Navlakhi®

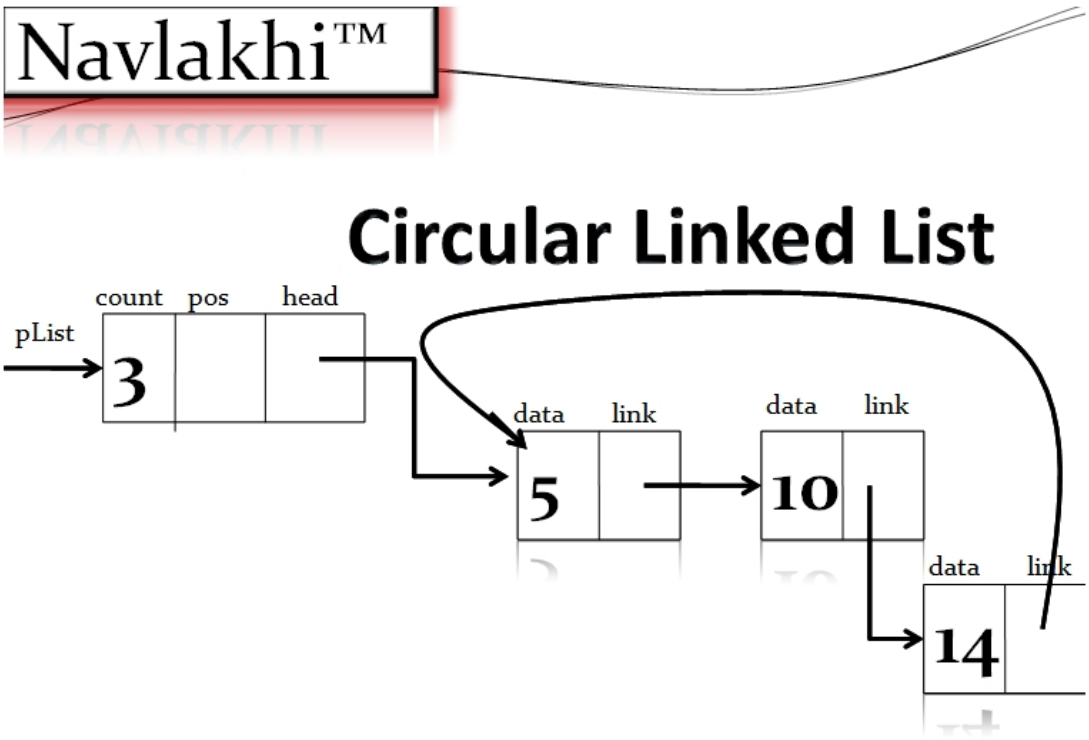
Circular Linked List

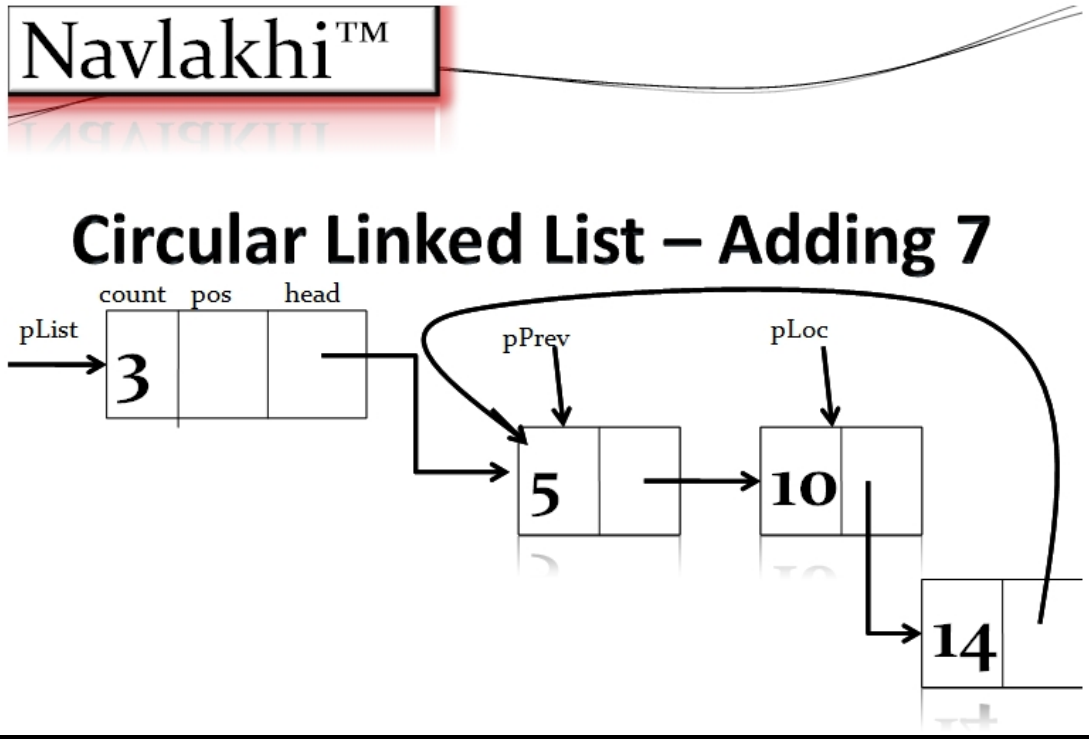
Methodology and Program

By Abhishek Navlakhi

Semester 3: Data Structures

ADDITION OF DATA

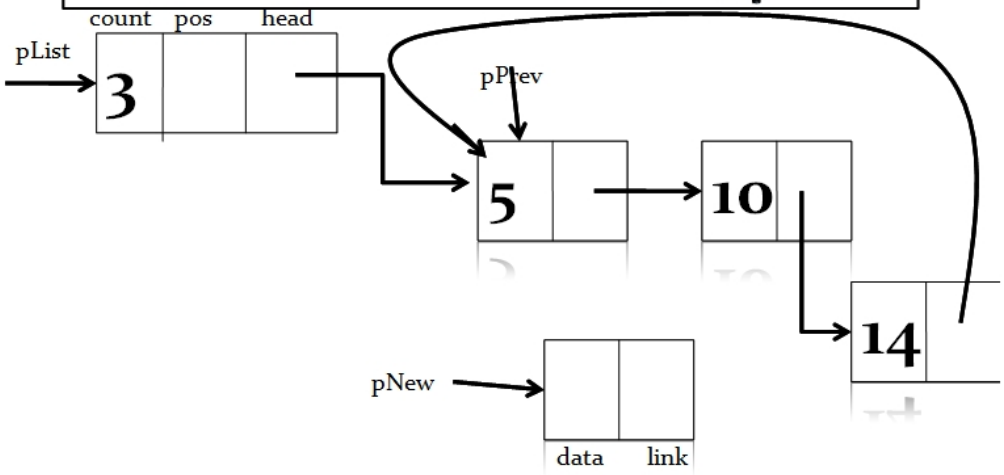




Navlakhi™

Circular Linked List – Adding 7

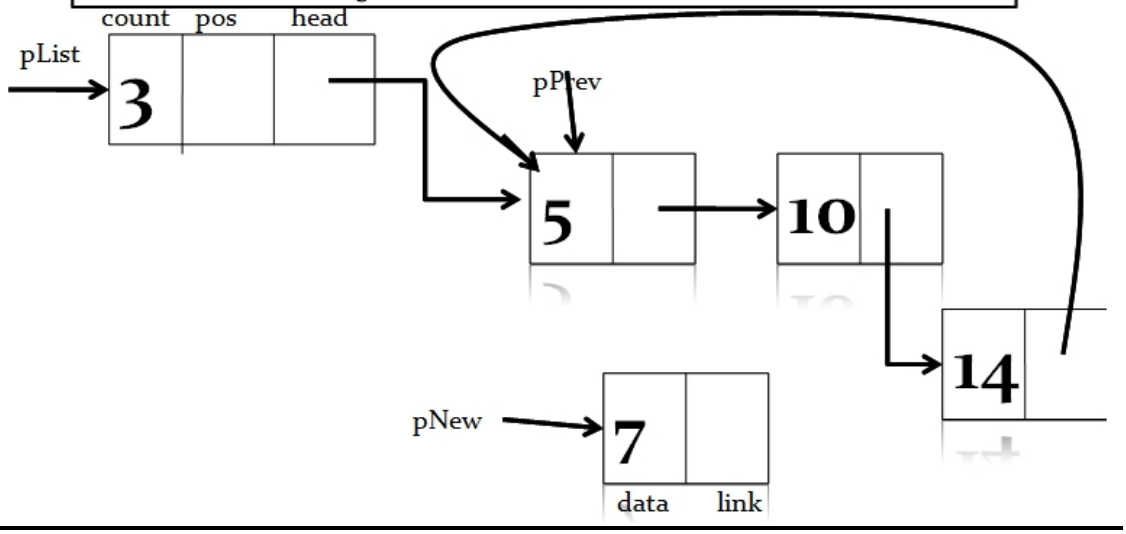
malloc pNew



Navlakhi™

Circular Linked List – Adding 7

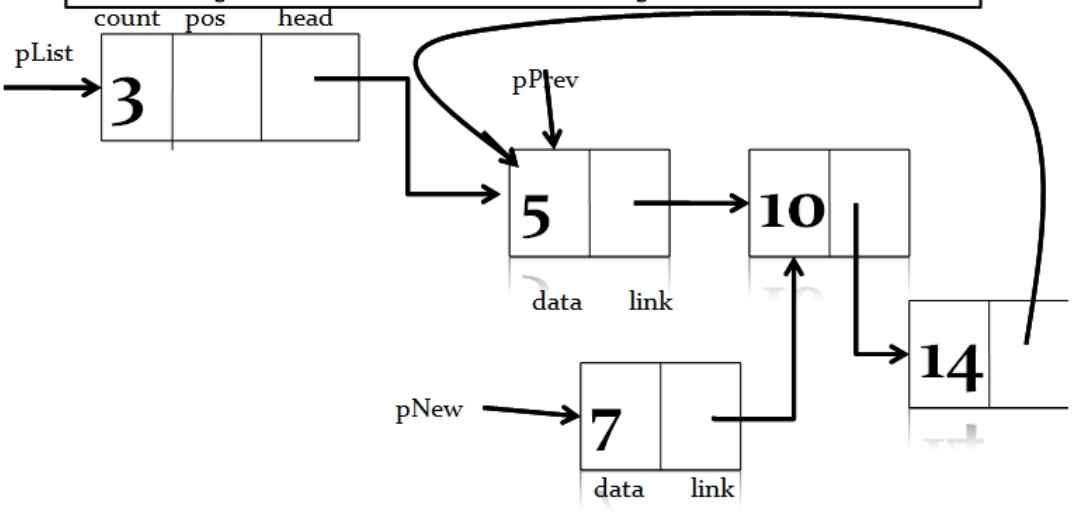
pNew->data = dataIn



Navlakhi™

Circular Linked List – Adding 7

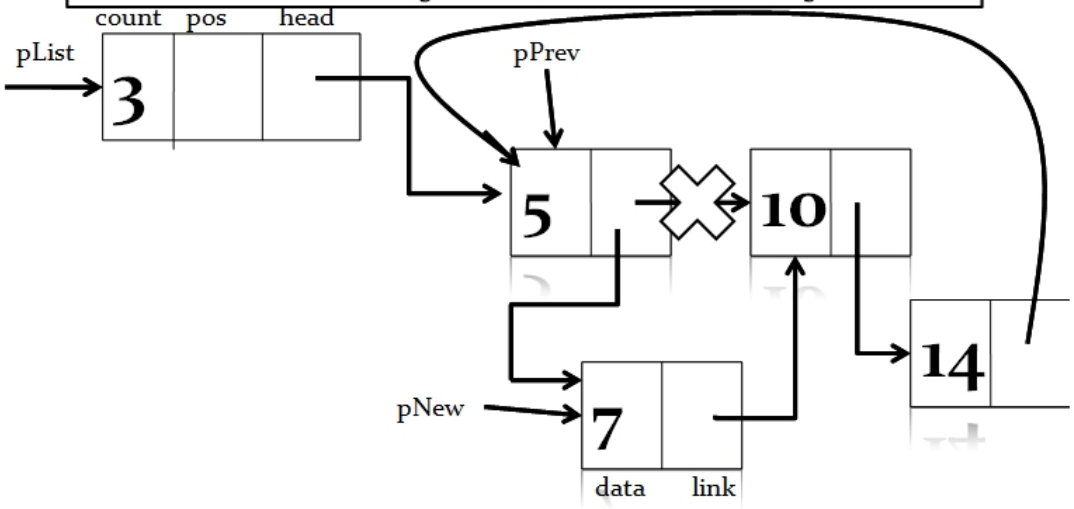
pNew->link = pPrev->link



Navlakhi™

Circular Linked List – Adding 7

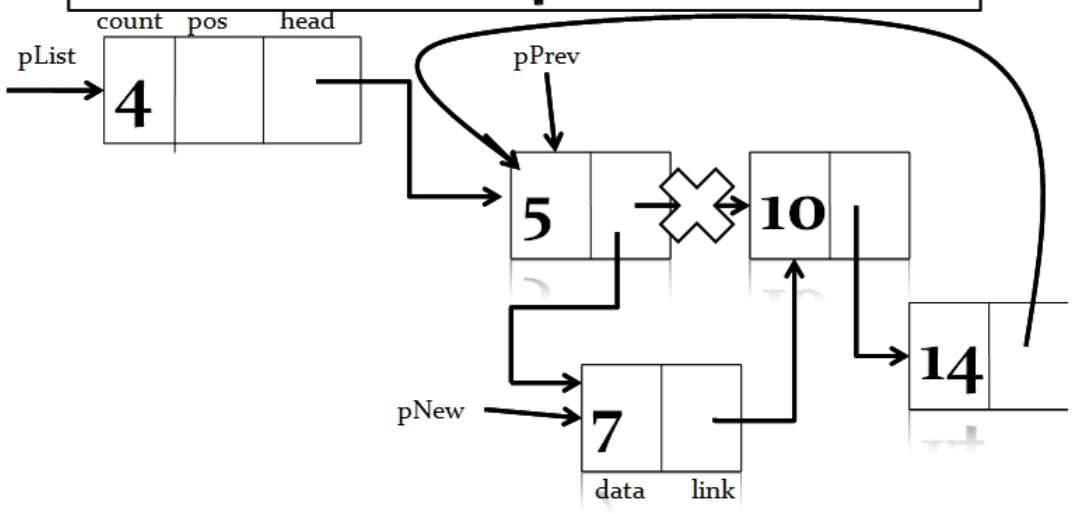
pPrev->link=pNew



Navlakhi™

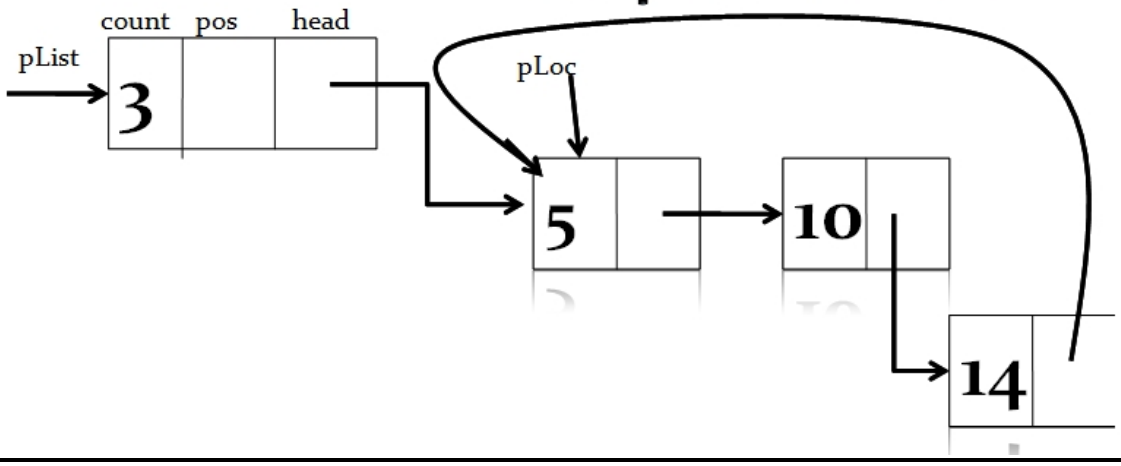
Circular Linked List – Adding 7

pList->count++



Navlakhi™

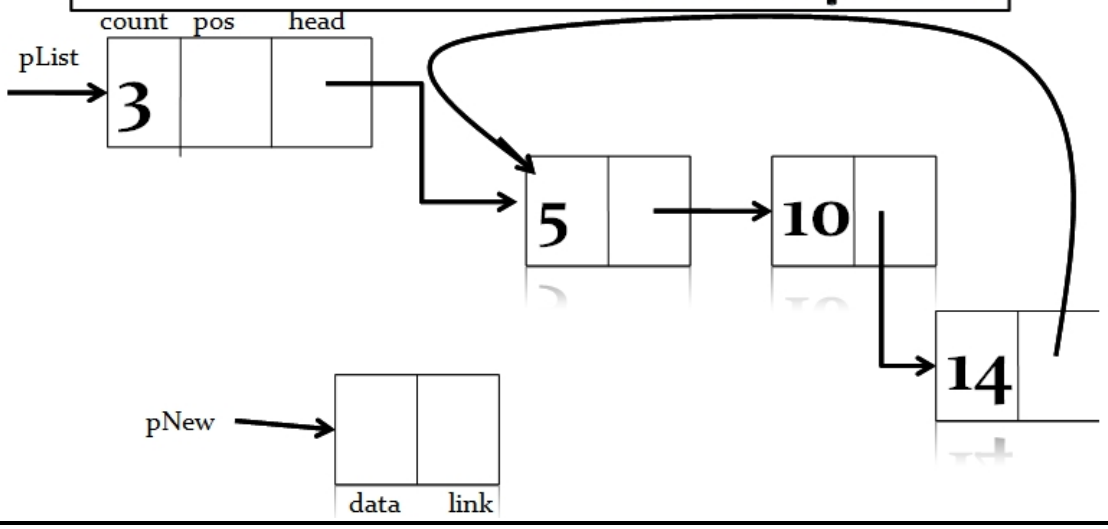
Circular Linked List – Adding 2 i.e. pPrev=NULL



Navlakhi™

Circular Linked List – Adding 2

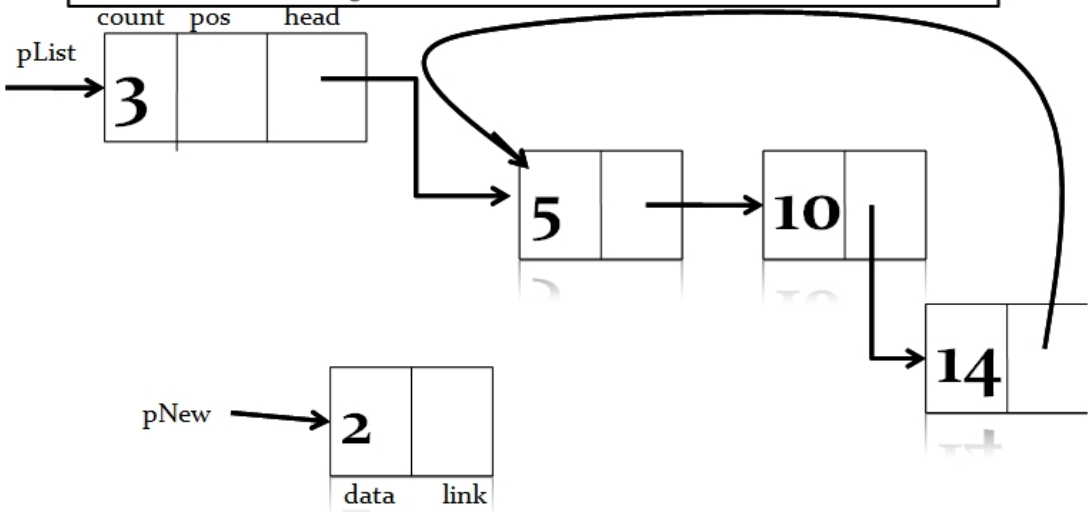
malloc pNew



Navlakhi™

Circular Linked List – Adding 2

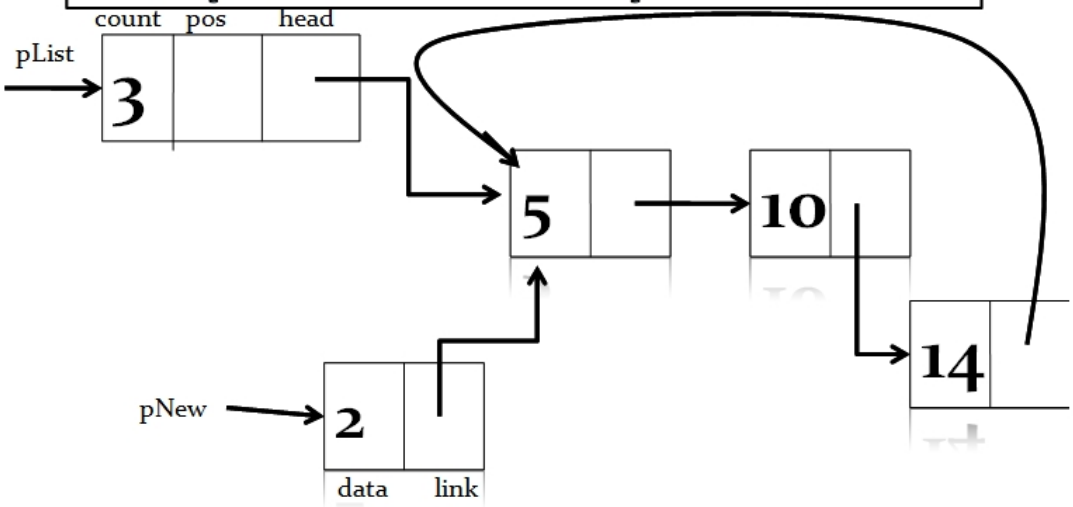
pNew->data = datain



Navlakhi™

Circular Linked List – Adding 2

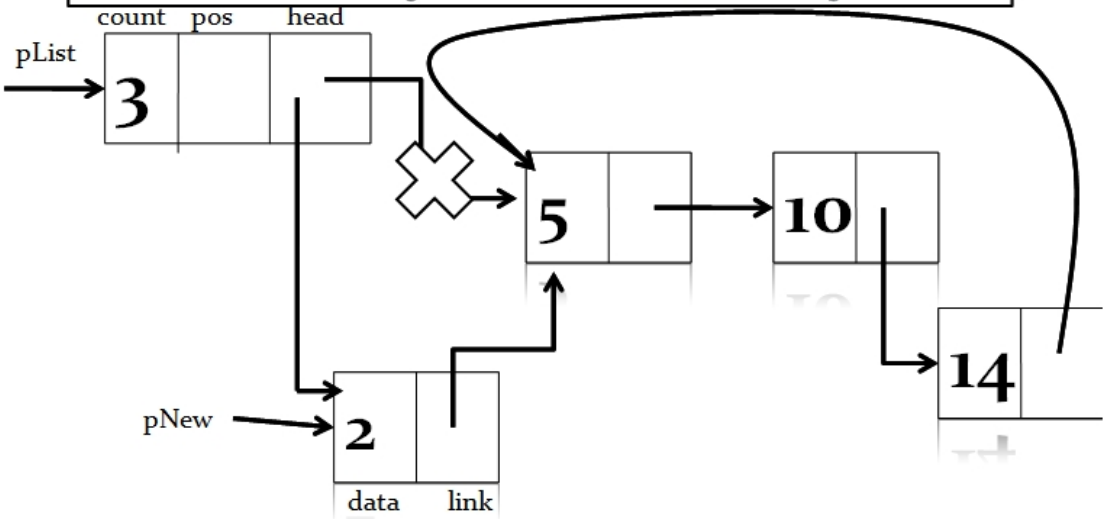
pNew->link = pList->head



Navlakhi™

Circular Linked List – Adding 2

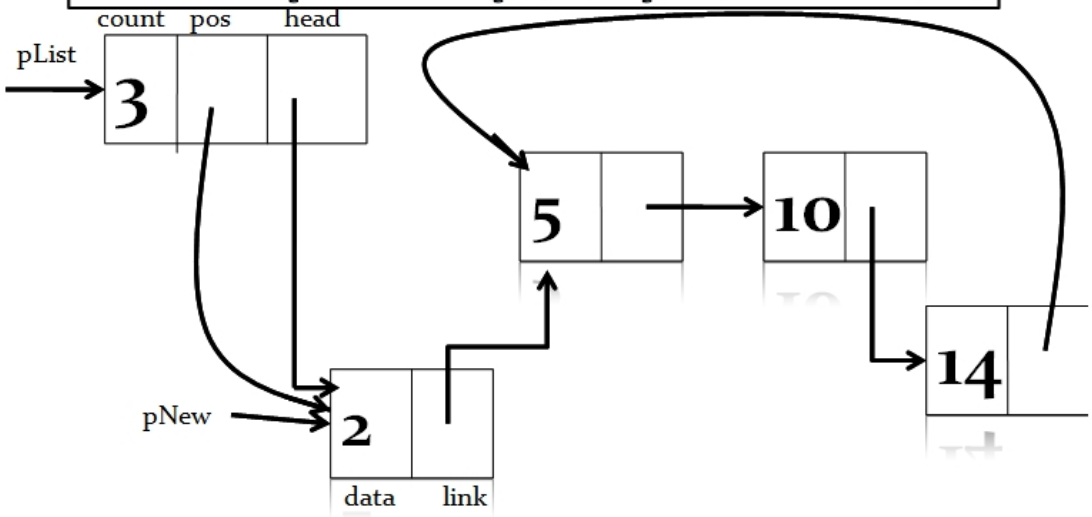
pList->head= pNew



Navlakhi™

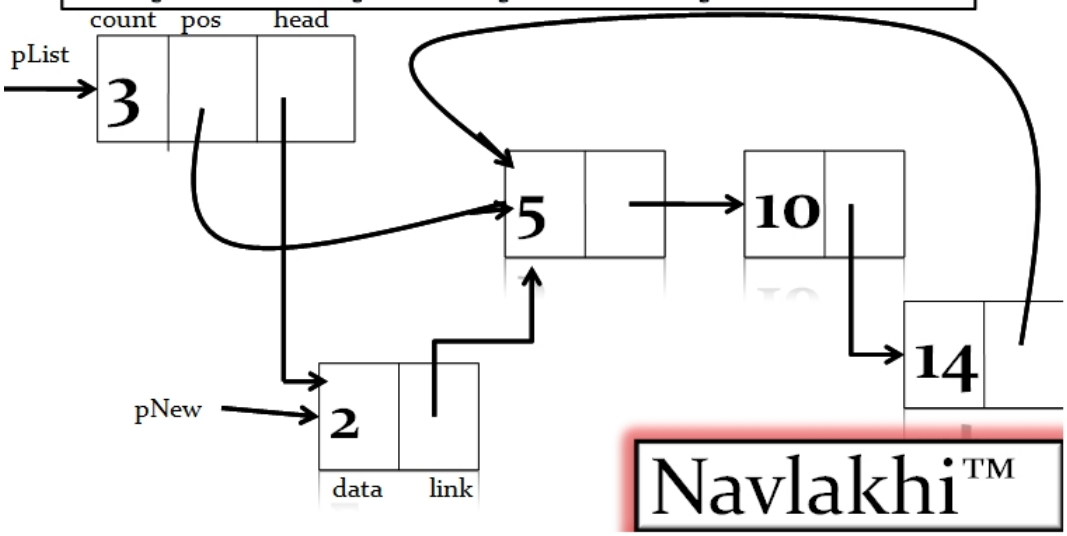
Circular Linked List – Adding 2

```
pList->pos=pList->head
```



Circular Linked List –Adding 2

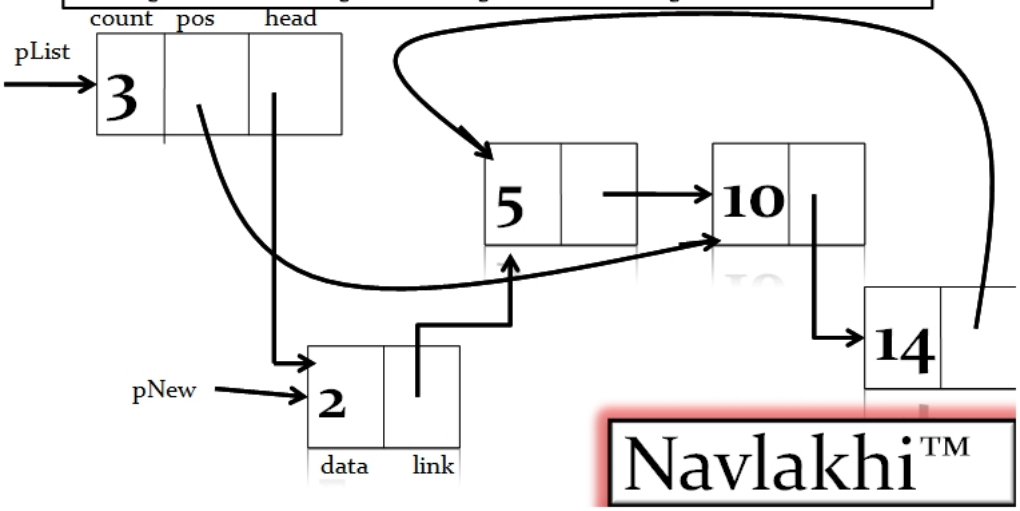
```
pList->count times do
pList->pos=pList->pos->link
```



Navlakhi™

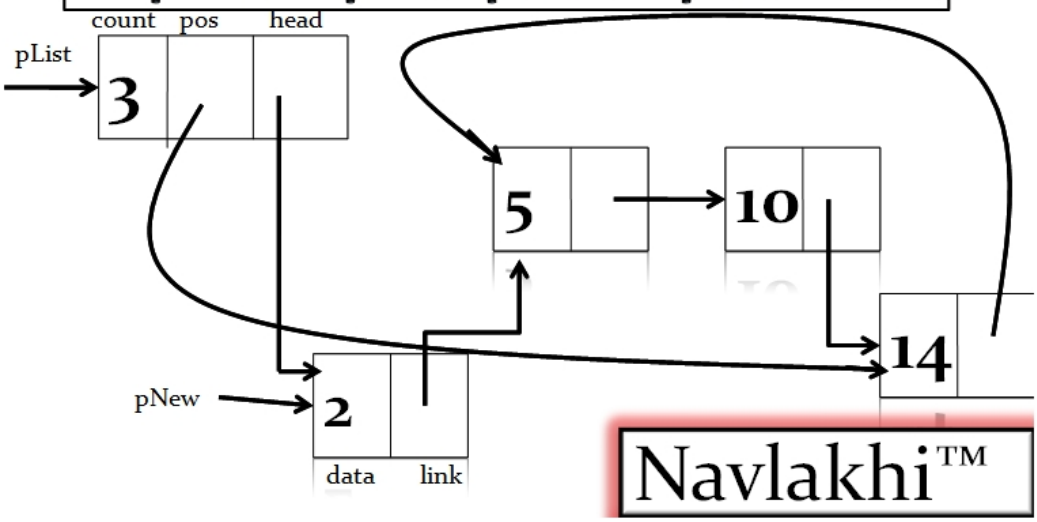
Circular Linked List –Adding 2

```
pList->count times do
pList->pos=pList->pos->link
```



Circular Linked List –Adding 2

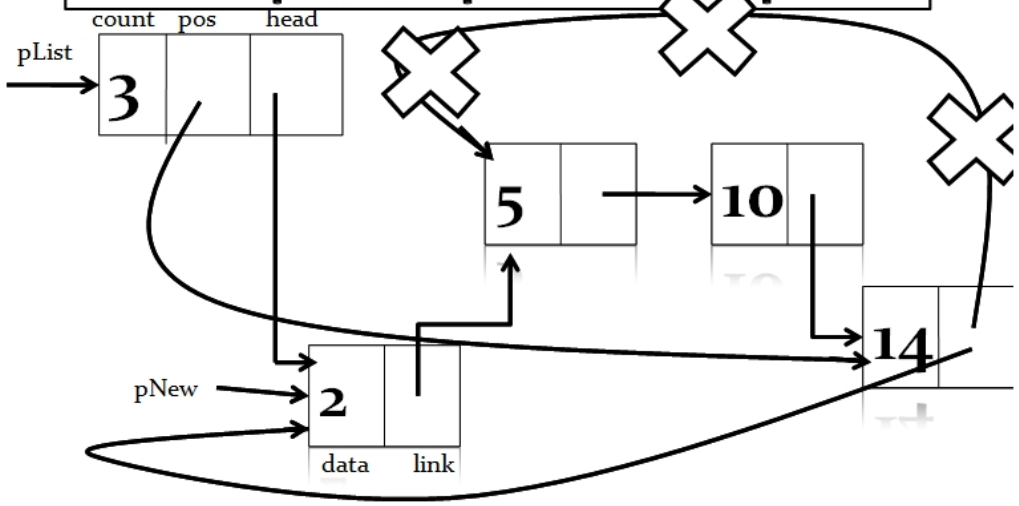
```
pList->count times do
pList->pos=pList->pos->link
```



Navlakhi™

Circular Linked List –Adding 2

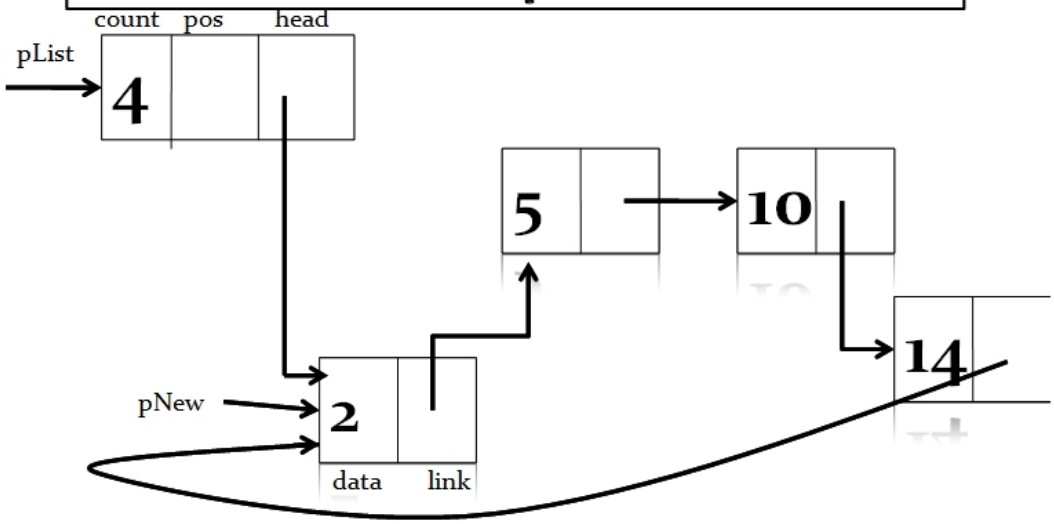
pList->pos->link=pNew



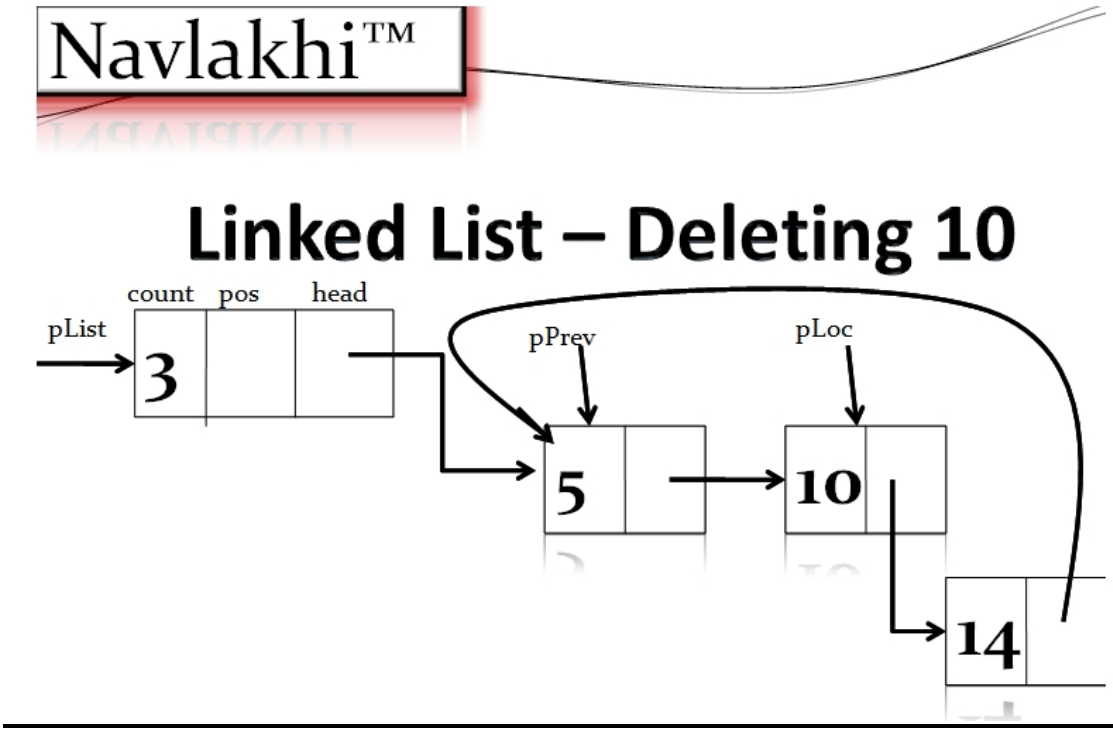
Navlakhi™

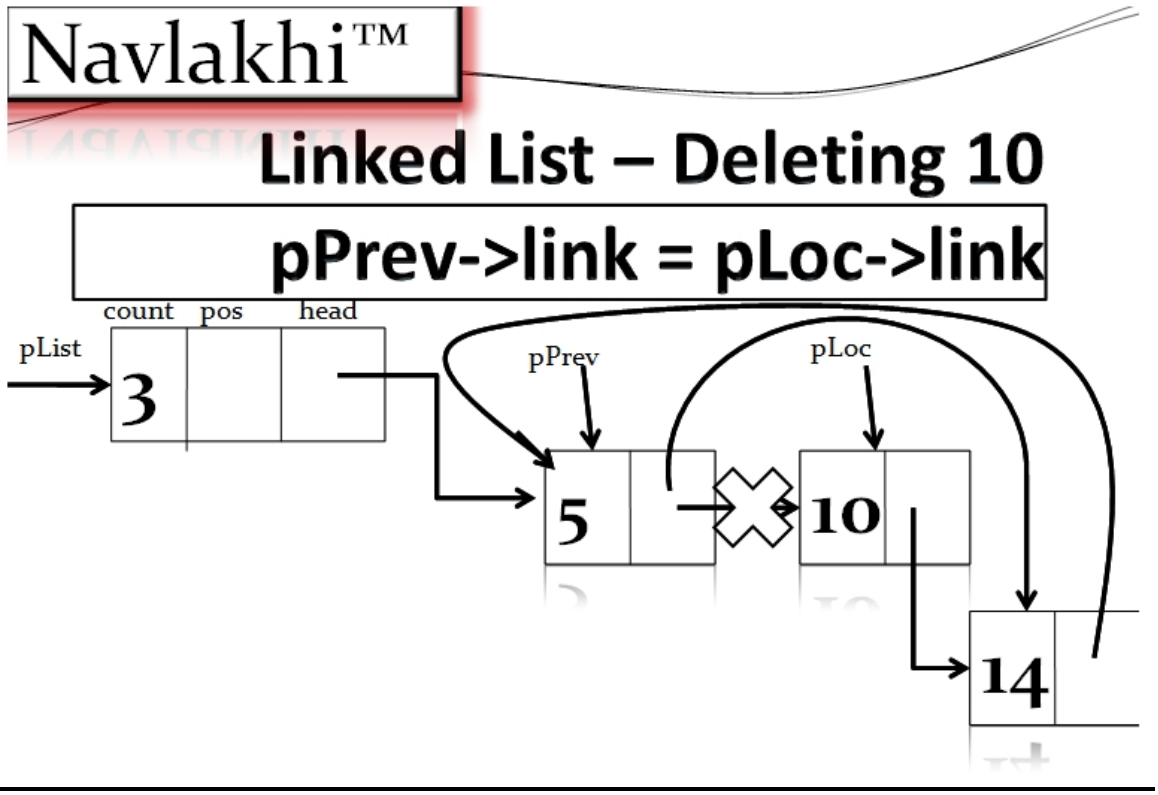
Circular Linked List –Adding 2

pList->count++



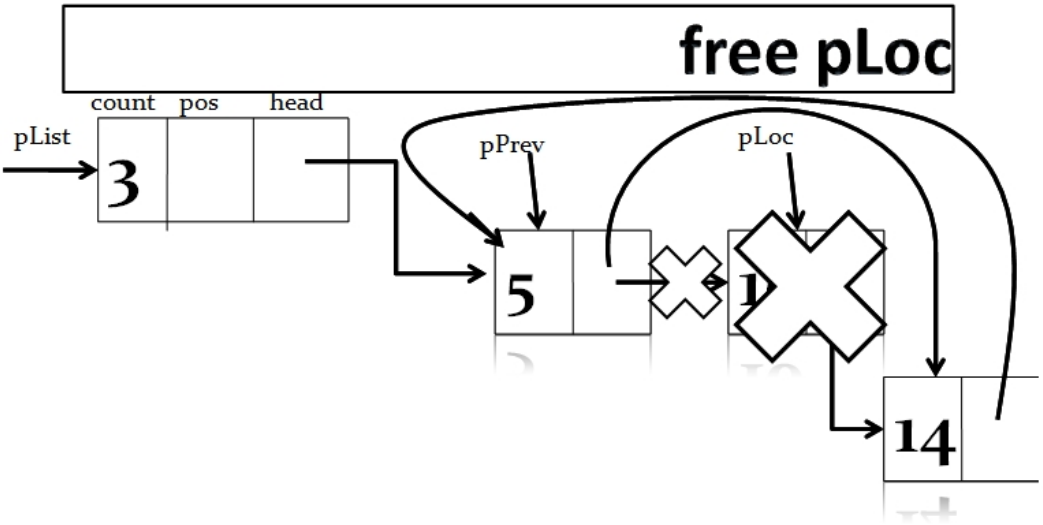
DELETING A NODE





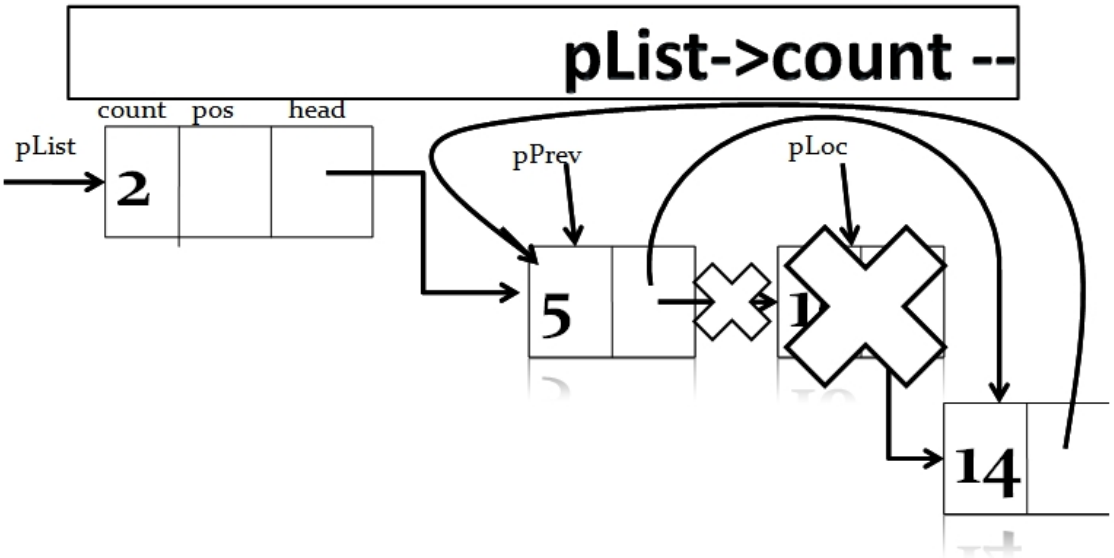
Navlakhi™

Linked List – Deleting 10



Navlakhi™

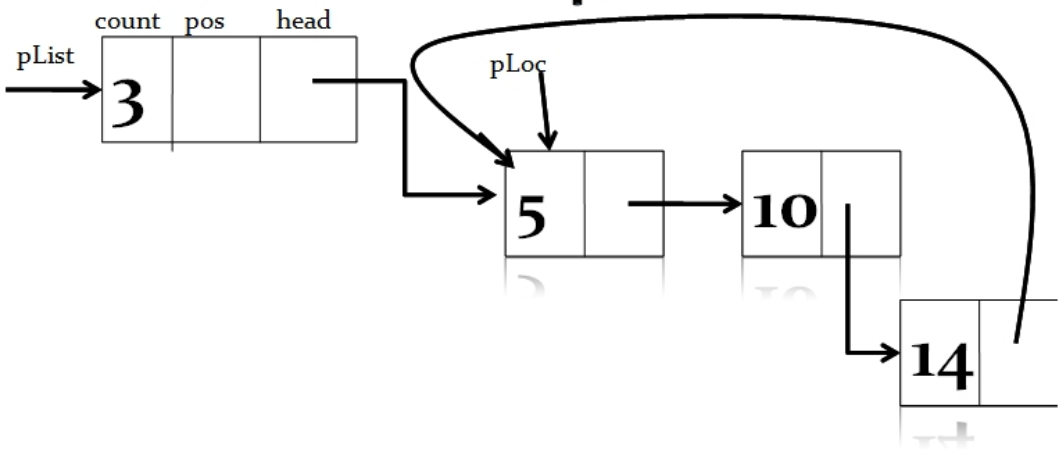
Linked List – Deleting 10



Navlakhi™

Linked List – Deleting 5

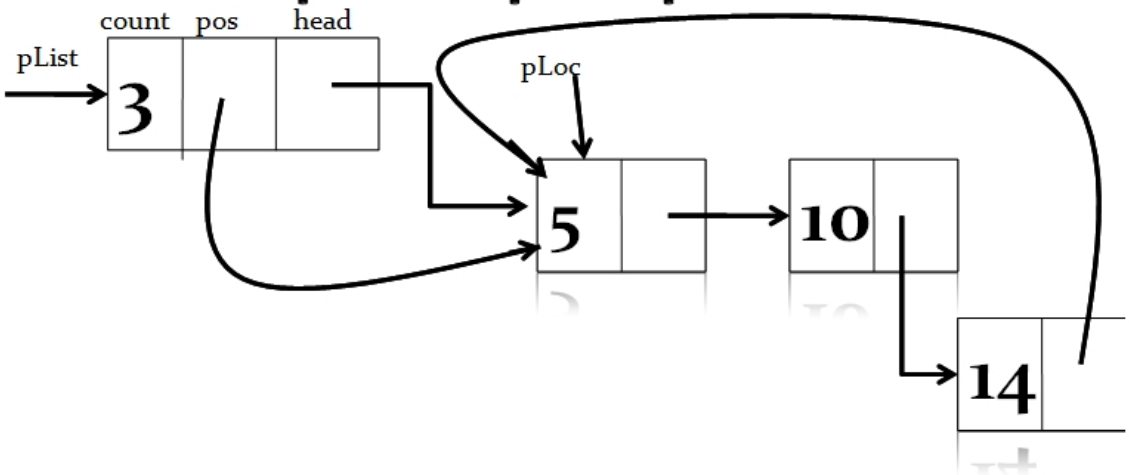
pPrev = NULL



Navlakhi™

Linked List – Deleting 5

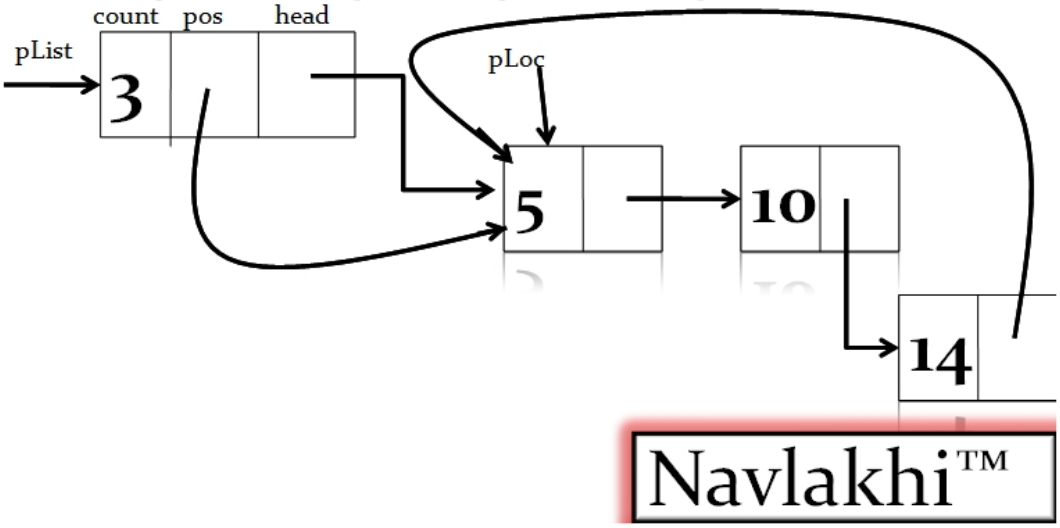
pList->pos=pList->head



Linked List – Deleting 5

pList->count -1 times

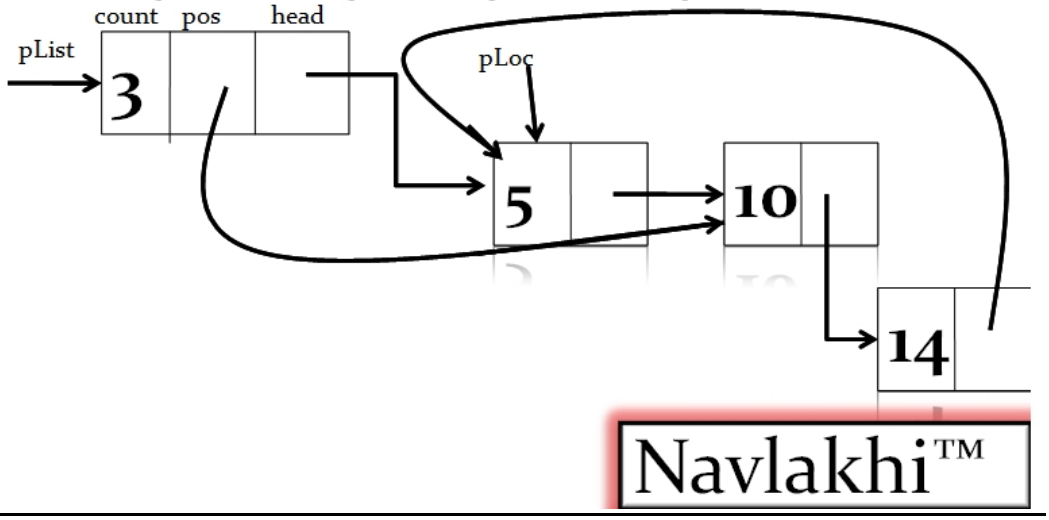
pList->pos=pList->pos->link



Linked List – Deleting 5

pList->count -1 times

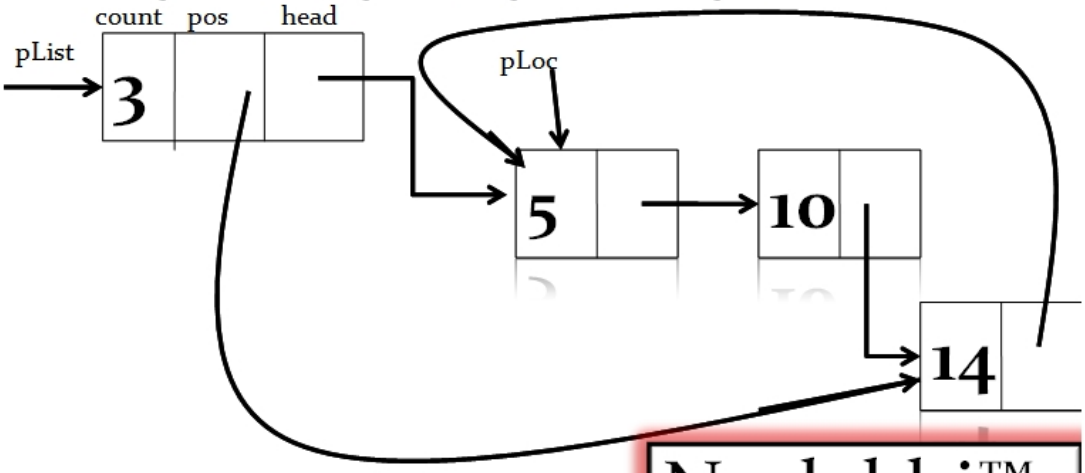
pList->pos=pList->pos->link



Linked List – Deleting 5

pList->count -1 times

pList->pos=pList->pos->link

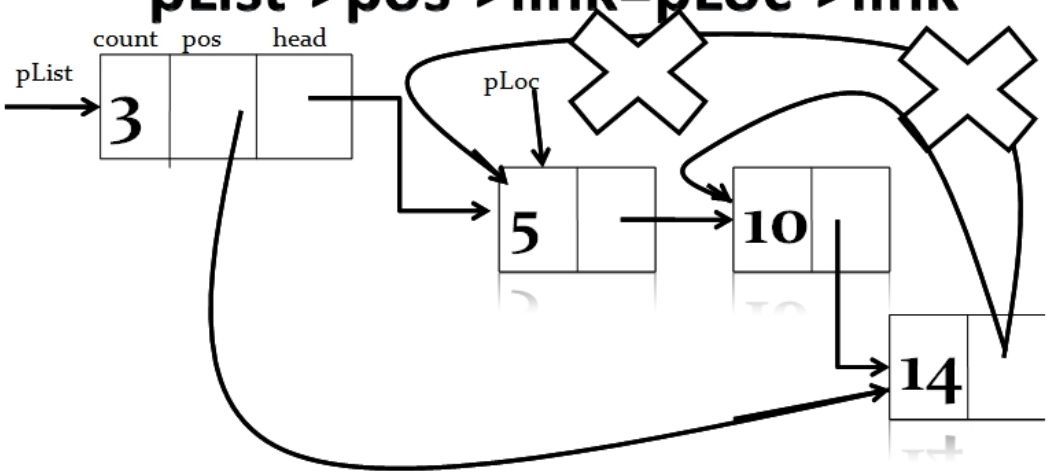


Navlakhi™

Navlakhi™

Linked List – Deleting 5

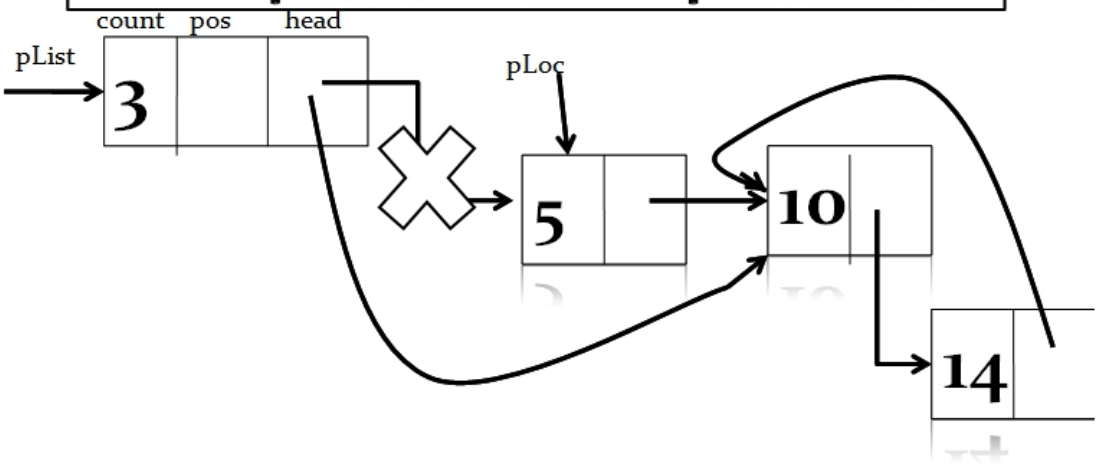
pList->pos->link=pLoc->link



Navlakhi™

Linked List – Deleting 5

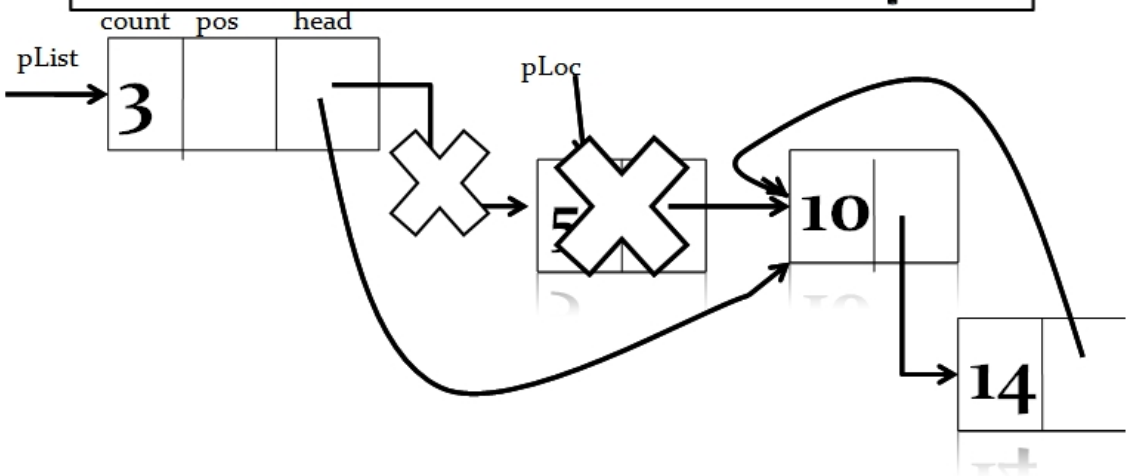
pList ->head=pLoc->link



Navlakhi™

Linked List – Deleting 5

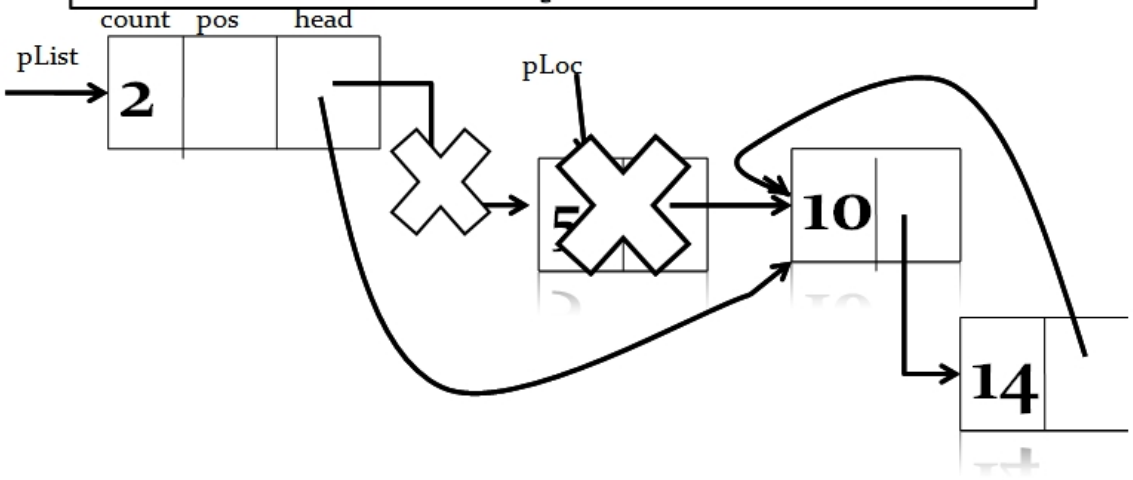
free pLoc



Navlakhi™

Linked List – Deleting 5

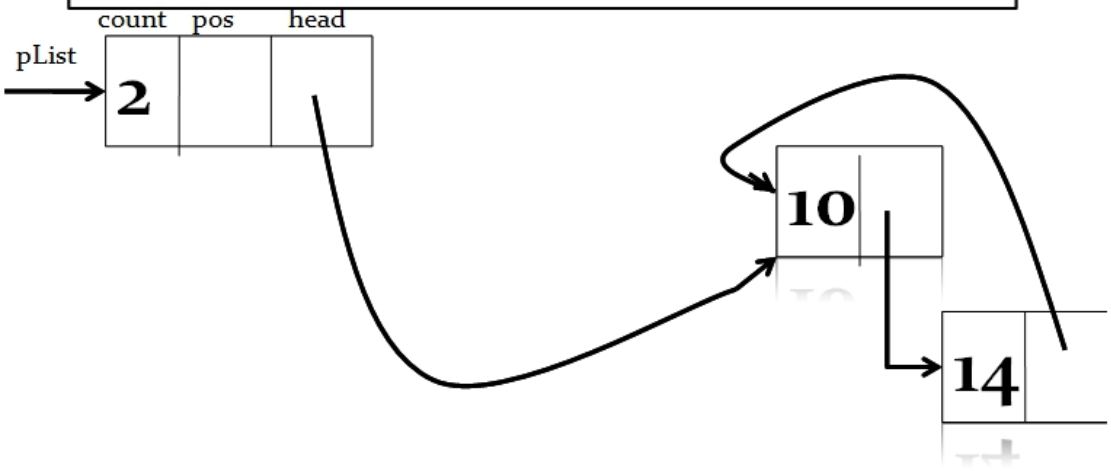
pList ->count - -



Navlakhi™

Linked List – Deleting 5

Final Senario



Program

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

struct node
{
int data;
struct node *link;
};

struct list
{
int count;
struct node *pos;
struct node *head;
}*pList;


struct node *pPrev,*pLoc;
```

```
int searchNode(int target)
{
int i;
pPrev=NULL;

pLoc=pList->head;

for (i=1;i<=pList->count;i++)
{
        if (target<=pLoc->data) break;
        pPrev=pLoc;
        pLoc=pLoc->link;
}

if (i>pList->count) pLoc=NULL;

if (pLoc==NULL)
        return 0; /*Not found*/
else
        if (target == pLoc->data) return 1; /*FOUND*/
else
        return 0;

}

void printList()
{
int i;
pList->pos=pList->head;
for (i=1;i<=pList->count;i++)
{
        printf("%d\t",pList->pos->data);
        pList->pos=pList->pos->link;
}
printf("\n******** END OF LIST ************\n");
}
```

```
void deleteNode()
{
int i;
if (pPrev==NULL) /*Removing 1st node*/
{
                    if (pList->count==1) pList->head=NULL;
                    else
                    {
                            pList->pos=pList->head;
                            /*Moving to the last node*/
                            for (i=1;i<pList->count;i++)
                                    pList->pos=pList->pos->link;

                            /*Setting the last nodes link to the new 1st node*/
                            /*Node the 1st node is to be dropped*/
                            /*Thus 2nd node is to become the new 1st node*/
                            pList->pos->link=pList->head->link; /*pList->pos->link=pLoc->link */

                            /*Now moving the head pointer*/
                            pList->head=pLoc->link;
                    }
}
else                pPrev->link=pLoc->link;

pList->count =pList->count - 1;
free(pLoc);
}



void removeNode(int key)
{
int found;

found=searchNode(key);

if (found)    deleteNode();
else   printf("Error: No matching data found\n");
}
```

```
int insertNode( int dataIn)
{
struct node *pNew;
int i;
pNew = (struct node *) malloc(sizeof(struct node));
if (pNew != NULL)
{
        pNew->data=dataIn;

        if (pPrev!=NULL)
        {
                pNew->link=pPrev->link;
                pPrev->link=pNew;
        }
        else
        {
                /*** Insert into Empty List or Start of LIST ***/
                pNew->link=pList->head;
                pList->head=pNew;
                /* Move last node link to this node */
                pList->pos=pList->head; /*Using pos as a temporary pointer*/

                /*Moving pos to the last node*/
                for(i=1;i<=pList->count;i++)
                pList->pos=pList->pos->link;

                /*Setting the last node's link to the new 1st node*/
                pList->pos->link=pNew;
        }
        pList->count+=1;
        return 1;
}
else
        return 0;
}
```

```
void addNode( int dataIn)
{
int found,success;

found = searchNode(dataIn);

if (found)
        printf("Data already inserted\n");
else
{
        success=insertNode(dataIn);

        if (success)
                printf("Data Inserted Successfully\n");
        else
                printf("Out of Memory.....\n");
}
}
```

```
int menu( )
{
int choice;
printf("\n\n********************\n\n");
printf(" ....   M E N U ...\n");
printf("1: Add new data\n");
printf("2: Delete data\n");
printf("3: Print List\n");
printf("4: Quit\n\n");
printf("********************\n\n");
        printf("feed in your choice: ");
        scanf("%d",&choice);

        return choice;
}


void createList( )
{
pList = (struct list *)malloc(sizeof(struct list));
if (pList != NULL)
{
        pList -> head=NULL;
        pList -> count=0;
}
else
{
        printf("Insufficient memory...\n");
        exit(0);
}

}
```

```
void main( )
{
int choice;
int dataIn,deleteKey;
clrscr();
createList();

do
{
        choice = menu();

        if (choice==1)
    {
                printf("Feed in the data: ");
                scanf("%d",&dataIn);
                addNode(dataIn);
    }
        else
        if (choice==2)
        {
                  printf("Enter key to be deleted: ");
                  scanf("%d",&deleteKey);
                  removeNode(deleteKey);
        }
        else
        if (choice==3)
        {
           printList();
        }

} while(choice!=4);
}
```

HOME OF EDUCATION

Navlakhi®

www.navlakhi.com
Home of Education

DATA Structures@ Navlakhi's