Link List

A **link list** is one of the fundamental data structures and can be used to implement other data structures. It consists of a sequence of nodes, each containing (one or more as per requirement) data fields and one (or two) references (two for doubly link list) pointing to the next (or previous) nodes. We can use Link lists when we don't know actual number of data. Link lists permit insertion and removal of nodes at any point in the list.

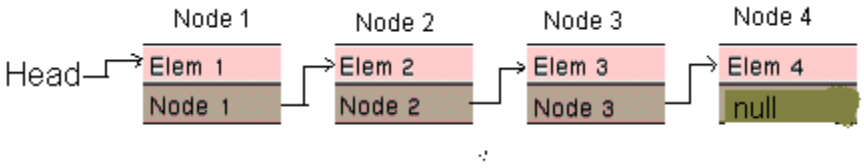
Field in node for Singly Link list:

-> Data field: Data field of link list node contains useful information,

-> Reference(Link): Reference or Link field of node contains Reference of the of the other node.

Singly Link list

A singly link list's node is divided into two field. The first field holds information about the node, and second field holds the reference of next node. When we link all node is some order then it becomes **Singly Link list**. In singly link list we have only one reference of next node, so in singly link list we can travels in only one way. The reference (link) points to the next node in the list, or to a null value(if its last node).For Singly Link list we need one Head reference to point to the First node of the list.



Program For Link list:

```
Class Node{
        Item item;
        Node next;          //reference to next node;


    Node()
    {
        item =  null;
        next = null;
```

```
        }

        Node(Item t)   // constructor for set item value and set next reference to null
        {
                item = t;
                next = null;
        }

        Node( Item t, Node n)   // constructor to set value and set next reference to given n
        {
                item = t;
                next = n;
        }

        Node ( Node n)
        {
                item = n.item;
                next = n.next;
        }


}

Class List
{
        Node head = null;

        void addFront(Item t)
        {
                Node n = new Node(t);                   // line 1
                n.next = ead;                           // line 2
                head = n;                               // line 3
        }
                // line 1 line 2 and line 3 can be replace by head = new Node(t,head);

        void addRear(Item i)
        {
                Node temp;

                if(head ==null)                         //  need to test when list is empty
                {
                        addFront(t);
                        return;
                }
                while( temp.next  != null)              // need to Locate  up to last node
                {
```

```
                temp = temp.next'
            }

            temp.next = new Node(t);
        }



        void addMiddle(Item t , Node pred)
        {
            pred.next = new Node (t , pred.next);
        }

        void delAfter(Node n)
        {
            //line 1
                n.next = n.next.next;
        }
            // check line 1 we need to take care for two case  one when list n is null
            // and n.next = null that is n is last node
            // so put line 1 ==   if(n == null || n.next == null)
}
```

In Link list We can Insert new Node at First position , at Last position or in middle of the list at given position. Let's assume that no. of node in Link list is N

So we need three different Function to insert node at different position.

Functions:

void addFront()

Function adds node to the first position in the list. Takes O(1) constant time to set next of new node to head and head to new node.

Void addRear()

Function adds node to the last position in the list. Takes O(N) time because we need to locate last node to insert. And in link list we don't have reference to last node so we need to traverse list.

Void addMiddle() :

Function adds node after given node. Takes O(1) time , because we have given reference to the node after which we have insert node.

In this way we can use Singly Link list, when number of node is not know priori.

But in Singly Link list, we can traverse List in only one direction. If we want to insert node before given node(p) instead of after given node (as we did in addMiddle() function) then it takes O(N) time to locate one previous node of the node p because we need to traverse List up to previous node of p.

Anyhow If we have reference to the previous node then we can use previous reference to locate previous node and Then insertion of node before give node will take O(1) time. In this way we can traverse List in both directions.

That kind of Link list in which we can traverse List in Both directions is known as Doubly Link list

Field in Node for Doubly Link List:

-> Data field: Data field of link list node contains useful information,

-> Next Reference(Link): Reference or Link field of node contains Reference of the next node in list.

-> Previous Reference(Link): Reference or Link field of node contains Reference of the previous node in list.

In both, Singly Link list and Doubly Link list, Head points to the first node, And next of the Last node contains null reference. In this case If we are at the last position and we want to traverse List from the first position then we have to use head. Instead of storing null in the next field of the last node we can reference to the first node. So from last node of list we can access first node of list using next of the last node.

This Kind of list, in which we can traverse List in Circular way, is known as **Circular Link list**. If Link list is Singly Link list then it know as **Circular Singly Link list.**

But in case of Doubly Link list circularity should hold for both directions. So next of last node points to the first node of list and previous of the first node points to the last node of list and It's called as **Doubly Circular Link list**.