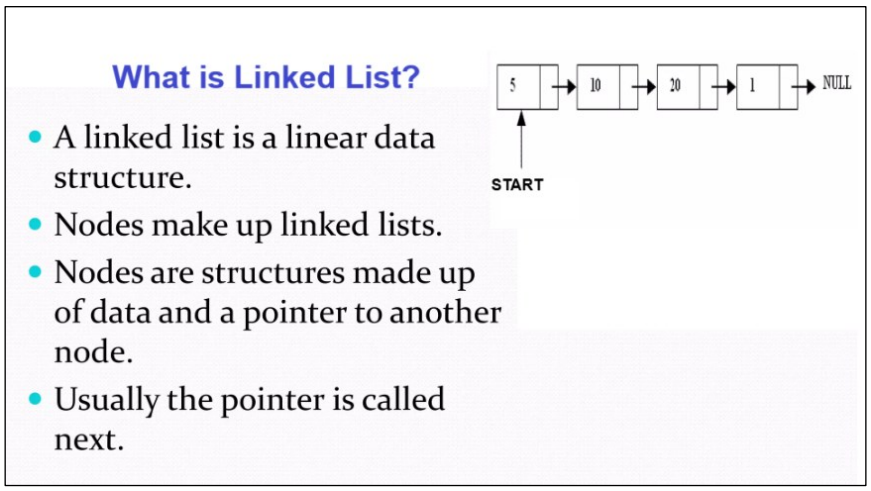
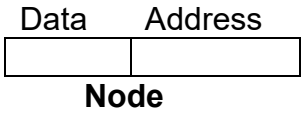
Linked List

A linked list is a chain of structures where each structure contains data as well as a pointer to the next structure. Linked lists are very useful and are very common data structures. They can be either ordered or unordered.

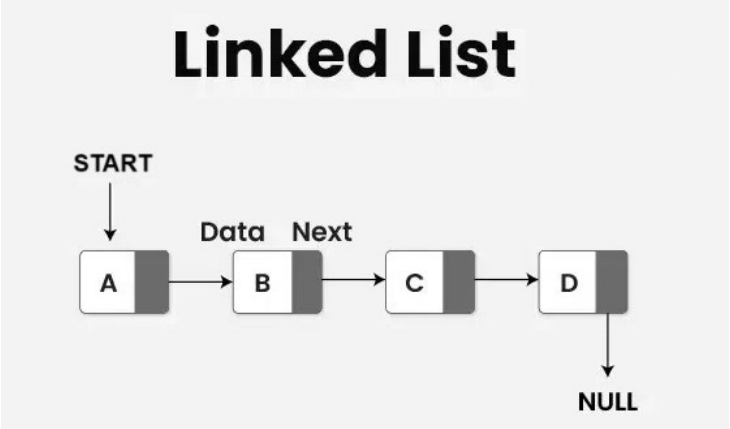


1. Structure of a Linked List

A linked list consists of a number of similar types of nodes, where each node can hold data as well as a pointer to the next node. Given below the structure of a node:



As a node is created using structure or object, such types of structures are also called **self-referential** structures as they can point to themselves or another structure of similar type.



2. Advantages of Linked List (over Array)

1. It is not necessary to know the number of elements and allocate memory for the linked list. Memory can be allocated as and when required at run time.
2. Inserting to and deleting from linked lists can be handled efficiently without having to restructure the list.

3. Disadvantages of Linked List

1. Randomly accessing is not possible.
2. Extra memory for a pointer is needed with every element in the list.

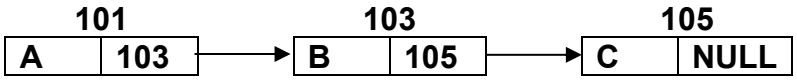
4. Linked List vs. Array

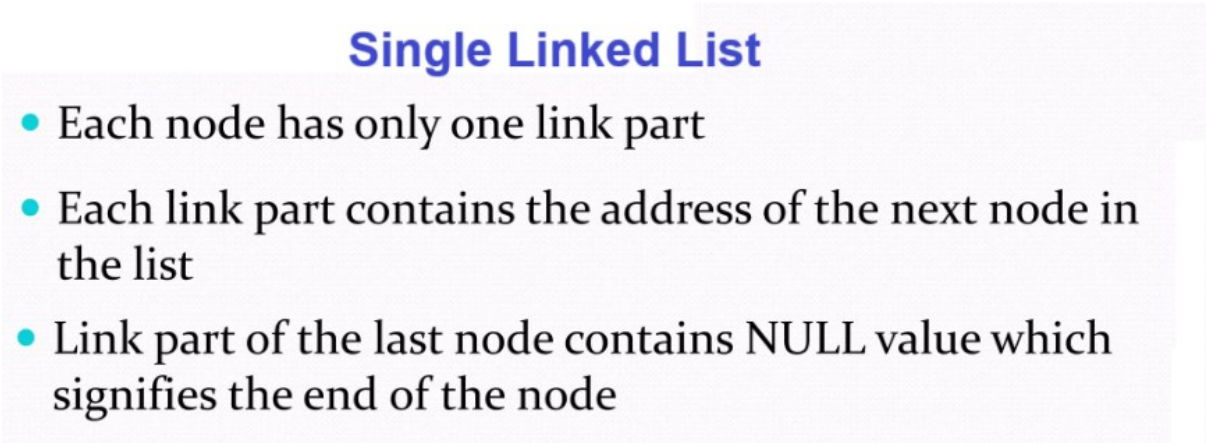
Arrays	Linked list
Fixed size: Resizing is expensive	Dynamic size
Insertions and Deletions are inefficient: Elements are usually shifted	Insertions and Deletions are efficient: No shifting
Random access i.e., efficient indexing	No random access → Not suitable for operations requiring accessing elements by index such as sorting
No memory waste if the array is full or almost full; otherwise may result in much memory waste.	Since memory is allocated dynamically(acc. to our need) there is no waste of memory.
Sequential access is faster [Reason: Elements in contiguous memory locations]	Sequential access is slow [Reason: Elements not in contiguous memory locations]

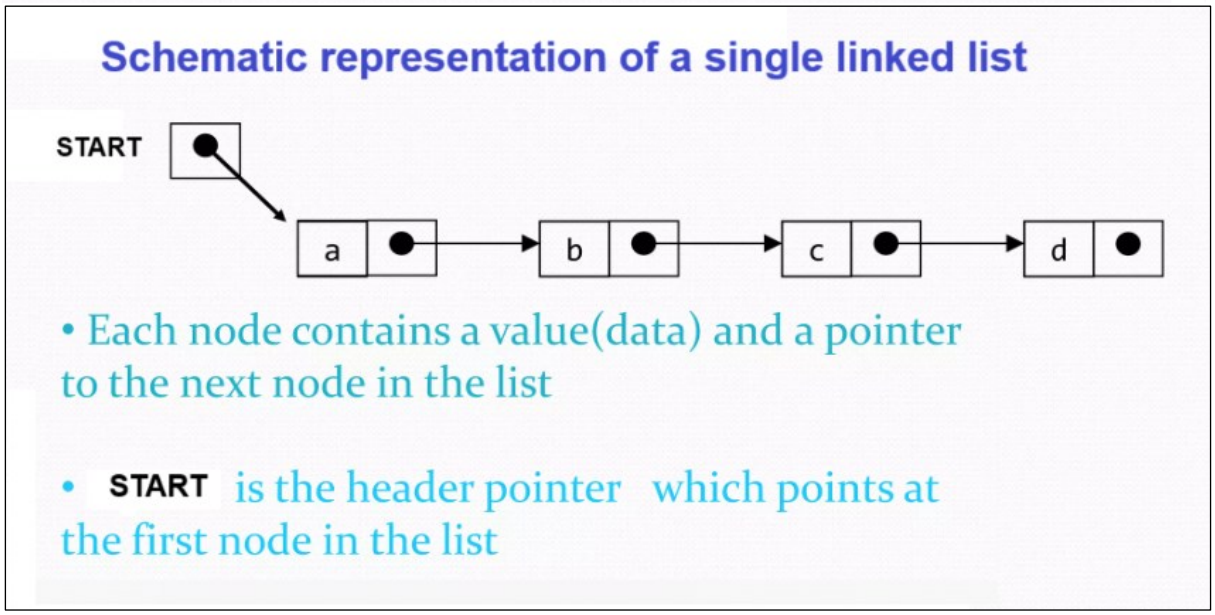
5. Different Types Of Linked Lists

There are various types of linked lists:-

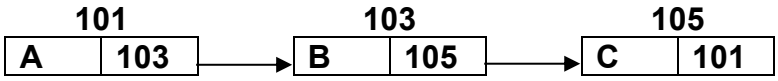
i. **Single Linked List -** also called singly linked list or one-way list, is defined as a collection of elements, called nodes, such that each node has at least one information field (data) and only one pointer field (link). This link field contains the memory address of the next node. Given below is the structure of a linear linked list holding 3 alphabets as data values.



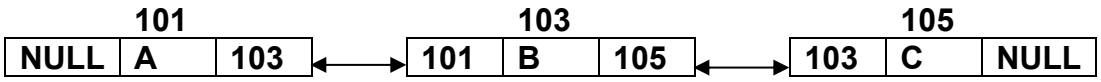




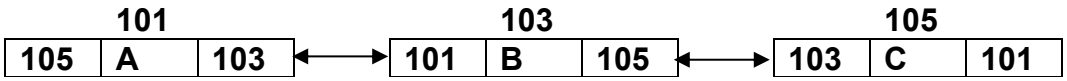
ii. **Circular Linked List** - in the case of a circular linked list, the null pointer in the last node of a list is replaced with the address of its first node i.e. link field of the last node contains a pointer back to the first node. Thus, in a circular linked list if we are at any given node and traverse the entire list then we ultimately end up at the starting point. Given below is the structure of a circular linked list.

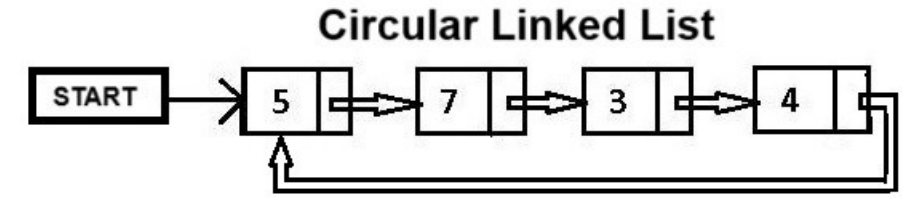


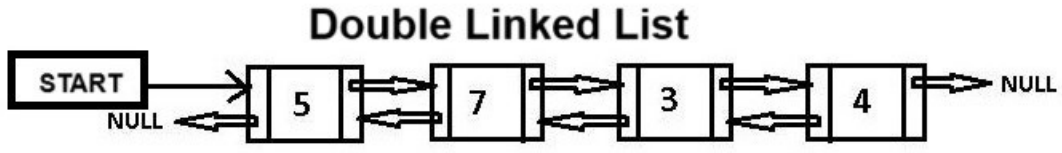
iii. **Doubly Linked List** - in a doubly linked list each node contains two links, one to its successor and one to its predecessor i.e. the address of both the next node and the previous node. Thus, in a doubly linked list, one can traverse the list in both forward and backward directions. In a doubly linked list, each node has one information field (data) and two pointer fields (linkprev and linknext). Given below is the structure of a doubly linked list.

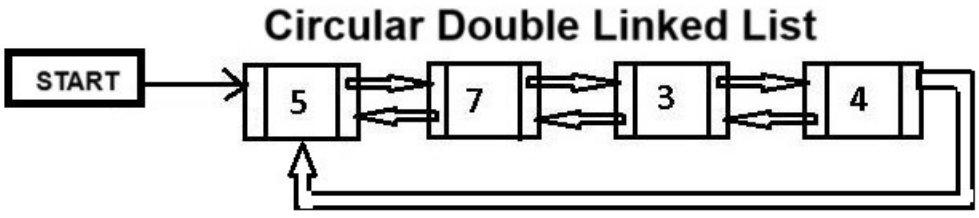


iv. **Circular Doubly Linked List** - in the case of a circular doubly linked list, the null pointer in the last node of a list is replaced with the address of its first node i.e. linknext field of the last node contains a pointer back to the first node and the null pointer in the first node of a list is replaced with the address of its last node i.e. linkprev field of a first node contain a pointer back to the last node.







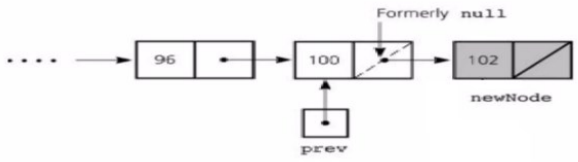


6. Various operations on the Linked List

1. Insertion in the list

a. Insertion at the end (Append)

Here we simply need to make the next pointer of the last node point to the new node



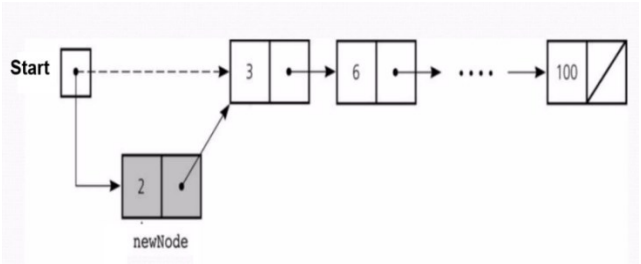
b. Insertion at the front

There are two steps to be followed:-

a) Make the next pointer of the node point towards the first node of the list

b) Make the start pointer point towards this new node

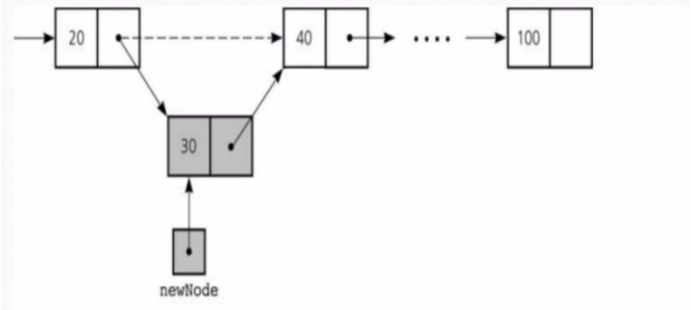
- If the list is empty simply make the start pointer point towards the new node;



c. Insertion at any position P

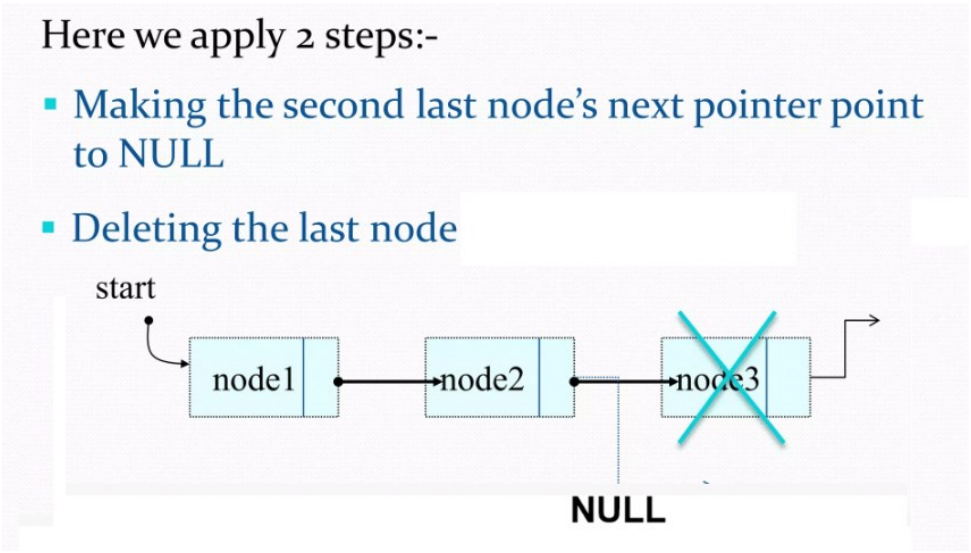
Here we again need to do 2 steps :-

- Make the next pointer of the node to be inserted point to the next node of the node after which you want to insert the node
- Make the next pointer of the node after which the node is to be inserted, point to the node to be inserted

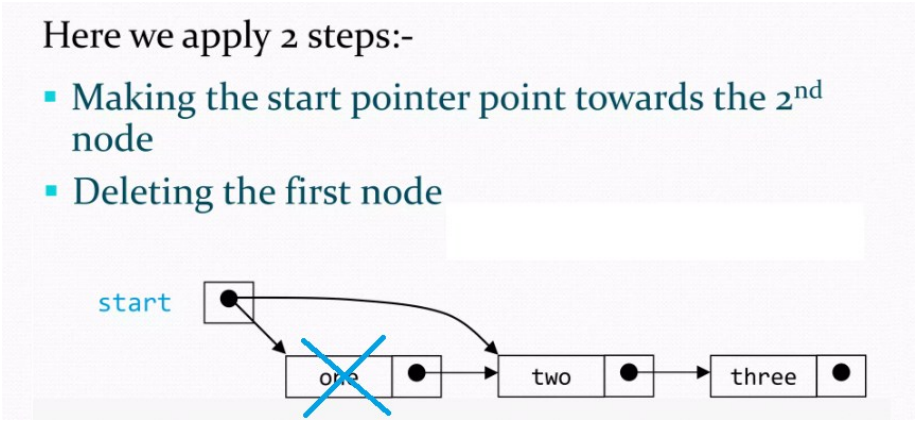


2. Deletion from the list

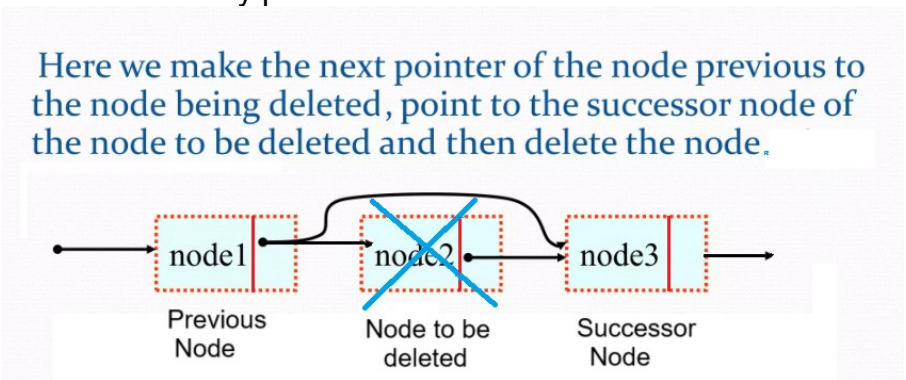
a. Deletion from the end



b. Deletion from the front



c. Deletion from any position P



3. Display of the list

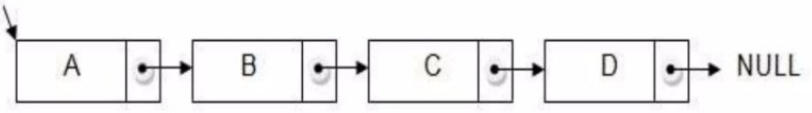
Here we traverse the list from the first node to the last node and print the data stored in each node until we reach NULL.

4. Searching in the list

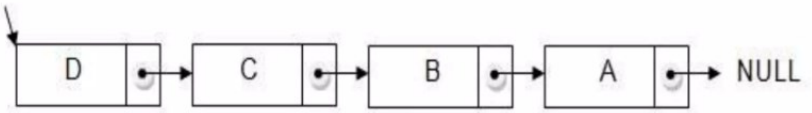
Here we traverse the list from the first node to the last node and check the data stored in each node with the search element, if found print it else traverse until we reach NULL.

5. Reverse of the list

- We can reverse a linked list by reversing the direction of the links between 2 nodes



Input



Output

6. Sorting of the list

a. By rearranging the data in the nodes (Sort by value)
   Here we swap the data of two nodes if they are not in the proper order until the entire list gets sorted.

b. By rearranging the nodes itself (Sort by memory address)
   Here we swap the two nodes, by changing the address, if they are not in proper order until the entire list gets sorted.

7. Merging of 2 linked lists into one linked list

Here we traverse the First list from the first node to the last node. Make the next pointer of the last node point to the first node of the Second list.

8. Splitting of one linked list into 2 separate linked lists

Here we traverse the linked list until the position P reaches, We make the next pointer of the node previous to position P points to NULL. Create a new Head for the list from the node at position P.

The following class will create a node (self-referential structure) to hold the data in the linked list

```
class Node
{
   int data;                  // variable to hold the data in the node
   Node next;                 // variable to hold the address of another node
   public Node(int n)         // parameterized constructor to initialize a node
   {
      data=n;
      next=null;
   }
}
```

The following class will create a linked list of n nodes depending upon the entries from the user

```
class LinkedList
{
   public Node start; //Node start will hold the address of the first node of the linked list
   public LinkedList( )    //default constructor initializing the first node with null
   {
      start=null;
   }

   public void appendNode(Node start, int s) //to add an item at the end of the list
   {
      Node n1=new Node(s);     //new Node created
      if(start==null)          //as no linked list exists
      {
         start=n1;             //new Node becomes the first Node of the linked list
         return;
      }
      Node n2=start;
      while(n2.next!=null)     //loop will run till the last node is reached
         n2=n2.next;           //n2 at the last Node in the linked list
      n2.next=n1;              //n2.next set to n1
   }

   public void addAtBegin(Node start, int s)  //to add an item at the front
   {
      Node n1=new Node(s);        //new node created
      if(start==null)
      {
         start=n1;
         return;
      }
      n1.next=start;              //n1.next set to start
      start=n1;                   //n1 becomes the 1st Node
    }
```

```
 public void delAtEnd(Node start)    //to delete an item from the end
 {
    Node n1, n2;
    if(start==null)
    {
       System.out.println("Empty Linked list!! Nothing to delete.");
       return;
    }
    n1=start;                        //n1 points to 1st node
    while(n1.next!=null)             //loop to reach to the last node
       n1=n1.next;                   //n1 at the last Node in the linked list

    n2=start;
    while(n2.next!=n1)               //loop to reach to the 2nd last node
       n2=n2.next;
    n2.next=null;                    //2nd last Node's next set to null
    System.out.println("Node deleted : value="+n1.data);
    n1=null;                         //node n1 gets deleted from the list
}

public void delAtBegin(Node start)  //to delete an item from the front
{
    Node n1;
    if(start==null)
    {
       System.out.println("\nEmpty Linked list! Nothing to delete.");
       return;
    }
    n1=start;                        //n1 is the first Node
    start=n1.next;                   //start is pointing to the 2nd Node
    System.out.println("Node deleted : value="+n1.data);
    n1=null;                         //node n1 gets deleted from the list
}

public void display(Node start) //to display the linked list
{
    Node n1;
    if(start==null)
    {
       System.out.println("\nEmpty Linked list! Nothing to display.");
       return;
    }
    n1=start;
    while(n1!=null) //loop will run until we reach null
    {
       System.out.print(n1.data+" --> ");
       n1=n1.next;
    }
    System.out.println("NULL"); //optional statement
}
```

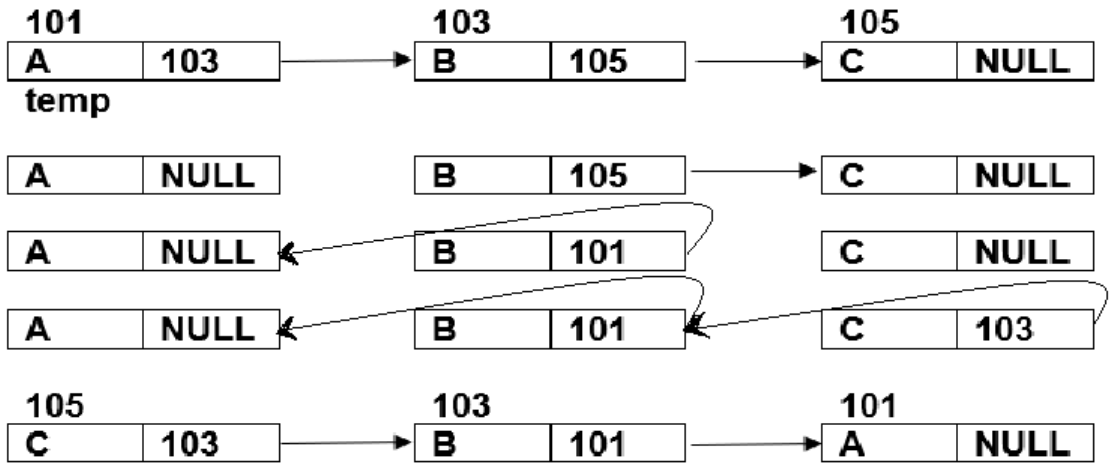
```
public int countNode(Node start) //returning the total no of nodes in the list
 {
    Node n1;
    if(start==null)
       return 0;
    int i=0;
    for(n1=start; n1!=null; n1=n1.next) //loop will run from the first node till NULL
    {
       i++; // incrementing the counter
    }
    return i;
 }

 public void addAtPos(Node start, int p, int n) //to add an item, n, at the pos. p
 {
    Node n1=new Node(n);        //new Node created
    if(start==null)             //as no linked list exists
    {
       start=n1;                //new Node becomes the first Node of the list
       return;
    }
    int i=countNode();          //calling the countNode function
    if(p<1||p>i+1)
    {
       System.out.println("Position out of range");
       return;
    }
    if(p==1)
      addAtBegin(start, n); //function call to add the data at the beginning of the list
    else if(p==i+1)
      appendNode(start, n); //function call to add the data at the end of the list
    else
    {
       Node n2=start;
       for(i=0; i<p-1; i++)  //loop will take n2 at one node previous to position p
           n2=n2.next;
       n1.next=n2.next;    //new node gets inserted at the position p
       n2.next=n1;         //n2.next set to n1
    }
 }
```

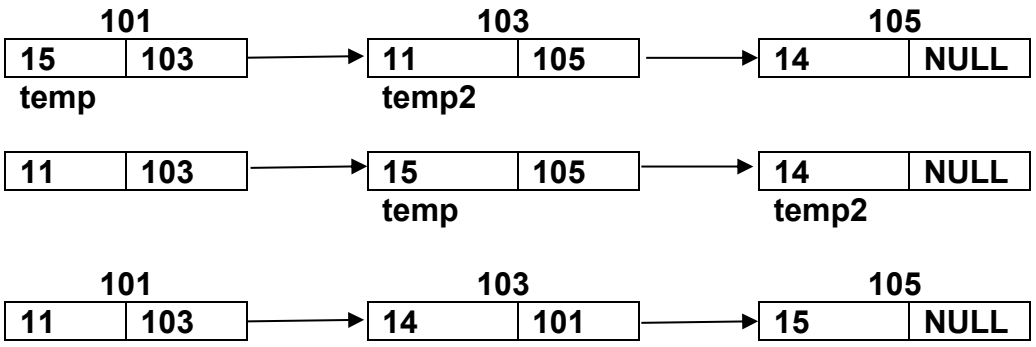
```
public void delAtPos(Node start, int p) //to delete an item from p in the list
  {
     Node n1;
     if(start==null)    //as no linked list exists
     {
        System.out.println("\nEmpty Linked list! Nothing to delete.");
        return;
     }
     int i=countNode();       //calling the countNode function
     if(p<1||p>i)
     {
        System.out.println("Position out of range");
        return;
     }
     if(p==1)
      delAtBegin(start);      //function call for deleting the first node from the linked list
     else if(p==i)
      delAtEnd(start);        //function call for deleting the last node from the linked list
     else
     {
        Node n2=start;
        for(i=0; i<p-1; i++)  //loop to reach one node before the deleting position
            n2=n2.next;
        n1=n2.next;                    //pointing to the deleting node
        n2.next=n1.next;               //n2.next set to n1
        System.out.println("Node deleted : value="+n1.data);
       n1=null;                        //node n1 gets removed from the linked list
     }
  }

  public void searchNode(Node start, int a)  //searching a no. in the linked list
  {
     Node n1;
     if(start==null)
     {
        System.out.println("\nEmpty Linked list! Nothing to display.");
        return;
     }
     boolean flag=false;
     for(int i=1, n1=start; n1!=null; n1=n1.next,i++)
     {
        if(n1.data==a)
        {
           flag=true;
           System.out.println("Element found at Node "+i);
        }
     }
     if(flag==false)
     System.out.println("No. not present in the list");
  }
```

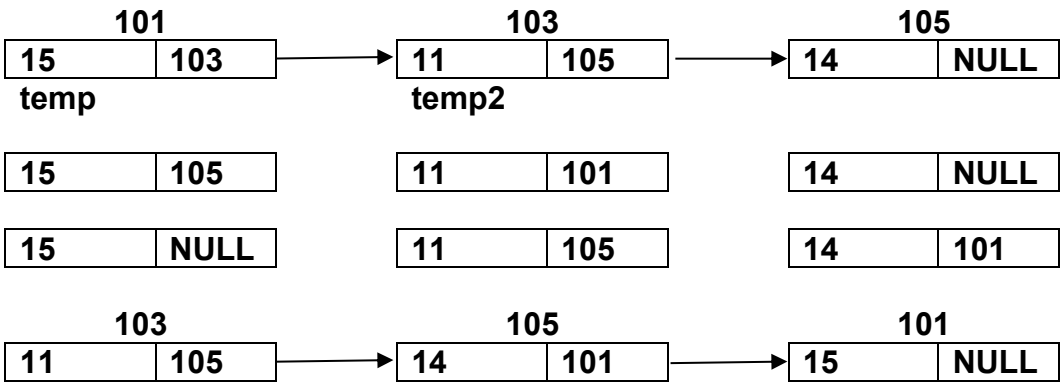
```
public void reverseList(Node start) //to reverse the linked list
{
   Node n1, n2, temp; //creating 3 objects of Node class
   temp=start; //Node temp is pointing to start i.e. the first node
   n2=null; //Node n2 points to null
   if(start==null)
   {
      System.out.println("\nEmpty Linked list! Nothing to reverse.");
      return;
   }
   while(temp!=null) //loop runs from the first node until reaches null
   {
      n1=n2;         // Node n1 will point to node n2
      n2=temp;       // Node n2 will point to temp
      temp=temp.next; // Node temp will move to the next node in the list
      n2.next=n1;  // next pointer of Node n2 will point to Node n1
   }
   start=n2;          // start will point to Node n2
   System.out.println("\nReversed list");
   display(start);
}
```



```
public void sortAsc(Node start)  //selection sort in ascending order
  {
     int len=countNode();
     int r1;
     Node temp,temp2;
     temp=start;
     if(start==null)
     {
        System.out.println("empty list, sorting not possible\n");
        return;
     }
     for(int i=0; i<len-1; i++)
     {
        temp2=temp.next;
        for(int j=i+1; j<len; j++)
        {
           if(temp.data>temp2.data)
           {
              r1=temp.data;
              temp.data=temp2.data;
              temp2.data=r1;
           }
           temp2=temp2.next;
        }
        temp=temp.next;
     }
     display(start);
  }
```



```
public void sortAscMem(Node start) //selection sort with address change
 {
    int len=countNode();
    Node temp,temp2,temp3;
    temp=start;
    if(start==null)
    {
       System.out.println("empty list, sorting not possible\n");
       return;
    }
    for(int i=0; i<len-1; i++)
    {
       temp2=temp.next;
       for(int j=i+1; j<len; j++)
       {
          if(temp.data>temp2.data)
          {
             temp3=start;
             while(temp3.next!=temp)
               temp3=temp3.next;
             temp.next=temp2.next;
             temp3.next=temp2;
             temp2.next=temp;
          }
          temp2=temp2.next;
       }
       temp=temp.next;
    }
    display(start);
 }
```



```
    public void mergeList(LinkedList L1, LinkedList L2) //merging 2 lists
    {
        Node n1;
        if(L1.start==null && L2.start==null)
        {
            System.out.println("Both Lists are Empty. Merge not possible");
            return;
        }
        else
        {
            this.start=L1.start;                    //1st list is pointed by n1
            n1=start;
            while(n1.next!=null)
                    n1=n1.next;
            n1.next=L2.start;                       //1st list linked to 2nd list
            System.out.println("Merged list :");
            display(this.start);
        }
    }

    public void splitList(Node start, int p)        //splitting 1 list into 2
    {
        LinkedList L=null;
        if(start==null)                             //as no linked list exists
        {
            System.out.println("\nEmpty Linked list!! Cannot split.");
            return;
        }
        int i=countNode();
        if(p<=1||p>i)
        {
            System.out.println("Position of splitting is out of range");
            return;
        }
        Node n1=start;
        for(i=1; i<p-1; i++)
                n1=n1.next;
        Node n2=n1.next;                            //n2 set to the new split node at pos p
        n1.next=null;                               //n1 terminates at pos p
        System.out.println("\n1st list:");
        display(n1);
        System.out.println("\n2nd list:");
        display(n2);
    }
}
```