



THE UNIVERSITY  
OF LAHORE  
**ISLAMABAD  
CAMPUS**

**Data Structures and Algorithms  
(CS09203)**

**Lab Report**

Name: Rehman ullah baig

Reg No: CSU-F16-118

Lab Report: 5

Submitted To: Sir, Usman Ahmed

## Lab: 05

### Link list-Basic Deletion at desired position

#### Objective:-

The objective of this session is to insertion, traversal and deletion at desired position in link list using C++.

#### Software Tools:-

Dev C++

#### Theory:-

This section discusses how to insert an item into, and delete an item from, a linked list. Consider the following definition of a node. (For simplicity, we assume that the info type is int.

```
struct nodeType
{
    int info nodeType* link;
};
```

We will use the following variable *nodeType*

*\*head, \*p, \*q, \*newNode;* **INSERTION:-**

Algorithms which insert nodes into the linked list come up in various situations. We discuss three of them here. The first one inserts a node at the beginning of the list, the second one inserts a node after a node with a given location, and the third one inserts a node into the sorted list.

#### Inserting at the Beginning of the List:-

Suppose our linked list is not necessarily sorted and there is no reason to insert a new node in any special place in the list. Then the easiest place to insert the node is at the beginning of the list. An algorithm that does so follows.

Algorithm:-

INSFIRST(INFO, LINK, START, AVAIL, ITEM)

This algorithm inserts ITEM as the first node in the list.

- 1.[OVERFLOW?] If AVAIL=NULL then write OVERFLOW and Exit.
2. [Remove first node from AVAIL list.] Set NEW=AVAIL and  
AVAIL=LINK[AVAIL]
- 3.Set INFO[NEW]=ITEM [Copies new data into new node]

4. Set  $\text{LINK}[\text{NEW}] = \text{START}$  [New node now points to the original first node]

5.Set START=NEW [Change START so it points to the new node ]

6.Exit.

Inserting after a Given Node:- Algorithm:-

INSLOC(INFO,;INK,START,AVAIL,LOC,ITEM)

This algorithm inserts ITEM so that item follows the node with location LOC or inserts ITEM as a first node when LOC=NULL.

1.[OVERFLOW?] If AVAIL=NULL then write OVERFLOW and Exit.

2.[Remove First Node from the AVAIL list].

Set NEW=AVAIL and AVAIL=LINK[AVAIL].

3.Set INFO[NEW] =ITEM. [Copies new data into new node.]

4.If LOC=NULL then [Inserts as first node] Set  
LINK[NEW]=START and START=NEW. else [Insert  
after node with location LOC]

Set LINK[NEW]=LINK[LOC] and LINK[LOC]=NEW. [End of If  
Structure]

5.Exit.

Inserting into a Sorted Linked List:-

Suppose ITEM is to be inserted into a sorted linked list. Then ITEM must be inserted between nodes A and B so that

$$INFO(A) < ITEM < INFO(B)$$

The following is the procedure which finds the location LOC of node A that is which finds the location LOC of the last node in the list whose value is less than ITEM. Traverse the list using pointer variables PTR and comparing ITEM with INFO[PTR] at every node. While traversing keep track of the location of the preceding node by using a pointer variable SAVE. Thus SAVE and PTR are updated by assignments.

Algorithm:-

FINDA(INFO, LINK START, ITEM,LOC)

This procedure finds the location LOC of the last node in a sorted list such that  
INFO[LOC] < ITEM or set LOC = NULL.

1.[List Empty?] If START = NULL then set LOC= NULL and Return.

2.[Special Case?] If ITEM<INFO[START] then Set LOC =NULL and Return.

3. Set  $SAVE = START$  and  $PTR = LINK[START]$  [Initialize Pointers]
4. Repeat Step 5 and 6 while  $PTR \neq NULL$ .
5. If  $ITEM < INFO[PTR]$  then  
     Set  $LOC = SAVE$  and Return. [End of If Structure]
6. Set  $SAVE = PTR$  and  $PTR = LINK[PTR]$  [Update Pointers] [End of Step 4 Loop]
7. Set  $LOC = SAVE$ .
8. Exit.

Now we have all the components to present an algorithm which inserts ITEM into a linked list. The simplicity of the algorithm comes from using the previous two procedures.

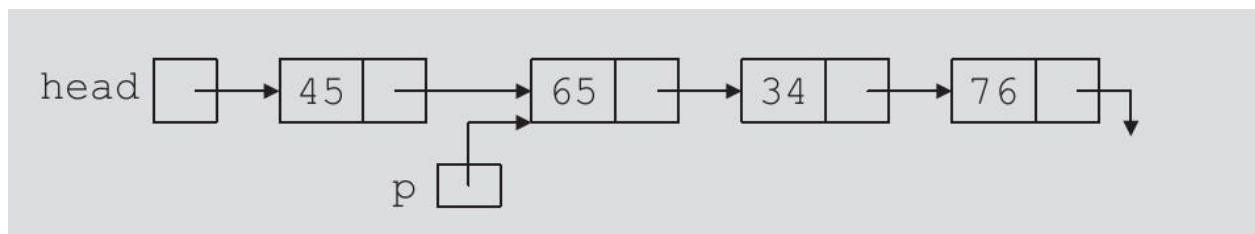
Algorithm:-

INSERT(INFO, LINK, START, AVAIL, ITEM)

This algorithm inserts ITEM into a sorted linked list.

1. Call FINDA(INFO, LINK, START, AVAIL, ITEM)
2. Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM).
3. Exit.

Consider the linked list shown in Figure 6.



Suppose that **p** points to the node with **info 65**, and a new node with **info 50** is to be created and inserted after **p**. Consider the following statements:

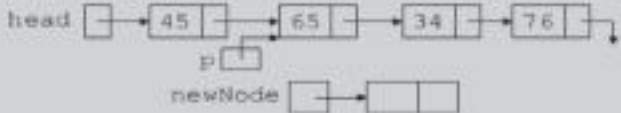
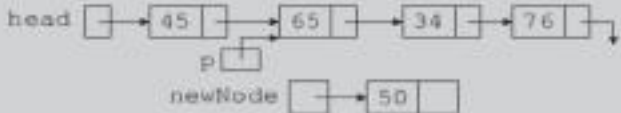
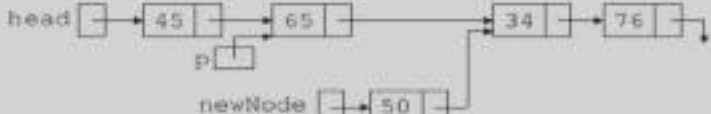

**newNode=new nodeType; //create new Node newNode-**

**>info=50; //store 50 in the newnode newNode->link=p-**

**>link;**

**p->link=newNode;**

Table 3 shows the effect of these statements. Table 3: Inserting a node in a linked list

Statement	Effect
<code>newNode = new nodeType;</code>	
<code>newNode-&gt;info = 50;</code>	
<code>newNode-&gt;link = p-&gt;link;</code>	
<code>p-&gt;link = newNode;</code>	

Note that the sequence of statements to insert the node, that is,

**`newNode->link = p->link;`**  
**`p->link = newNode;`**

is very important because to insert **`newNode`** in the list we use only one pointer, **`p`**, to adjust the links of the nodes of the linked list. Suppose that we reverse the sequence of the statements and execute the statements in the following order:

**`p->link = newNode;`**  
**`newNode->link = p->link;`**

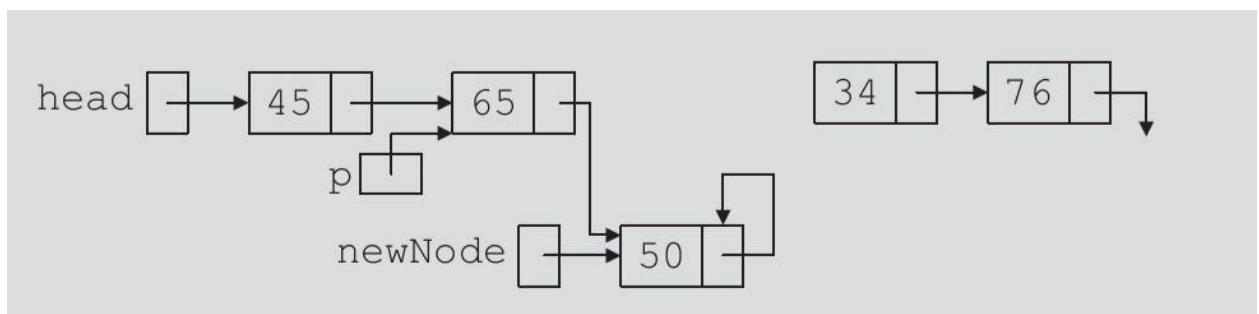


Figure: List after the execution of the statement **`p->link = newNode;`** followed by the execution of the statement **`newNode->link = p->link;`**

From Figure, it is clear that **`newNode`** points back to itself and the remainder of the list is lost.

## Conclusion

---

---

---

---

—

---

(Concerned Teacher/Lab  
Engineer)

## Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
struct node
    {
        int data;
        node *prev, *next;
    }*q, *temp, *start=NULL;
int c1, c2 ;
void create();
void display();
void insert();
void del();
int main()
```

```

{
    system("cls");
    while(1)
    {
        system("cls");
        cout <<"press 1 for adding data \n";
        cout <<"press 2 for displaying data \n";
        cout <<"press 3 for insertion \n";
        cout <<"press 4 for deletion \n";
        cout <<"press 0 for exit\n";
        char ch;
        ch=getch();
        switch(ch)
        {
            case '1':
                system("cls");
                create();
                break;

            case '2':
                system("cls");
                display();
                getch();
                break;

            case '3':
                system("cls");
                insert();
                break;

            case '4':
                system("cls");
                del();
                break;

            case '0':
                exit(1);
        }
    }
}

void create()
{
    temp = new node;
    temp -> next = NULL;
    cout <<"\nEnter data\n";
    cin >> temp -> data ;
    if(start == NULL)
    {
        start = temp;
        temp -> prev = NULL;
    }
    else
    {
        q= start;
        while(q->next != NULL)
        {
            q = q->next;
        }
    }
}

```



```

        }
        q->next = temp;
        temp->prev = q;
    }
}
void display()
{
    q=start;
    while(q!=NULL)
    {
        cout<<q->data<<endl;
        q = q->next;
    }
}
void insert()
{
    cout <<"Press 1 for insertion at start\n";
    cout <<"Press 2 for insertion at middle\n";
    cout <<"Press 3 for insertion at end\n";
    int choice;
    cin>>choice;
    switch(choice)
    {
        case 1:
            system("cls");
            temp = new node;
            cout<<"Enter data \n";
            cin>>temp->data;
            start->prev =temp;
            temp->next = start;
            start =  temp;
            temp -> prev = NULL;
            break;

        case 2:
            system("cls");
            cout<<"Enter the data after which u want to add
this\n";

            int ch;
            cin>>ch;
            q= start;
            while(q->next!=NULL)
            {
                if(q->data == ch)
                {
                    temp = new node;
                    cout<<"Enter data \n";
                    cin>>temp->data;
                    q->next->prev = temp;
                    temp->next = q->next;
                    temp->prev = q;
                    q->next = temp;
                }
                q = q->next;
            }
        }
    }
}

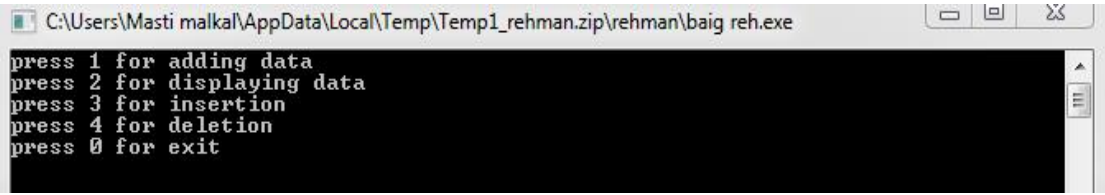
```

```

        }
        break;
    case 3:
        system("cls");
        temp = new node;
        cout<<"Enter data\n";
        cin>> temp->data;
        temp->next = NULL;
        q = start;
        while(q->next != NULL)
        {
            q= q->next;
        }
        q->next = temp;
        temp->prev = NULL;
    }
}
void del()
{
    system("cls");
    cout<<"Enter the data you want to delete \n";
    int num;
    cin>>num;
    q = start;
    if (start->data == num)
        start = start -> next;
    else
    {
        while(q != NULL)
        {
            if(q->next->data == num)
            {
                temp = q->next;
                q->next = temp->next;
                temp->next->prev = q;
                delete temp;
            }
            q = q->next;
        }
    }
}
}

```

## Output:



```

C:\Users\Masti malkal\AppData\Local\Temp\Temp1_rehman.zip\rehman\baig reh.exe
press 1 for adding data
press 2 for displaying data
press 3 for insertion
press 4 for deletion
press 0 for exit

```

```
C:\Users\Masti malkal\AppData\Local\Temp\Temp1_rehman.zip\rehman\baig reh.exe
1
2
3
4
5
```

```
C:\Users\Masti malkal\AppData\Local\Temp\Temp1_rehman.zip\rehman\baig reh.exe
Press 1 for insertion at start
Press 2 for insertion at middle
Press 3 for insertion at end
```

```
C:\Users\Masti malkal\AppData\Local\Temp\Temp1_rehman.zip\rehman\baig reh.exe
Enter the data you want to delete
6_
```

```
C:\Users\Masti malkal\AppData\Local\Temp\Temp1_rehman.zip\rehman\baig reh.exe
1
2
3
4
5
6
```

```
C:\Users\Masti malkal\AppData\Local\Temp\Temp1_rehman.zip\rehman\baig reh.exe
1
2
3
4
5
```