



Linked List

Methodology and Program

By Abhishek Navlakhi
Semester 3: Data Structures

This document is for private circulation for the students of Navlakhi®.
More educational content can be found on navlakhi.education
Contact Numbers 9820246760/9769479368/9820009650/51/022-23548585

Linked list is a connected list of items. This form of structure is useful over the array type data structure when an unpredictable and large volume of data is expected. The major objective for storage of data is to make retrieval faster. Hence there has to be some logical way of adding data items in the list. Here we will be using the ascending order as the storage order for the data item, hence we can stop as soon as we get an item greater or equal to the target item specified.

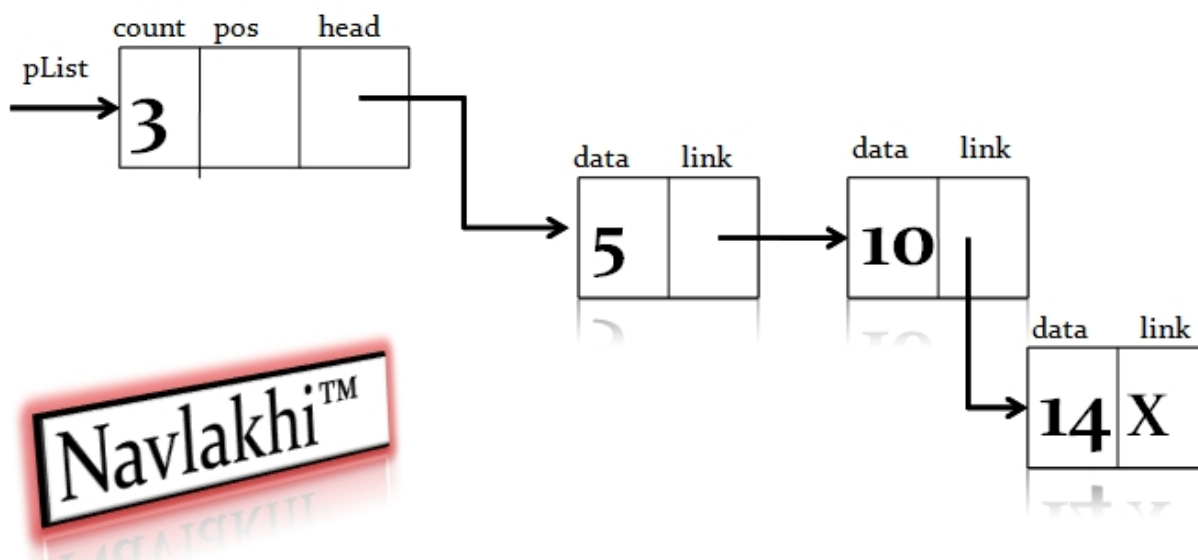
Lets lay down a few programming basics

ADDITION OF DATA

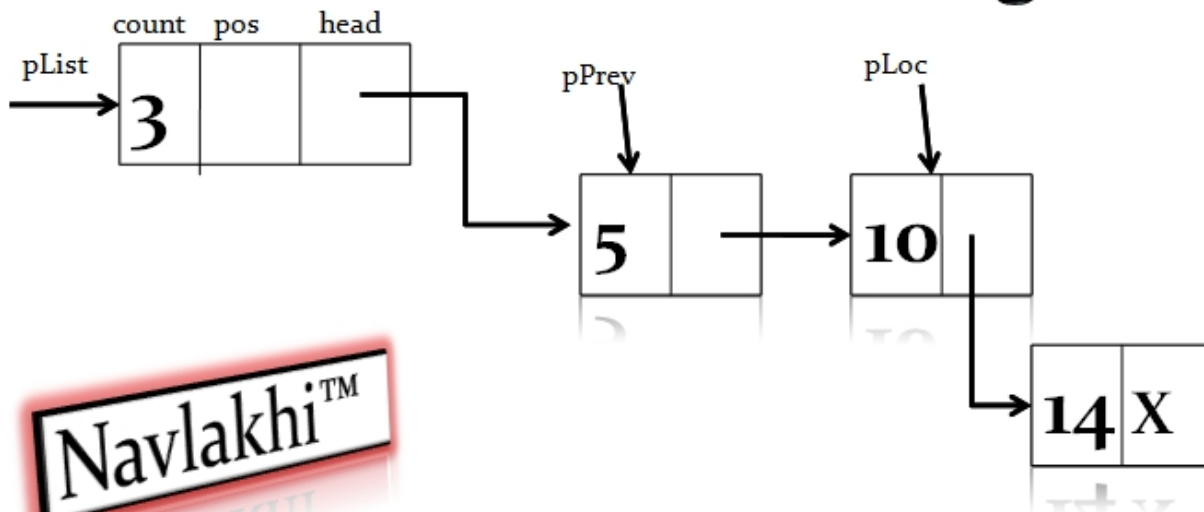
Case 1: pPrev!=NULL

Assuming we are adding 7, and the searchNode function will first locate where to insert the new data item. 'pPrev' will point (refer) to the location just smaller than the data to be inserted (7 in this case) & 'pLoc' will point to the next item.

Linked List

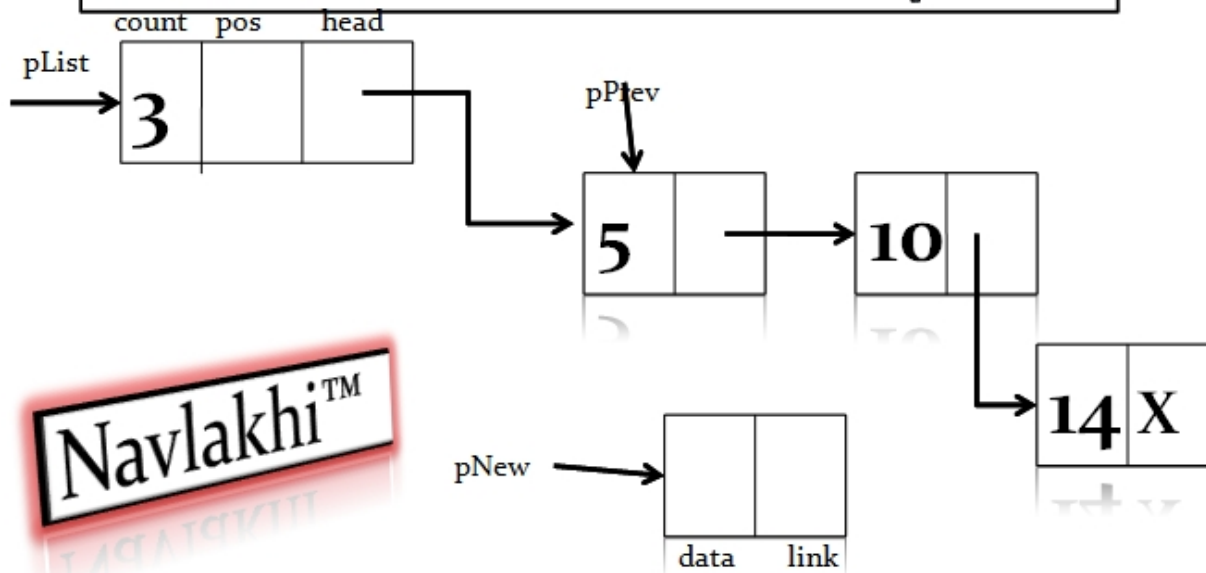


Linked List – Adding 7



Navlakhi™

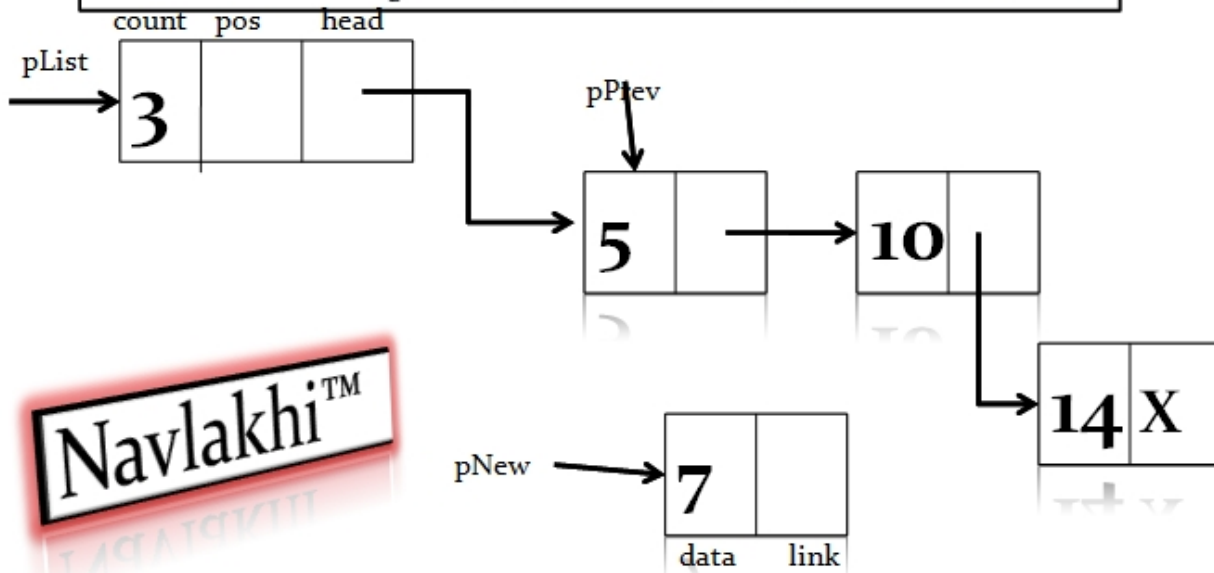
Linked List – Adding 7 malloc pNew



Navlakhi™

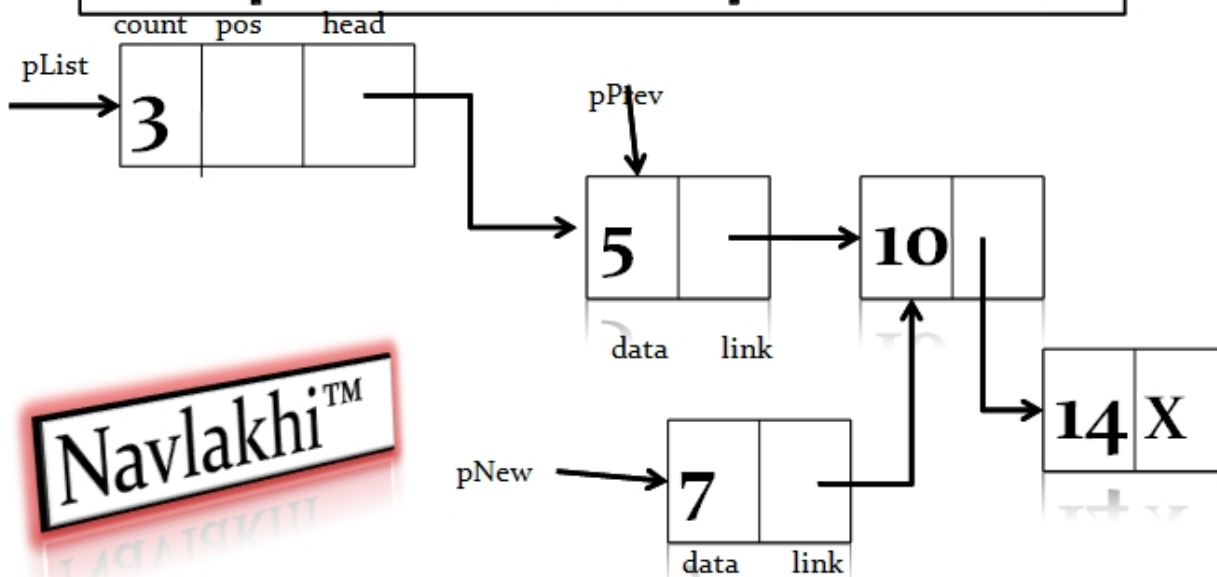
Linked List – Adding 7

pNew->data = dataIn



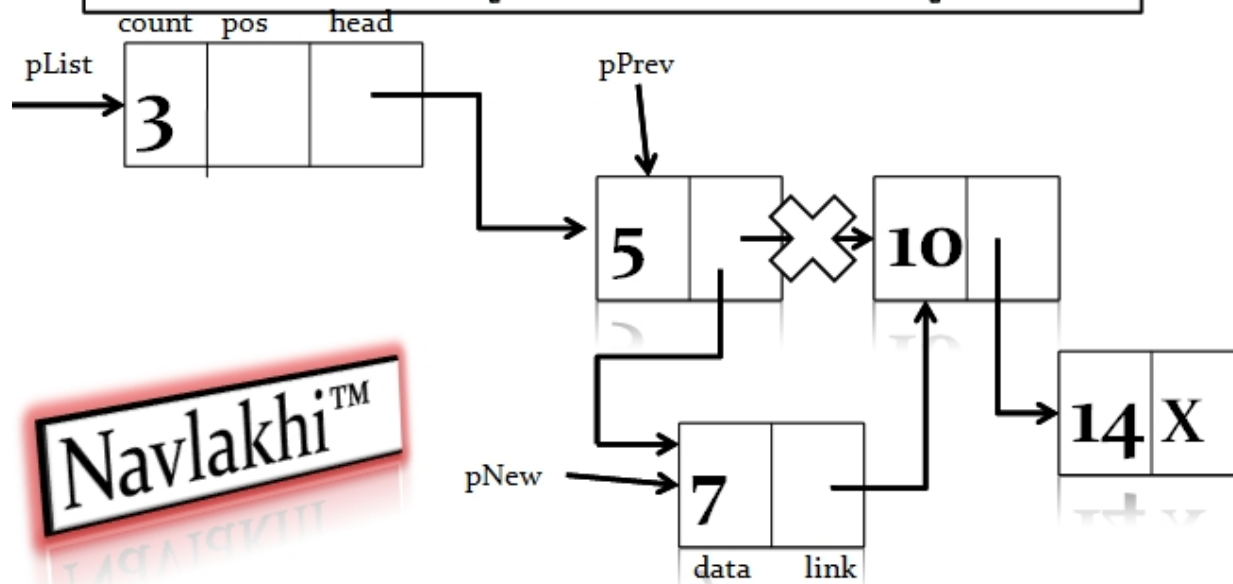
Linked List – Adding 7

pNew->link = pPrev->link



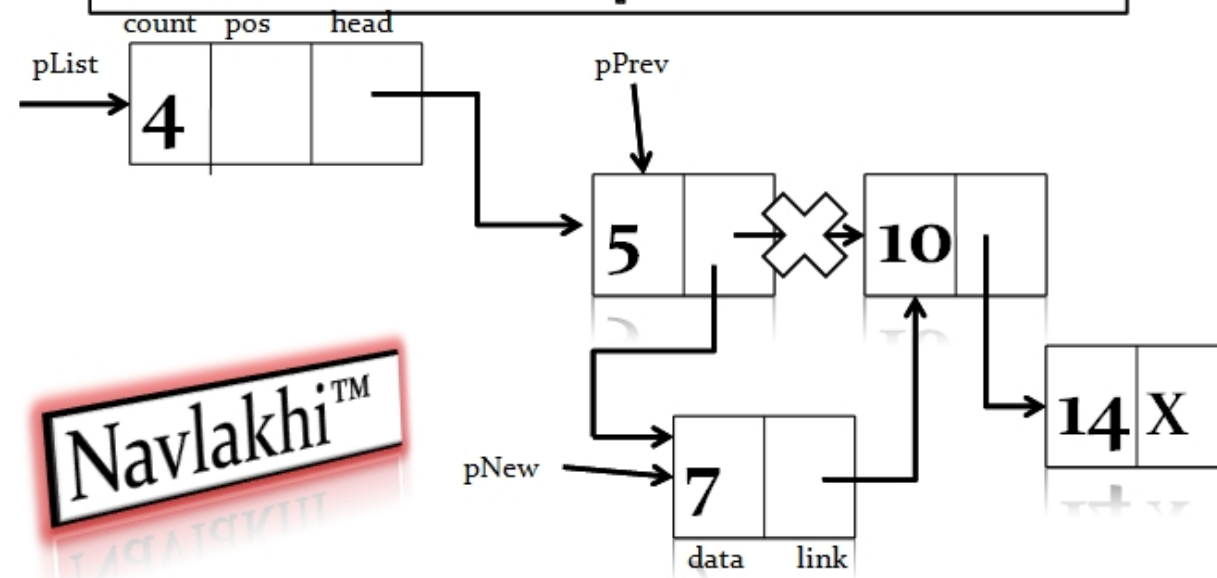
Linked List – Adding 7

pPrev->link=pNew



Linked List – Adding 7

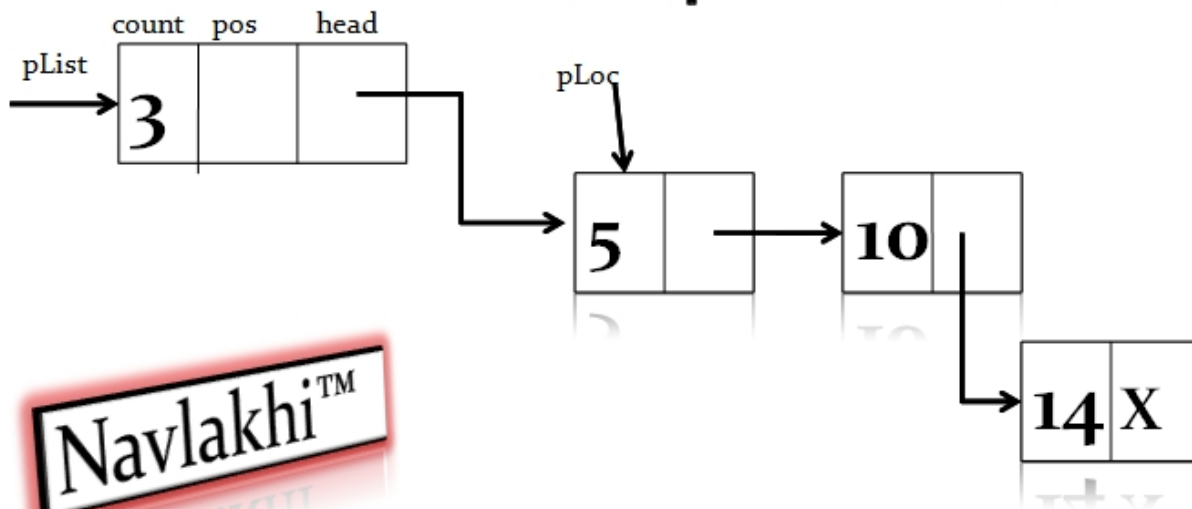
pList->count++



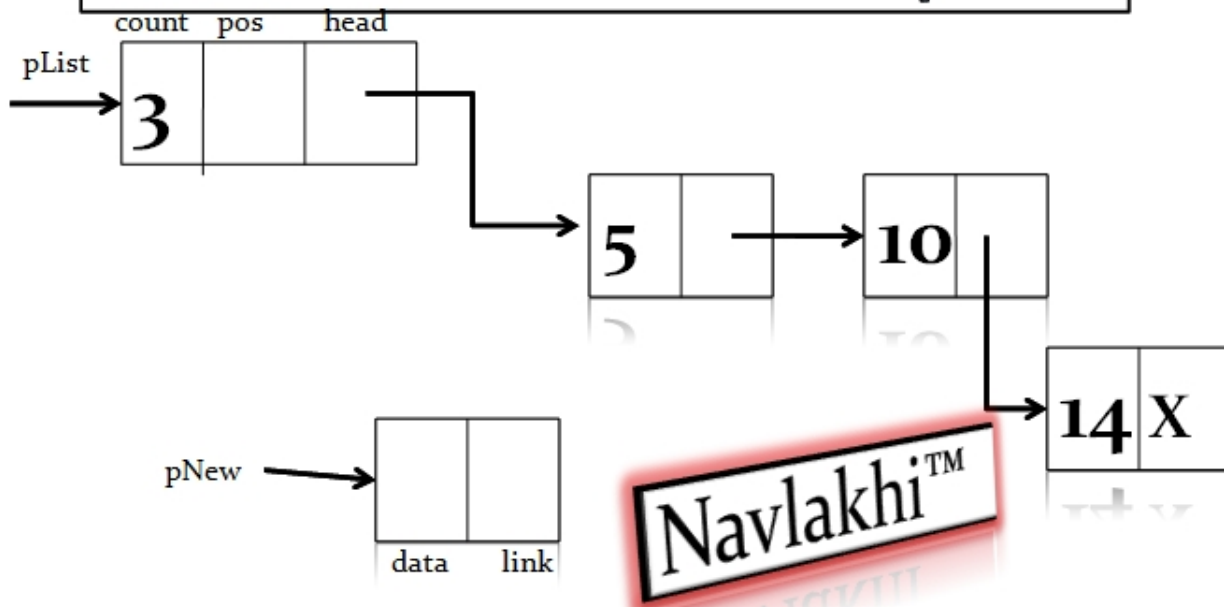
Case 2: pPrev==NULL

Assuming we are adding smaller than the smallest, i.e. before the old first, then pPrev will be NULL

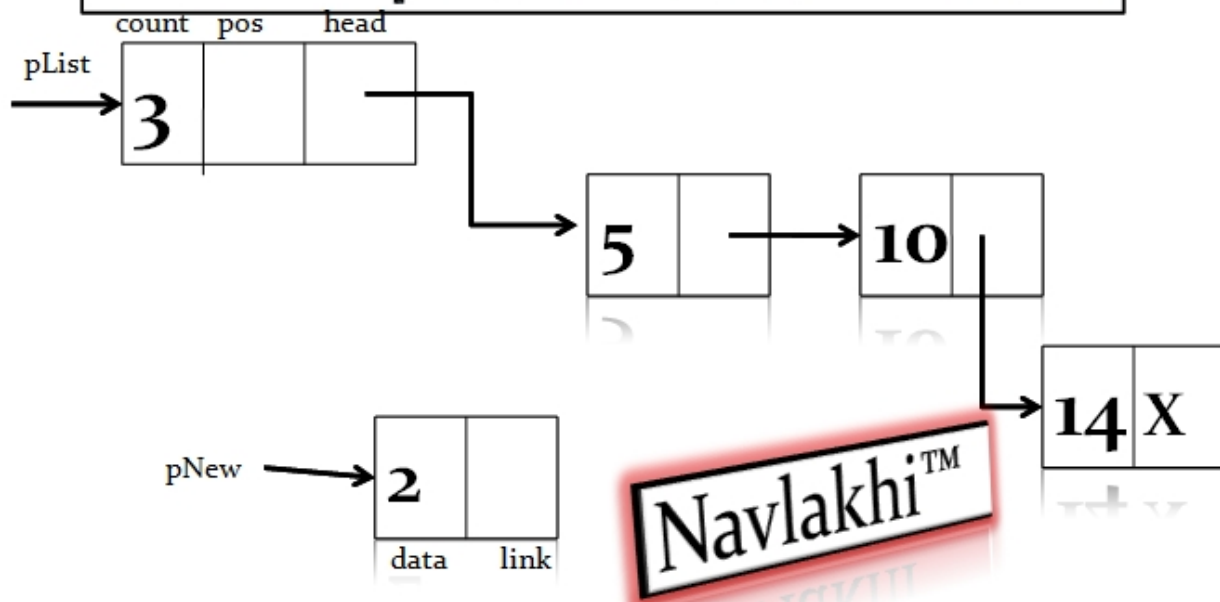
Linked List – Adding 2 i.e. pPrev=NULL



Linked List – Adding 2 malloc pNew

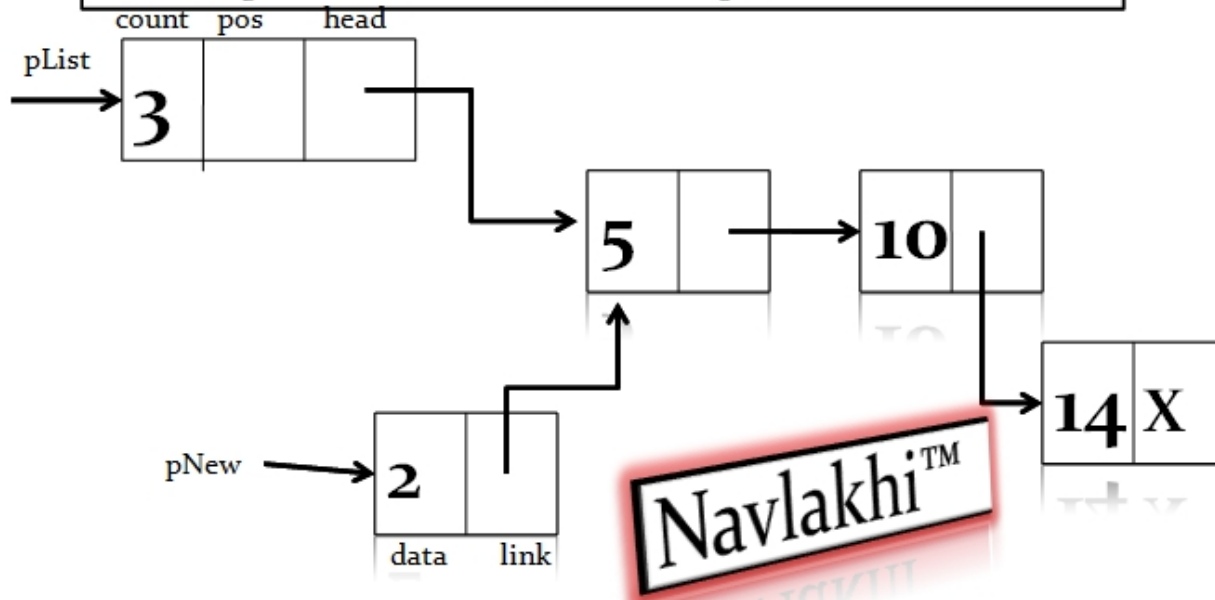


Linked List – Adding 2 pNew->data = datain



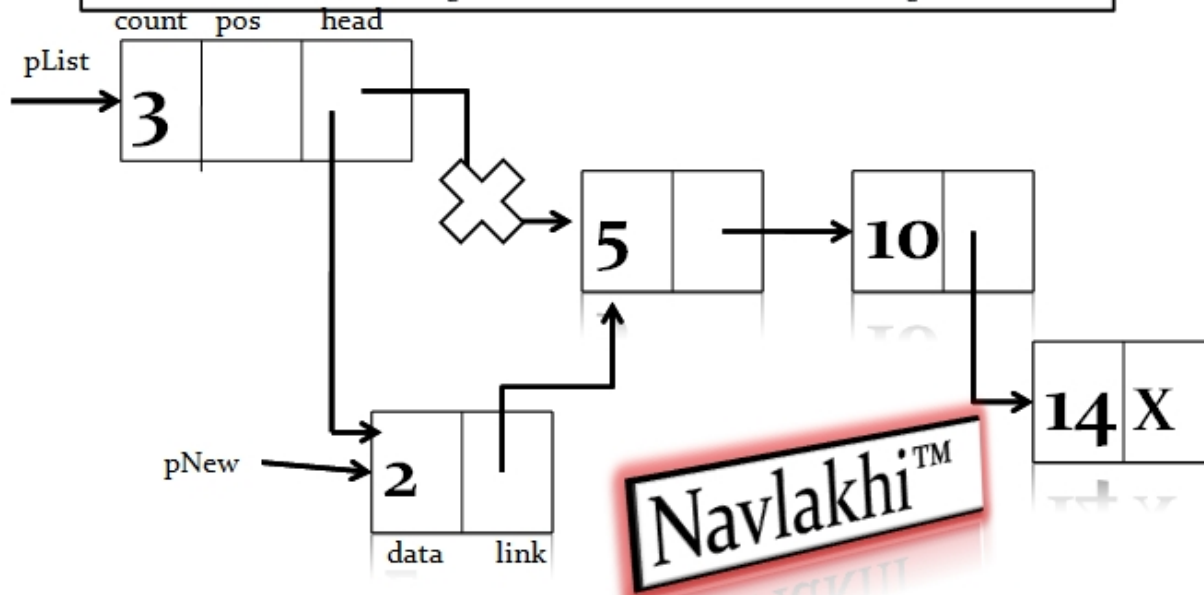
Linked List – Adding 2

pNew->link = pList->head

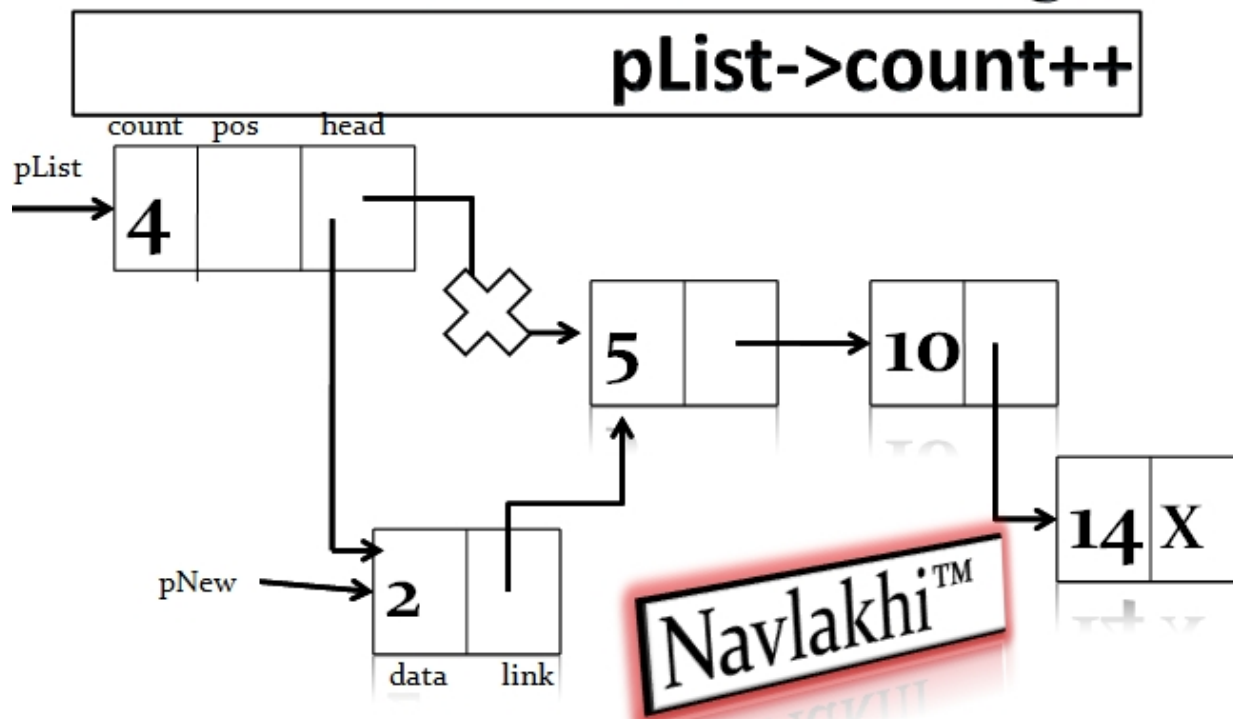


Linked List – Adding 2

pList->head = pNew



Linked List – Adding 2

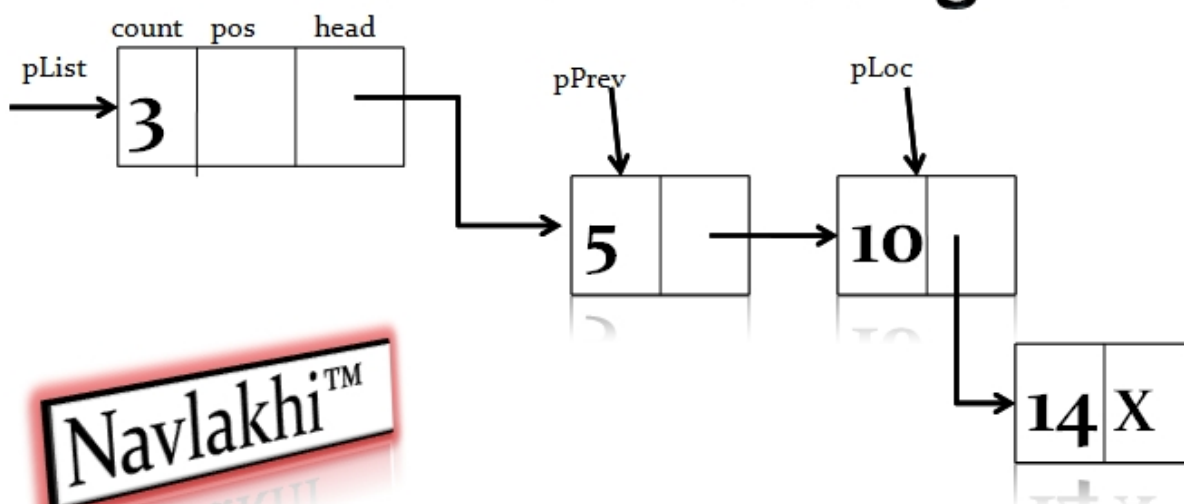


DELETING A NODE

Case 1: pPrev!=NULL (Not deleting the 1st Node)

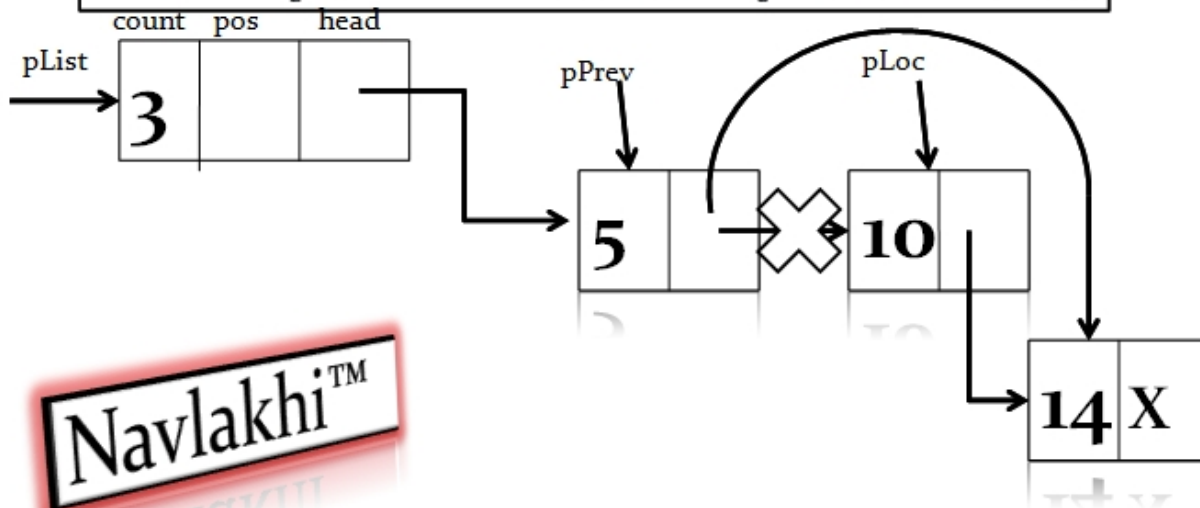
The same searchNode function will set pPrev and pLoc as shown below.

Linked List – Deleting 10



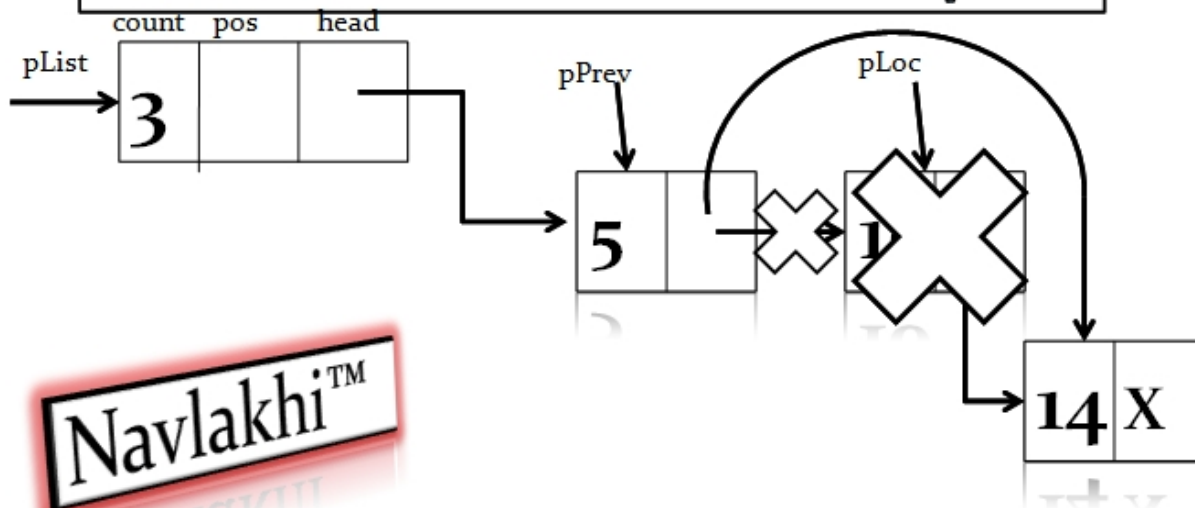
Linked List – Deleting 10

pPrev->link = pLoc->link

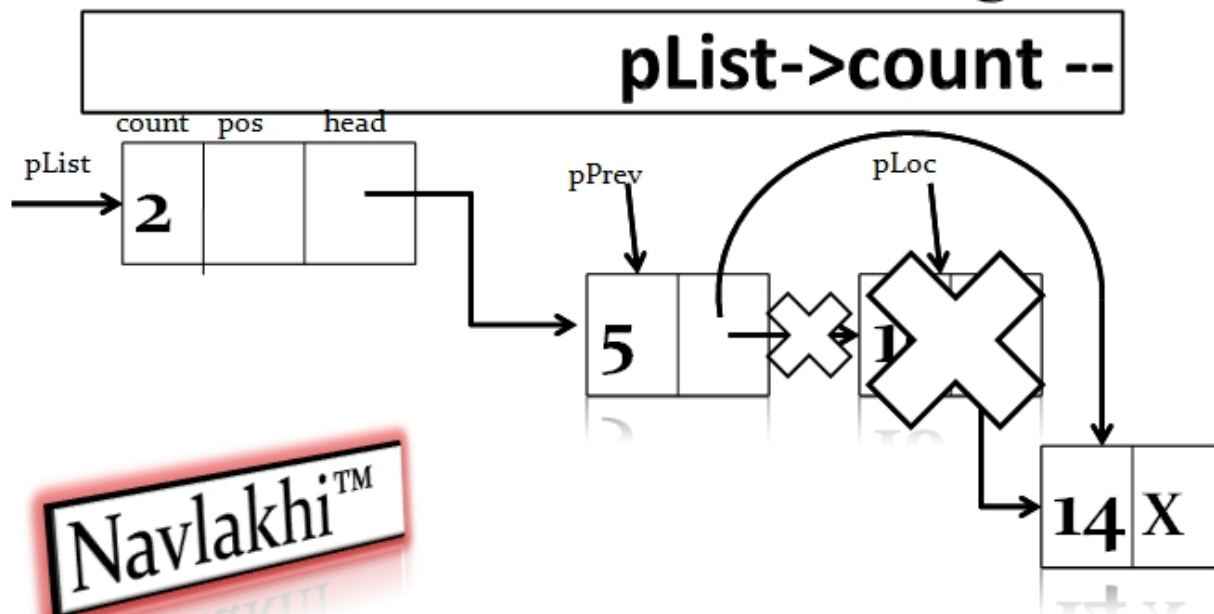


Linked List – Deleting 10

free pLoc



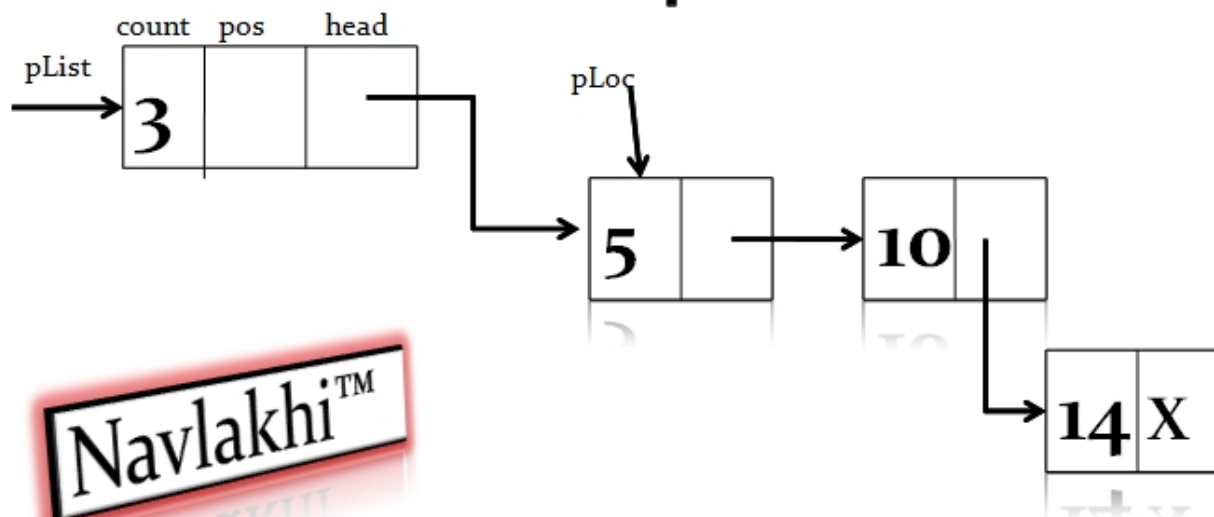
Linked List – Deleting 10



Case 2: Deleting the first Node (pPrev==NULL)

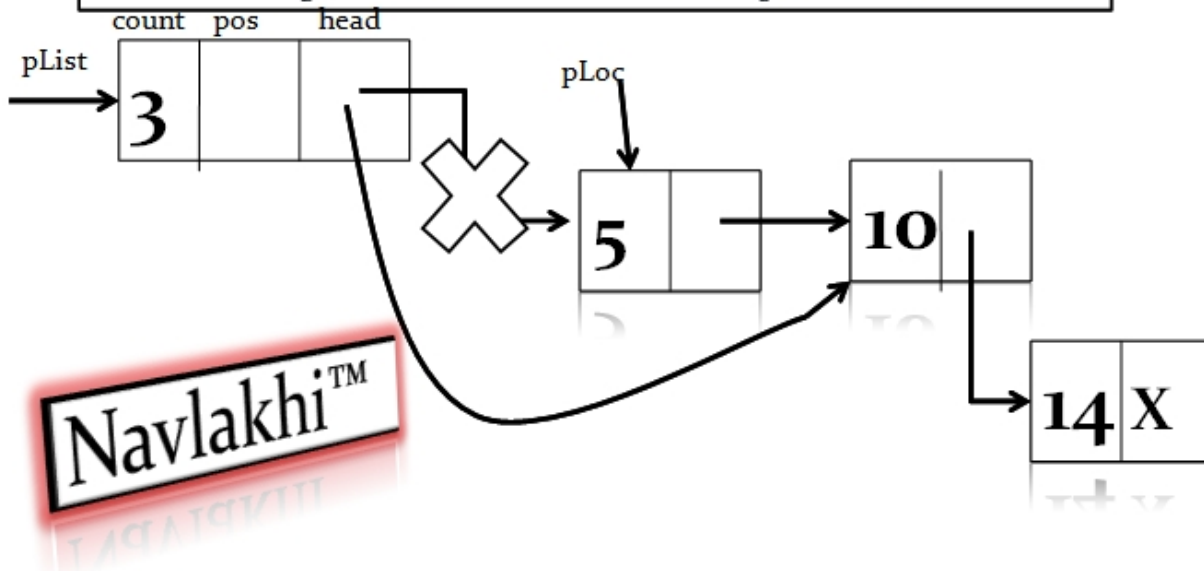
Linked List – Deleting 5

pPrev = NULL

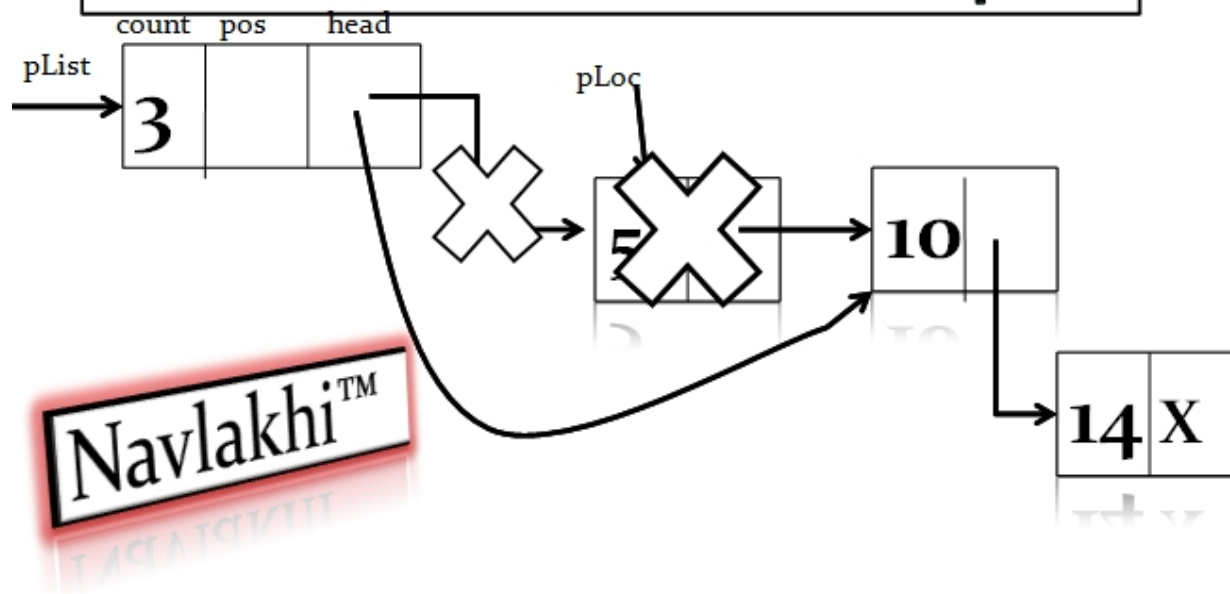


Linked List – Deleting 5

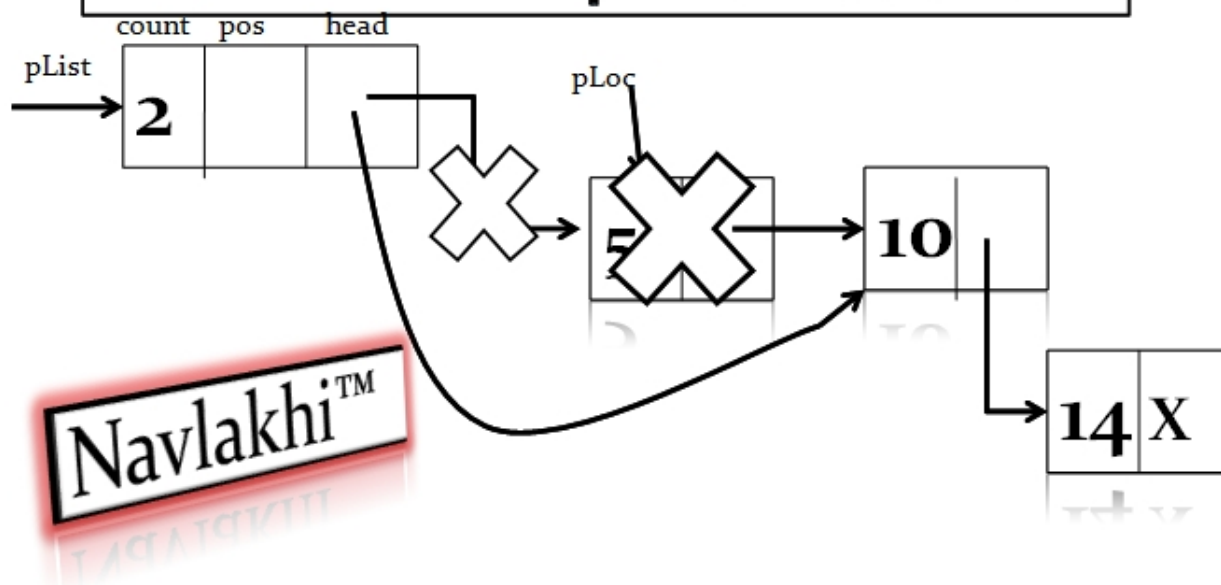
pList ->head=pLoc->link



Linked List – Deleting 5 free pLoc



Linked List – Deleting 5 pList ->count - -



Program

```
/* Singly Linked List */
#include <stdio.h>
#include <alloc.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)
{
    pPrev=NULL;

    pLoc=pList->head;

    while(pLoc!=NULL && target > pLoc->data)
    {
        pPrev=pLoc;
        pLoc=pLoc->link;
    }

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

```
void printList( )
{
    int i;
    if (pList->count==0) printf("The List is Empty \n");
    else
    {
        pList->pos=pList->head;
        printf("..... The List data is as follows ..... \n");
        for(i=0;i< pList->count;i++)
        {
            printf("%d\t",pList->pos->data);
            pList->pos=pList->pos->link;
        }
        printf("\n***** END OF LIST ***** \n");
    }
}
```

```
void deleteNode( )
{
    if (pPrev==NULL)
        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==1)
        deleteNode();
    else
        printf("Error: No matching data found\n");
}
```



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

        if (pPrev!=NULL)
        {
            pNew->link=pPrev->link;
            pPrev->link=pNew;
        }
        else
        {
            pNew->link=pList->head;
            pList->head=pNew;
        }
        pList->count+=1;
        return 1;
    }
    else
        return 0;
}
```

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

    found = searchNode(dataIn);
    if (found==1) printf("Data already inserted\n");
    else
    {
        success=insertNode(dataIn);
        if (success==1)    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 to create Head Node...Exiting..\n");
        exit(1);
    }
}
```

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

    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™

**The
BEST
Teaching** **Supers**

No 1. in Engineering Coaching