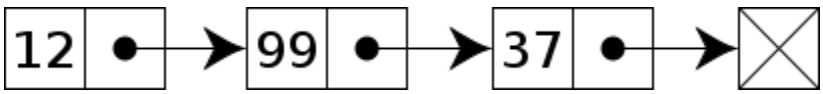
Linked list

In computer science, a **linked list** is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a datum and a reference (in other words, a *link*) to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.

The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal.



CODES (C)

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
typedef struct linkedlist
{
	int no;
	struct linkedlist *n;
}node;
void append(node **p,int z)
{
	node * t,*r;
	t=(node *)malloc(sizeof(node));
	t->no=z;
	t->n=NULL;
	if(*p==NULL)
	*p=t;
	else
	{
		r=*p;
		while(r->n!=NULL)r=r->n;
		r->n=t;
	}
}
void display(node *p)
{
	for(;p;p=p->n)
		printf("%d",p->no);
}
int main()
{
	node * t=NULL;
	int no1,ans;
	clrscr();
	do
	{
		printf("\nEnter number :");
		scanf("%d",&no1);
		append(&t,no1);
		printf("\n continue (Y/N)");
		scanf("%d",&ans);
	}while(ans=='Y'||ans=='y');
	display(t);
	getch();
	return 0;
}
```

CODES (JAVA)

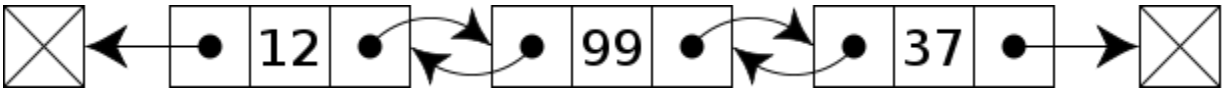
```
import java.lang.*;
import java.util.*;

public class Node
{
  Node data;
  node next;
}

public class SinglyLinkeList
{
  Node start;
  int size;
}

 public SinnglyLinkedList()
{
 start=null;
 size=0;
}



public void add(Node data)
{
  if(size=0)
  {
    start=new Node();
    start.next=null;
    start.data=data;
  }
  else
  {
    Node currentnode=getnode(size-1);
    Node newnode=new Node();
    newnode.data=data;
    newnode.next=null;
    currentnode.next=newnode;
  }
  size++;
 }
```

```
public void insertfront(Node data)
 {
  if(size==0)
 {
   Node newnode=new Node();
   start.next=null;
   start.data=data;
 }
 else
 {
    Node newnode=new Node();
     newnode.data=data;
    newnode.next=start;
}
size++;
}

 public void insertAt(int position,Node data)
 {
   if(position==0)
   {
    insertatfront(Node data);
    }
    else if(position==size-1)
   {
    insertatlast(data);
   }
    else
   {
     Node tempnode=getNodeAt(position-1);
     Node newnode= new Node();
     newnode.data=data;
     newnode.next=tempnode.next;
     size++;
   }
 }

public Node getFirst()
{
  return getNodeAt(0);
}
 public Node getLast()
 {
   return getNodeAt(size-1);
 }
 public Node removeAtFirst()
 {
```

```
    if(size==0)
    {
      System.out.println("Empty List ");
    }
    else
    {
       Node tempnode=getNodeAt(position-1);
       Node data=tempnode.next.data;
       tempnode.next=tempnode.next.next;
       size--;
        return data;
    }
}

    Node data=start.data;
    start=start.next;
    size--;
    return data;
 }
}
 public Node removeAtLast()
 {
  if(size==0)
  {
    System.out.println("Empty List ");
  }
  else
  {
     Node data=getNodeAt(size-1);
     Node data=tempnode.next.data;
     size--;
     return data;
  }
}
```

Doubly-linked list

A **doubly-linked list** is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called *links* , that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. If there is only one sentinel node, then the list is circularly linked via the sentinel node. It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.



CODES (C)

```
#include<stdio.h>
#include<malloc.h>
#include<conio.h>
typedef struct doublelinkedlist
{
        int d;
        struct doublelinkedlist *f,*b ;
}node;
void append(node **q,node **t)
{
        int no;
        char ans;
        node *s=NULL;
        do
        {
                s=(node*)malloc(sizeof(node));
                printf("\nEnter element :");
                scanf("%d",&no);
                s->d=no;
                s->b=NULL ;
                if(*q==NULL)
                {
                        *q=s;
                        *t=s;
                }
                else
                {
                        (*q)->f=s;
                        s->b=(*q);
                        (*q)=s;
                }
                printf("\nContinue (Y/N)");
                ans=getche();
        }while(ans=='Y'||ans=='y');
        (*q)->f=NULL;
}
void delbeg(node **q)
{
        node *x=NULL;
        x=(*q);
        (*q)=(*q)->f;
        (*q)->b=NULL;
        printf("Deleted item :%d",x->d);
        free(x);
}
```

```
void delpos(node *q,int loc)
{
        int i;
        node *x=NULL;
        for(i=1;i<=loc-2;i++)
                q=q->f;
        x=q->f;
        q->f=x->f;
        if(x->f)
                x->f->b=q;
        printf("Deleted item :%d",x->d);
        free(x);
}
void insertatbeg(node **q,int z)
{
        node *m;
        m=(node *)malloc(sizeof(node));
        m->d=z;
        m->f=(*q);
        (*q)->b=m;
        (*q)=m;
        m->b=NULL;
}
void insertatpos(node *q,int loc,int z)
{
        node *t;
        int i;
        t=(node *)malloc(sizeof(node));
        t->d=z;
        for(i=1;i<=loc-2;i++)
                q=q->f;
        t->f=q->f;
        t->b=q;
        q->f=t;
        if(t->f)
                t->f->b=q;
}
void display(node *q,node *t)
{
        for(q=t;q;q=q->f)
                printf("%d->",q->d);
}
void reverse(node *q)
{
         for(;q;q=q->b)
                printf("%d->",q->d);
}
```

CODES (JAVA)

```
import java.lang.*;
import java.util.*;
import java.io.*;

class DLinkedList
{
  private int data;

  private DLinkedList next;
  private DLinkedList prev;

  public DLinkedList()
  {
    data = 0;
    next = null;
    prev = null;
  }

  public DLinkedList(int value)
  {
    data = value;
    next = null;
    prev = null;
  }

  public DLinkedList InsertNext(int value)
  {
    DLinkedList node = new DLinkedList(value);
    if(this.next == null)
    {

        // Easy to handle
        node.prev = this;
        node.next = null; // already set in constructor
        this.next = node;
    }
    else
    {
        // Insert in the middle
        DLinkedList temp = this.next;
        node.prev = this;
        node.next = temp;
        this.next = node;
        temp.prev = node;
        // temp.next does not have to be changed
```

```
  }
  return node;
}

public DLinkedList InsertPrev(int value)
{
  DLinkedList node = new DLinkedList(value);
  if(this.prev == null)
  {
      node.prev = null; // already set on constructor
      node.next = this;
      this.prev = node;
  }
  else
  {

      // Insert in the middle
      DLinkedList temp = this.prev;
      node.prev = temp;
      node.next = this;
      this.prev = node;
      temp.next = node;
      // temp.prev does not have to be changed
  }
  return node;
}

public void TraverseFront()
{
  TraverseFront(this);
}

public void TraverseFront(DLinkedList node)
{
  if(node == null)
      node = this;
  System.out.println("\n\nTraversing in Forward Direction\n\n");

  while(node != null)
  {
      System.out.println(node.data);
      node = node.next;
  }
}
```

```
public void TraverseBack()
{
  TraverseBack(this);
}

public void TraverseBack(DLinkedList node)
{
  if(node == null)
      node = this;
  System.out.println("\n\nTraversing in Backward Direction\n\n");
  while(node != null)
  {
      System.out.println(node.data);
      node = node.prev;
  }
}
```