Experiment # 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:-

1. Code Blocks with GCC compiler.

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 LINK[NEW]=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 ≠ 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.

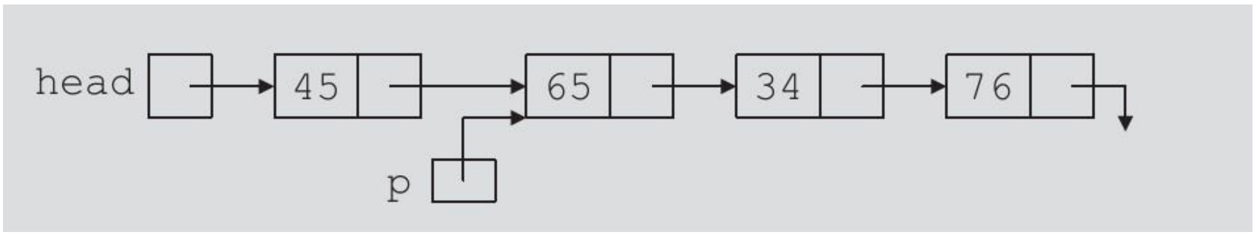
Algorithm:-

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:

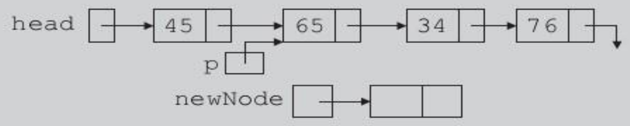
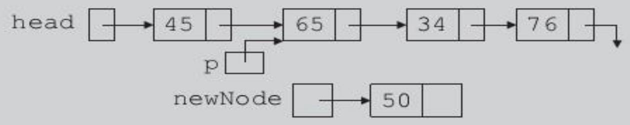
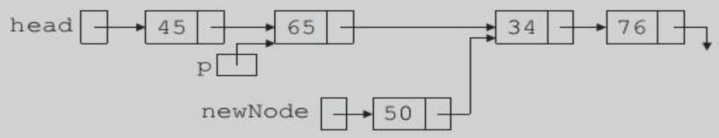
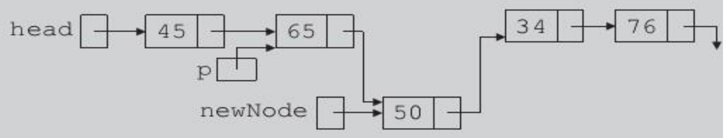
```
newNode=new nodeType; //create new Node newNode-

>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
`newNode = new nodeType;`	
`newNode->info = 50;`	
`newNode->link = p->link;`	
`p->link = newNode;`	

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;

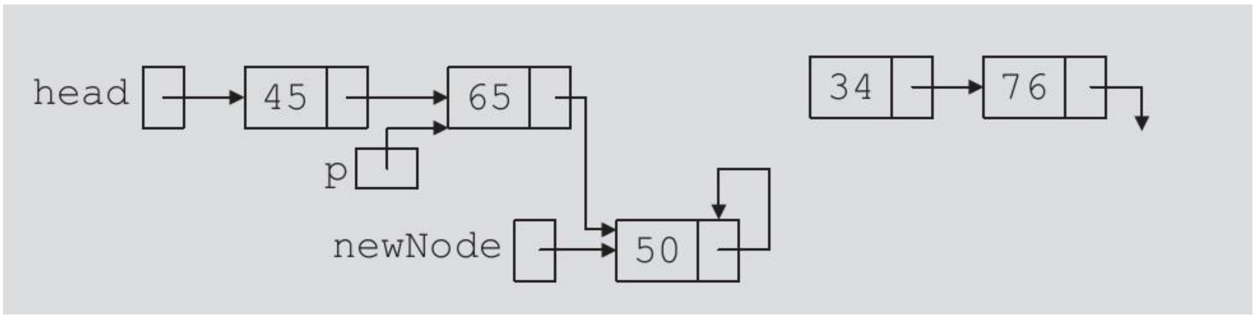


Figure: List after the execution of the statement **p->link = newNode;**followed by the execution of the statement **newNode->link = p->link;**

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

Lab Task:-

Write a C++ code using functions for the following operations.

1.Creating a linked List.
2.Traversing a Linked List.
3.Inserting the node at the start of the list.
4.Inserting a node after a given node.
5.Inserting a node in a sorted list.

Create a complete menu for the above options and also create option for reusing it.

Conclusion

(Concerned Teacher/Lab Engineer)