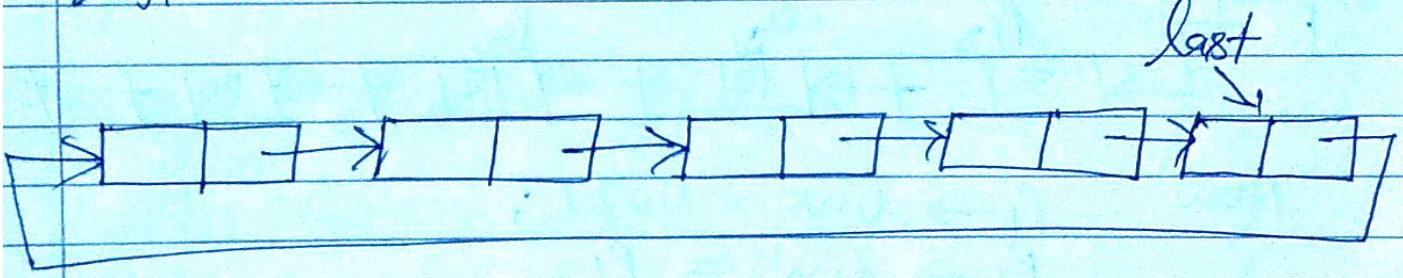
Circular linked list

Circular linked list is implemented to overcome the problem face in singly linked list. when traversing a singly linked list we traverse in forword way we can not access the privious node.
Suppose we are at the last node of singly linked list and we want to acces the second node of linked list then again we start traversing from the first node.

Circular linked list hey only small change. In circular linked list link part of node point to the first node of linked list.



Traversing in Circular linked list

Traversal in list hey a need of pointer variable which points to first node of list. Here we maintain the pointer last which points to last node. But link part of this last pointer points to first node of list.

So-

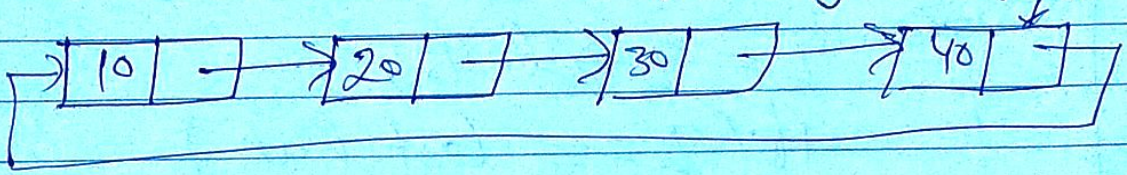
```
struct node *q;
q -> last -> link;
while(q != last)
{
    q = q -> link;
}
```

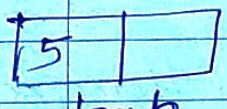
Insertion into a circular linked list -->

Insertion in a circular linked list may be possible in two ways -

(1) Insertion at beginning.
(2) Insertion in between.

Case 1. Insertion at beginning. >

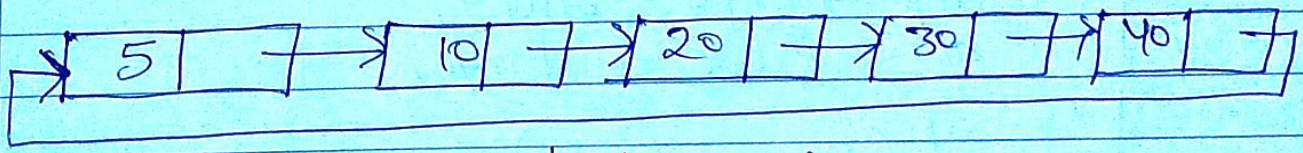




we want to add tmp at first position.

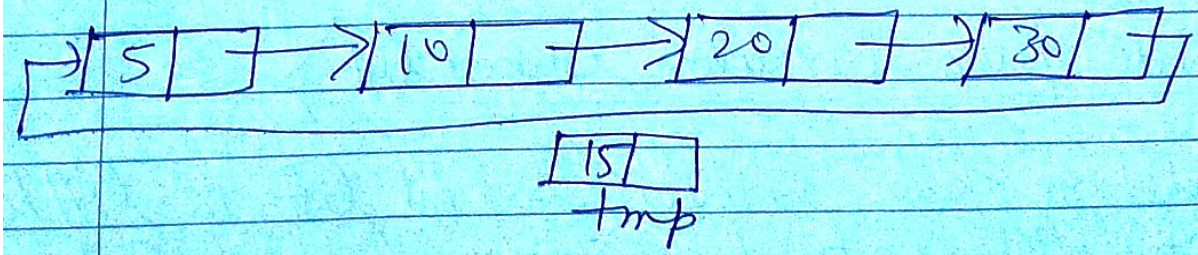
```
struct node *tmp;
tmp = (void*) malloc (sizeof struct node);
tmp -> info = data
tmp -> link = last -> link;
last -> link = tmp;
```

After Insertion



Case 2. Insertion in between. >

Insertion in between is same as singly linked list



For insert tmp at position 3 first we traverse list with the help of q pointer and stop traversing when (position-1) node reached.

Complete function –

```
void add after (int data, int pos)
{
    struct node *tmp, *q;
    int i;
    q = last -> link;
    for(i=0; i< pos-1; i++)
    {
        q = q -> link;
        if(q == last link)
        {
            printf("There are less than pos elements \n", pos);
            return;
        }
    }
    tmp = (void *) malloc(sizeof(struct node));
    tmp -> link = q -> link;
    tmp -> info = data;
    q -> link = tmp;
    if (q == last)
        last = tmp;
}
```

Delete from Circular linked list

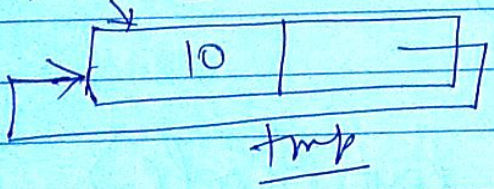
Deletion from circular linked list is little bit different from singly linked list. Here we have a need to take care of some more conditions because of it's circular behaviour.

There are four cases for deletion :-

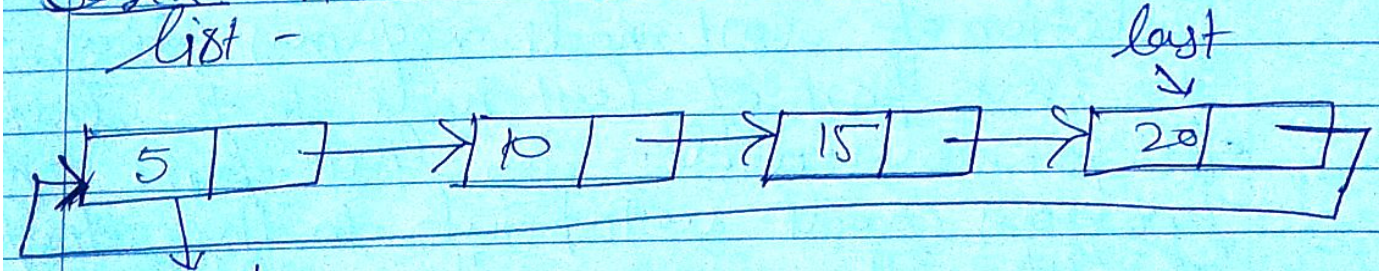
(1) If list has only one element.
(2) Node to be deleted is the first node of list.
(3) Deletion in between.
(4) Node to be deleted is last node of list.

Case 1 :- If list has only one node.
Here we check the condition for only one element of list then assign NULL value to last pointer because after deletion no node will be in list.

```
struct node *tmp;
if(last->link==last && last->info==num)
{
  tmp = last;
  last = NULL;
  free(tmp);
}
```

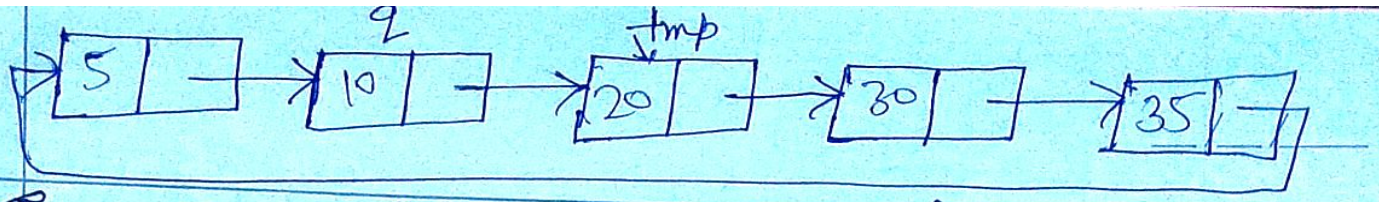


Case 2 - Node to be deleted is the first node of list -



We want to delete first node i.e 


```
struct node *tmp, *q;
q = last->link;
if(q->info==num)
{
  tmp = q;
  last->link = q->link;
  free(tmp);
}
```



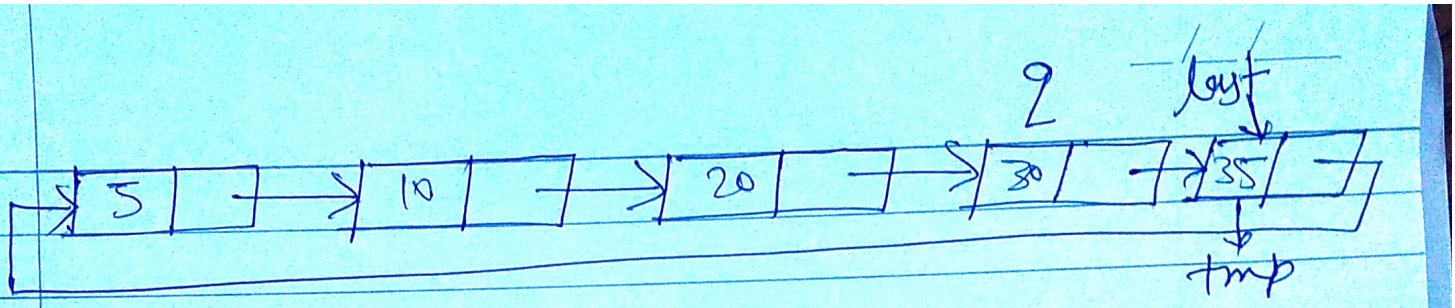
Case 3 – Deletion in between :

For deletion in between first we traverse the ~~node~~ list ~~& then delete the node~~ and search the deleted node then delete the node. Pointer q points to the previous node of the deleted node.

```
struct node *tmp, *q;
q = last -> link;
while(q -> link != last)
{
    if(q -> link -> info == ~~item~~ data)
    {
        tmp = q -> link;
        q -> link = tmp -> link;
        free(tmp);
    }
    q = q -> link;
}
```

Case 4. Deletion of last node :

Deletion of last node requires to assign the link part of last node to the link part of previous node. So link part of previous node will point to the first node of list. Then assign the address of previous node to the pointer variable last because after deletion of last node pointer variable last should point to the previous node.




```
struct node *q, *tmp;
while (q->link != last)
    q = q->link
tmp = q->link;
q->link = last->link;
last = q;
```

```
/* Program of circular linked list*/
# include <stdio.h>
# include <malloc.h>

struct node
{
        int info;
        struct node *link;
}*last;

main( )
{
        int choice,n,m,po,i;
        last=NULL;
        while(1)
        {
                printf("1.Create List\n");
                printf("2.Add at begining\n");
                printf("3.Add after \n");
                printf("4.Delete\n");
                printf("5.Display\n");
                printf("6.Quit\n");
                printf("Enter your choice : ");
                scanf("%d",&choice);

                switch(choice)
                {
                case 1:
                        printf("How many nodes you want : ");
                        scanf("%d",&n);
                        for(i=0; i < n;i++)
                        {
                                printf("Enter the element : ");
                                scanf("%d",&m);
                                create_list(m);
                        }
                        break;
                case 2:
                        printf("Enter the element : ");
```

```
                scanf("%d",&m);
                addatbeg(m);
                break;
        case 3:
                printf("Enter the element : ");
                scanf("%d",&m);
                printf("Enter the position after which this element is inserted : ");
                scanf("%d",&po);
                addafter(m,po);
                break;
        case 4:
                if(last == NULL)
                {
                        printf("List underflow\n");
                        continue;
                }
                printf("Enter the number for deletion : ");
                scanf("%d",&m);
                del(m);
                break;
        case 5:
                display( );
                break;
        case 6:
                exit( );
        default:
                printf("Wrong choice\n");
        }/*End of switch*/
    }/*End of while*/
}/*End of main( )*/

create_list(int num)
{
        struct node *q,*tmp;
        tmp= malloc(sizeof(struct node));
        tmp->info = num;

        if(last == NULL)
        {
                last = tmp;
                tmp->link = last;
        }
        else
        {
                tmp->link = last->link; /*added at the end of list*/
                last->link = tmp;
                last = tmp;
        }
}/*End of create_list( )*/
```

```
addatbeg(int num)
{
        struct node *tmp;
        tmp = malloc(sizeof(struct node));
        tmp->info = num;
        tmp->link = last->link;
        last->link = tmp;
}/*End of addatbeg( )*/

addafter(int num,int pos)
{

        struct node *tmp,*q;
        int i;
        q = last->link;
        for(i=0; i < pos-1; i++)
        {
                q = q->link;
                if(q == last->link)
                {
                        printf("There are less than %d elements\n",pos);
                        return;
                }
        }/*End of for*/
        tmp = malloc(sizeof(struct node) );
        tmp->link = q->link;
        tmp->info = num;
        q->link = tmp;
        if(q==last)    /*Element inserted at the end*/
                last=tmp;
}/*End of addafter( )*/

del(int num)
{
        struct node *tmp,*q;
        if( last->link == last && last->info == num)  /*Only one element*/
        {
                tmp = last;
                last = NULL;
                free(tmp);
                return;
        }
        q = last->link;
        if(q->info == num)
        {
                tmp = q;
                last->link = q->link;
                free(tmp);
```

```
                return;
        }
        while(q->link != last)
        {
                if(q->link->info == num)    /*Element deleted in between*/
                {
                        tmp = q->link;
                        q->link = tmp->link;
                        free(tmp);
                        printf("%d deleted\n",num);
                        return;
                }
                q = q->link;
        }/*End of while*/
        if(q->link->info == num)    /*Last element deleted q->link=last*/
        {
                tmp = q->link;
                q->link = last->link;
                free(tmp);
                last = q;
                return;
        }
        printf("Element %d not found\n",num);
}/*End of del( )*/

display( )
{
        struct node *q;
        if(last == NULL)
        {
                printf("List is empty\n");
                return;
        }
        q = last->link;
        printf("List is :\n");
        while(q != last)
        {
                printf("%d ", q->info);
                q = q->link;
        }
        printf("%d\n",last->info);
}/*End of display( )*/
```