

Lecture 6

In this lecture, we will study about

1. Insertion in a Linked List
 1. At the beginning
 2. At the end
 3. Anywhere in between
2. Deletion in a Linked List
 1. First Node
 2. Last Node
 3. Any node

Insert A node at the beginning

We have a linked list, head points to the first node (Or head stores the address of first node). We want to insert a node at the beginning, i.e. Insert a node before the first node.

See the flash presentation first. Since we want to insert a node at the beginning, that means we need to do the following ((뒤따르는) 다음의, 다음에 계속되는) 2 things,

1. head will contain a new address (of new node).
2. And new node will point to the existing first node or new node's link will contain the address of existing first node.

Let us start from the beginning.

Step 1. Since we want to insert a new node, we need to create a new node. Call this node as temp.

```
temp = (node *) malloc (sizeof(node));
```

Step2. Ask the user to enter data.

```
printf("Enter data");  
scanf("%d", temp->data);
```

Step3: We want to insert temp in the beginning, that means we want to see temp as the first node. If temp is the first node, then What will be stored in temp's link?

temp's link will contain the address of current first node.
That means,

```
temp -> link = head; (because head contains the address of first node.)
```

Step4: Half of the job is finished. Now we only want head to contain the address of first node, or we want head to point to the first node.

```
head = temp;
```

Complete Program

```
temp = (node *) malloc(sizeof(node));
printf("Enter data");
scanf("temp->data");

temp->link = head;
head = temp;
//print link list
while (temp != null)
{
    printf("%d", temp->data, "->");
    temp = temp->link; // move temp to next node.
}
```

Source Code for Insertion in the Beginning

Insert A new node after the last node

We have a linked list, head points to the first node (Or head contains the address of first node). We want to insert a node after the last node.

Since we want to insert a new node after the last node. We need to do following ((**뒤따르는**) **다음의**, **다음에 계속되는**) things.

First, we want temp1(temp1 is a pointer of type node, i.e. node * temp1) to point to the last node.

Step 1 : Make temp point to the first node.

```
temp = head;
```

Step 2 : Move temp until it reaches last node.

```
While (temp -> link!= null)
    temp = temp -> link;
```

Now temp points to the last node.

Step 3 : Create a new node. Call this node as temp1.

```
temp1 = (node *) malloc(sizeof(node));
```

Step 4: Ask the user to enter value.

```
printf ("Enter value");
scanf("%d", temp1->data);
```

Step 5 : Since temp1 is going to be last node, hence temp1's link should contain null.

```
temp1->link = null;
```

Step 6 : To make temp1 as a last node, make temp's link point to temp1 or store address of temp1 in temp's link.

```
temp->link = temp1;
```

Complete code

```
// Move temp to the last node
temp = head;
While (temp -> link!= null)
    temp = temp -> link;

// Create a new node
temp1 = (node *) malloc(sizeof(node));
printf ("Enter value");
scanf("%d", temp1->data);

// new node link contains null
temp1->link = null;

// temp link stores the address of temp1
temp->link = temp1;

//print link list
```

```
temp = head;
while (temp != null)
{
    printf("%d", temp->data, "->");
    temp = temp->link; // move temp to next node.
}
```

Insert At the end of Linked List

Insert a node anywhere in between link list

In this case, before inserting a node, we need to ask user, where he want to insert.

There are 2 ways of asking the user

1. We can ask the user after which node he want to insert (3rd node or 4th node)
2. We can ask the user to enter the value of node after which he want to insert.

After we know where user want to insert, we will move temp to that node. Let us do this first.

Step1 : Ask the user, to enter value of the node, after which he want to insert.

```
printf("Enter value of node after which you want to insert");
scanf(val);
```

We want temp to point to the node whose value user gave in above statement.

Step2 : Make temp point to head and move temp to the node whose value user gave above.

```
temp = head
while (temp->data != val)
    temp = temp -> link;
```

Step3 : Now, temp points to the node after which we want to insert a new node.

Create a new node. Call this node as temp1.

```
temp1 = (node *) malloc(sizeof(node));
```

Step4 : Ask the user to enter data.

```
printf("Enter data");  
scanf("%d", temp1->data);
```

Step5 : We want to insert temp1 after temp. Make f point to the node just (바로, 틀림없이, 마침, 꼭) next ((시간이)a [관사 없이] (현재를 기준으로 하여)) to temp. Or f contains the address of node just next to temp.

```
f = temp -> link;
```

Step6 : Now, we want to insert temp1 in between temp and f. That means, temp1's link will contain the address of f.

```
temp1->link = f;
```

Step7 : And, temp's link will contain the address of temp1.

```
temp->link = temp1;
```

Here is the complete code

```
printf("Enter value of node after which you want to insert");  
scanf(val);  
  
/*  
Make temp point to first node and then move temp to node whose  
data is val  
*/  
temp = head  
while (temp->data != val)  
temp = temp -> link;  
  
// Create a new node, ask the user to enter value  
temp1 = (node *) malloc(sizeof(node));  
printf("Enter data");  
scanf("%d", temp1->data);  
  
// f points to the node just next to temp  
f = temp->link;  
  
// temp1's link stores the address of f.  
temp1->link = f;
```

```

// temp's link stores the address of temp1.
temp->link = temp1;

//print link list
temp = head;
while (temp != null)
{
    printf("%d", temp->data,"->");
    temp = temp->link; // move temp to next node.
}

```

Delete the first node of Link list

If we want to delete the first node, then head should point to second node (because after deletion of first node, second node will become the first node).

Step1: Make temp point to the first node

```
temp = head
```

Step2: Move head to the second node.

```
head = head -> link;
```

Step3: Make temp's link point to null or store null.

```
temp->link = null;
```

Step4: Delete first node or temp.

```
delete (temp);
```

Complete code:

```

// Make temp points to the first node.
temp = head;

// move head to the second node.
head = head -> link;

// first node (temp) link points to null or stores null

```

```

        temp -> link = null;

// delete first node or temp.
        delete(temp);

//print link list
        temp = head;
        while (temp != null)
        {
                printf("%d", temp->data,"->");
                temp = temp->link; // move temp to next node.
        }

```

Delete last node from Link List

If we want to delete the last node, then second last node will become the last node. This means, second last node link should contain null after deletion of last node.

We need 2 variables, sl and l.

'l' will point to the last node and 'sl' will point to the second last node.

Step1: To move 'sl' to the second last node, we will first point 'sl' to the first node and then move 'sl' to the second last node.

```

        sl=head;
        while (sl ->link->link != null)
                sl = sl-> link;

```

Step2 : Make 'l' point to the last node

```

        l = sl -> link;

```

Step3 : Since sl will be last node, after deletion of last node, that means, sl's link should contain null.

```

        sl -> link= null;

```

Step 4: delete last node.

```

        delete (l).

```

Here is the complete program

```

// Make sl point to the first node
    sl=head;

//Move sl to the second last node.
    while (sl ->link->link != null)
        sl = sl-> link;

// Make l point to the last node
    l = sl -> link;

// sl's line stores null.
    sl -> link= null;

//delete last node
    delete (l).

//print link list
    temp = head;
    while (temp != null)
    {
        printf("%d", temp->data,"->");
        temp = temp->link; // move temp to next node.
    }

```

Delete a node in between.

If we want to delete a node in between, we need to ask user, to enter value of the node which he want to delete.

Then move temp to that node.

Step 1: Ask the user to enter value of the node, which he want to delete

```

printf("Enter the value of node that you wish to delete");
scanf("%d", val);

```

Step 2: Make temp point to first node and move temp to the node whose value user gave in step 1.

```

temp = head;
while (temp->data != val)
    temp = temp -> link;

```


Step 3: Now, temp points to the node, which we want to delete. Make 'prev' point to the node behind temp.

```
prev = head;
while (prev->link != temp)
    prev = prev->link;
```

Step 3: Now temp points to the node which user wish to delete. And prev points to the node behind temp.

Make temp1 point to the next node after temp.

```
temp1 = temp->link;
```

Step 4: If we want to delete temp, then prev's link should contains the address of temp1.

```
prev->link = temp1;
```

Step 5: And now, store null in temp's link.

```
temp->link = null
```

Step 6: delete temp.

```
delete(temp);
```

Here is the complete code

```
// ask the user to enter value of node which he wish to delete
printf("Enter value of the node that you wish to delete");
scanf("%d", &val);

// Make temp points to the first node.
temp=head;

/*
Make temp point to the node which user want to delete.
*/
temp = head;
```

```

        while (temp->data != val)
            temp = temp -> link;

/*
Make prev point to the node before temp.
*/

    prev = head;
    while (prev -> link != temp)
        prev = prev->link;

// temp1 points to the node after temp(node that user want to delete).
temp1 = temp->link;

// prev's link stores the address of temp1
prev->link = temp1;

// Since we are going to delete temp, so temp's link stores null now.
temp->link = null;

// finally, delete temp.
delete(temp);

//print link list
temp = head;
while (temp != null)
{
    printf("%d", temp->data,"->");
    temp = temp->link; // move temp to next node.
}

```