Create a temp node and copy the tail node into temp and update the next pointer of tail previous node that points to head with the temp node pointer.

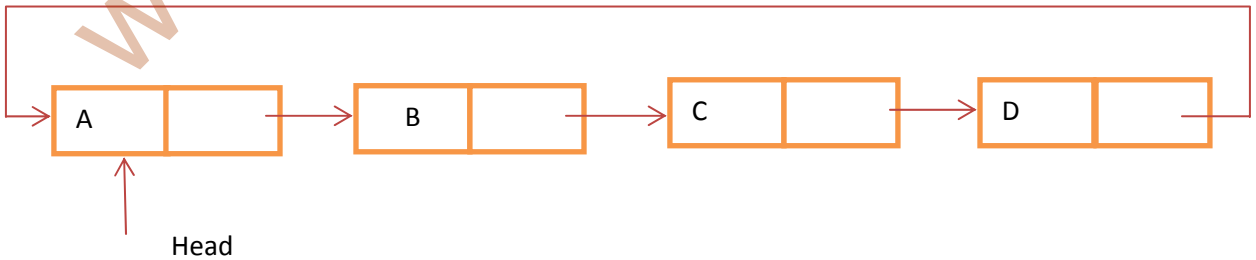


- Delete the temp node.



```
Void DeleteatEnd(structCLLNode* head){
        StructCLLNode *temp;
        StructCLLNode *p;
        temp=head;
        if(head==NULL)
                printf("List if empty\n");
        else{
                while(temp->next!=head){
                        p=temp;
                        temp=temp->next;
                 }
                p->next=head;
                free(temp);
                temp=NULL;
        return;
        }
}
```

5. Display Elements in List: Let us assume that the list is accessed by its head node. Since all nodes are arranged in circular fashion, the tail node next pointer points to the head node. To display the elements in the list to traverse the list until reach the head node.



```
Void displatList(structCLLNode* head){
        structCLLNode* p;
        p=head;
        if(head==NULL){
                printf("List isEmpty\n");
        }
        while(p!=head){
                printf("->%d",p->data);
                p=p->next;
        }
}
```