# Lecture 4- Linked List(2)

### Insert data after a given node of Linked List

- We have to insert a new node after a given node.
- We will set the next of new node to the next of given node.
- Then we will set the next of given node to new node

So the method for singly Linked List will look like this,

```
void InsertAfter(Node prev_node, int new_data)
{
    if (prev_node == null) {
        Console.WriteLine("The given previous node Cannot be null");
        return;
    }
    Node new_node = new Node(new_data);
    new_node.next = prev_node.next;
    prev_node.next = new_node;
}
```

To perform this operation on doubly linked list we need to follow two extra steps

1. Set the previous of new node to given node.
2. Set the previous of the next node of given node to the new node.

So, the method for Doubly Linked List will look like this.

```
void InsertAfter(DNode prev_node, int data)
```

```
{
    if (prev_node == null) {
        Console.WriteLine("The given prevoius node cannot be null");
        return;
    }
    DNode newNode = new DNode(data);
    newNode.next = prev_node.next;
    prev_node.next = newNode;
    newNode.prev = prev_node;
    if (newNode.next != null) {
        newNode.next.prev = newNode;
    }
}
```

**Delete a node from Linked List using a given key value**

- First step is to find the node having the key value.
- We will traverse through the Linked list, and use one extra pointer to keep track of the previous node while traversing the linked list.
- If the node to be deleted is the first node, then simply set the Next pointer of the Head to point to the next element from the Node to be deleted.
- If the node is in the middle somewhere, then find the node before it, and make the Node before it point to the Node next to it.
- If the node to be deleted is last node, then find the node before it, and set it to point to null.

So, the method for singly linked list will look like this,

```
void DeleteNodebyKey(int key)
```

```
{
    Node temp =  head;
    Node prev = null;
    if (temp != null && temp.data == key) {
                head = temp.next;
        return;
    }
    while (temp != null && temp.data != key) {
        prev = temp;
        temp = temp.next;
    }
    if (temp == null) {
        return;
    }
    prev.next = temp.next;
}
```

To perform this operation on doubly linked list we don't need any extra pointer for previous node as Doubly linked list already have a pointer to previous node.so the delete method will be,

```
void DeleteNodebyKey(DoubleLinkedList doubleLinkedList, int key)
{
    DNode temp = doubleLinkedList.head;
```

```
    if (temp != null && temp.data == key) {
        doubleLinkedList.head = temp.next;
        doubleLinkedList.head.prev = null;
        return;
    }
    while (temp != null && temp.data != key) {
        temp = temp.next;
    }
    if (temp == null) {
        return;
    }
    if (temp.next != null) {
        temp.next.prev = temp.prev;
    }
    if (temp.prev != null) {
        temp.prev.next = temp.next;
    }
}
```

**Reverse a Singly Linked list**

This is one of the most famous interview questions. We need to reverse the links of each node to point to its previous node, and the last node should be the head node.This can be achieved by iterative as well as recursive methods. Here I am explaining the iterative method.

- We need two extra pointers to keep track of previous and next node, initialize them to null.
- Start traversing the list from head node to last node and reverse the pointer of one node in each iteration.
- Once the list is exhausted, set last node as head node.

The method will look like this,

```
public void ReverseLinkedList(SingleLinkedList singlyList)
{
    Node prev = null;
    Node current = singlyList.head;
    Node temp = null;
    while (current != null) {
        temp = current.next;
        current.next = prev;
        prev = current;
        current = temp;
    }
    singlyList.head = prev;
}
```