



SUBJECT: DATA STRUCTURE AND PROGRAM DESIGN

SUBJECT CODE: BECSE402T

SEMESTER: 4TH (CSE)

SUBJECT INCHARGE: DR. SHRIKANT ZADE

UNIT 3 NOTES

UNIT III:

(07 Hrs)

Linked List: Representation of ordered list using array and its operation, Linked Lists, Singly linked list, Implementation of linked list using static and dynamic memory allocation, operations on linked list, polynomial representations using linked list, circular linked list, doubly linked list.

UNIT-III

LINEAR LIST

INTRODUCTION

Linear Data Structures:

Linear data structures are those data structures in which data elements are accessed (read and written) in sequential fashion (one by one). Ex: Stacks, Queues, Lists, Arrays

Non Linear Data Structures:

Non Linear Data Structures are those in which data elements are not accessed in sequential fashion.

Ex: trees, graphs

Difference between Linear and Nonlinear Data Structures

Main difference between linear and nonlinear data structures lie in the way they organize data elements. In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory. In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. Due to this nonlinear structure, they might be difficult to be implemented in computer's linear memory compared to implementing linear data structures. Selecting one data structure type over the other should be done carefully by considering the relationship among the data elements that needs to be stored.

LINEAR LIST

A data structure is said to be linear if its elements form a sequence. A linear list is a list that displays the relationship of adjacency between elements.

A Linear list can be defined as a data object whose instances are of the form $(e_1, e_2, e_3 \dots e_n)$ where n is a finite natural number. The e_i terms are the elements of the list and n is its length. The elements may be viewed as atomic as their individual structure is not relevant to the structure of the list. When $n=0$, the list is empty. When $n>0$, e_1 is the first element and e_n the last. I.e; e_1 comes before e_2 , e_2 comes before e_3 and so on.

Some examples of the Linear List are

- An alphabetized list of students in a class
- A list of exam scores in non decreasing order
- A list of gold medal winners in the Olympics
- An alphabetized list of members of Congress

The following are the operations that performed on the Linear List

- ✓ Create a Linear List
- ✓ Destroy a Linear List
- ✓ Determine whether the list is empty
- ✓ Determine the size of the List
- ✓ Find the element with a given index
- ✓ Find the index of a given number
- ✓ Delete, erase or remove an element given its index
- ✓ Insert a new element so that it has a given index

A Linear List may be specified as an abstract Data type (ADT) in which we provide a specification of the instance as well as of the operations that are to be performed. The below abstract data type omitted specifying operations to create and destroy instance of the data type. All ADT specifications implicitly include an operation to create an empty instance and optionally, an operation to destroy an instance.

AbstractDataType *linearList*

```
{  
    instances  
        ordered finite collections of zero or more elements  
  
    operations  
        empty() : return true if the list is empty, false otherwise  
        size() : return the list size (i.e., number of elements in the list)  
        get(index): return the indexth element of the list  
        indexOf(x): return the index of the first occurrence of x in the list,  
                    return -1 if x is not in the list  
        erase(index): remove/delete the indexth element, elements with higher in-  
                    dex have their index reduced by 1  
        insert(index, x): insert x as the indexth element, elements with index  $\geq$  index  
                    have their index increased by 1  
        output(): output the list elements from left to right  
}
```

Abstract data type specification of a linear list

Array Representation: (Formula Based Representation)

A formula based representation uses an array to represent the instance of an object. Each position of the Array is called a Cell or Node and is large enough to hold one of the elements that make up an instance, while in other cases one array can represent several instances. Individual elements of an instance are located in the array using a mathematical formula.

Suppose one array is used for each list to be represented. We need to map the elements of a list to positions in the array used to represent it. In a formula based representation, a mathematical formula determines the location of each element. A simple mapping formulas is

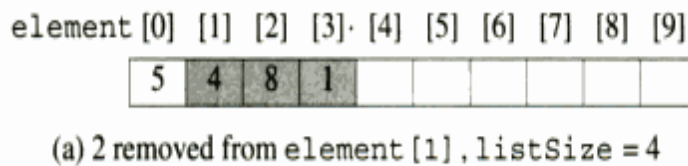
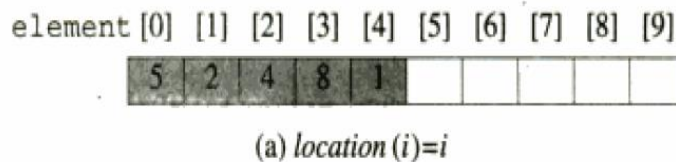
$\text{Location (i)} = i-1$

This equation states that the i^{th} element of the list is in position $i-1$ of the array. The below figure shows a five element list represented in the array element using the mapping of equation.

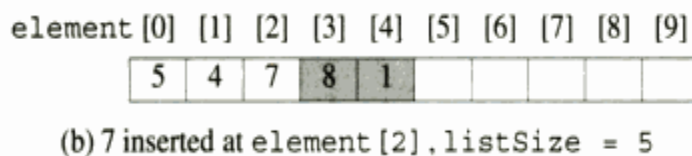
To completely specify the list we need to know its current length or size. For this purpose we use variable length. Length is zero when list is empty. Program gives the resulting C++ class definition. Since the data type of the list element may vary from application to application, we have defined a template class in which the user specifies the element data type *T*. the data members *length*, *MaxSize* and *element* are private members are private members, while the remaining members are public. Insert and delete have been defined to return a reference to a linear list.

Insertion and Deletion of a Linear List:

Suppose we want to remove an element e_i from the list by moving to its right down by 1. For example, to remove an element $e_1=2$ from the list, we have to move the elements $e_2=4$, $e_3=8$, and $e_4=1$, which are to the right of e_1 , to positions 1, 2 and 3 of the array element. The below figure shows this result. The shaded elements are moved.



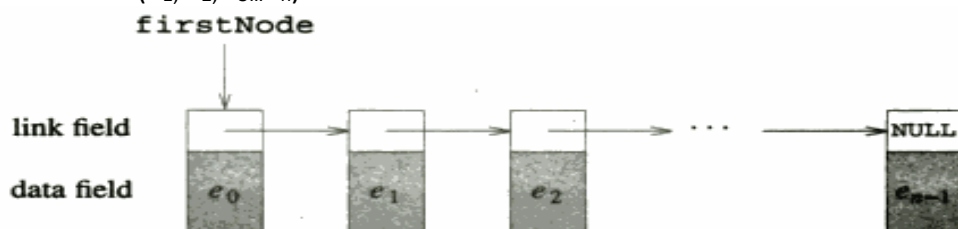
To insert an element so that it becomes element i of a list, must move the existing element e_i and all elements to its right one position right and then put the new element into position i of the array. For example to insert 7 as the second element of the list, we first move elements e_2 and e_3 to the right by 1 and then put 7 in to second position 2 of the array. The below figure shows this result. The shaded elements were moved.



Linked Representation And Chains

In a linked list representation each element of an instance of a data object is represented in a cell or node. The nodes however need not be component of an array and no formula is used to locate individual elements. Instead of each node keeps explicit information about the location of other relevant nodes. This explicit information about the location of another node is called Link or Pointer.

Let $L=(e_1, e_2, e_3...e_n)$ be a linear List. In one possible linked representation for this list, each element e_i is represented in a separate node. Each node has exactly one link field that is used to locate the next element in the linear list. So the node for e_i links to that for e_{i+1} , $0 \leq i < n-1$. The node for e_{n-1} has no need to link to and so its link field is NULL. The pointer variables first locate the first node in the representation. The below figure shows the linked representation of a List $= (e_1, e_2, e_3...e_n)$.

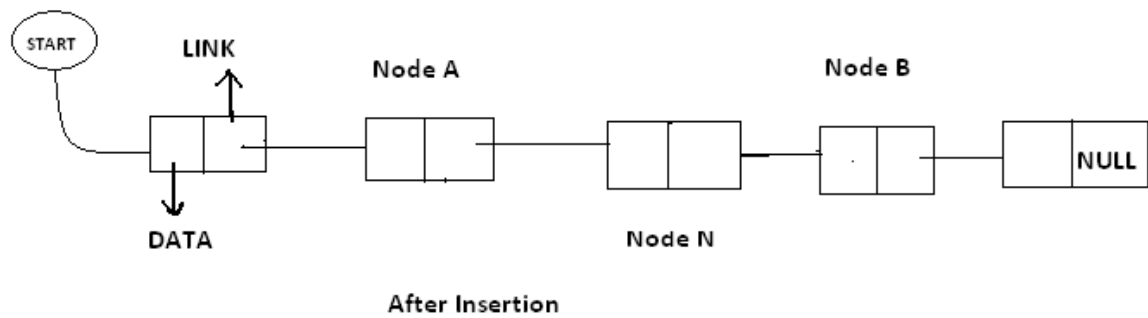
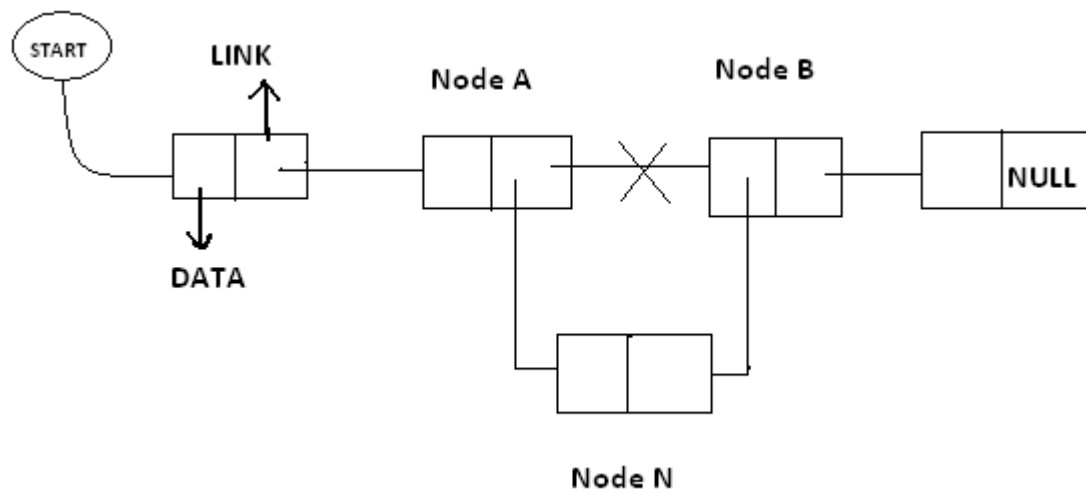


Linked representation of a linear list

Since each node in the Linked representation of the above figure has exactly one link, the structure of this figure is called a '**Single Linked List**'. the nodes are ordered from left to right with each node (other than last one) linking to the next, and the last node has a NULL link, the structure is also called a **chain**.

Insertion and Deletion of a Single Linked List:

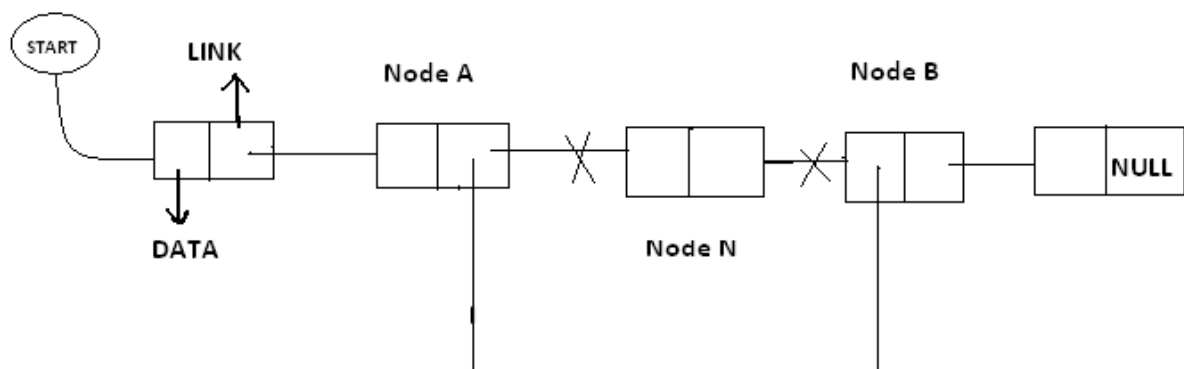
Insertion Let the list be a Linked list with successive nodes A and B as shown in below figure. suppose a node N is to be inserted into the list between the node A and B.

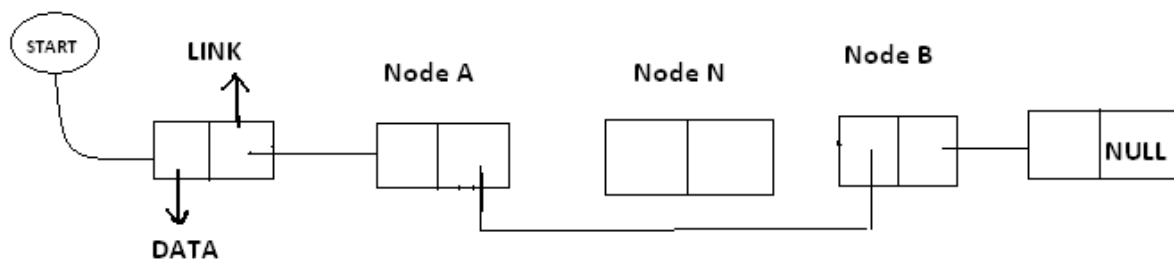


In the New list the Node A points to the new Node N and the new node N points to the node B to which Node A previously pointed.

Deletion:

Let list be a Linked list with node N between Nodes A and B is as shown in the following figure.



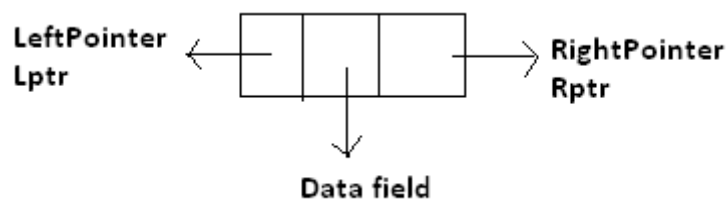


After Deletion Node N in Between Node A abd Node B

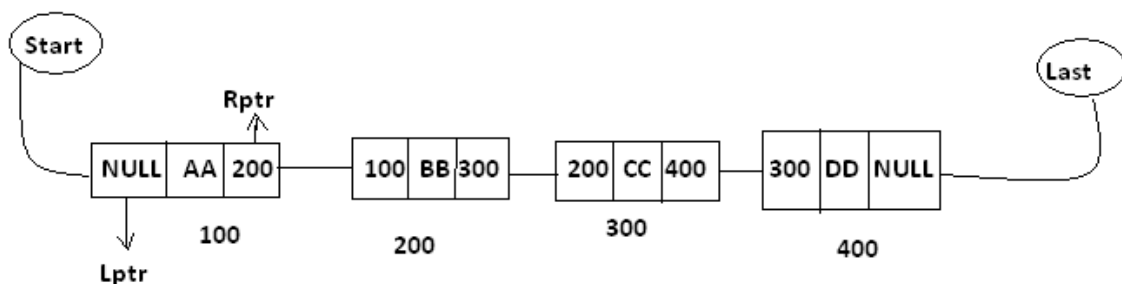
In the new list the node N is to be deleted from the Linked List. The deletion occurs as the link field in the Node A is made to point node B this excluding node N from its path.

DOUBLE LINKED LIST (Or) TWO WAY LINKED LIST

In certain applications it is very desirable that list be traversed in either forward direction or Back word direction. The property of Double Linked List implies that each node must contain two link fields instead of one. The links are used to denote the preceding and succeeding of the node. The link denoting the preceding of a node is called Left Link. The link denoting succeeding of a node is called Right Link. The list contain this type of node is called a **"Double Linked List"** or **"Two Way List"**. The Node structure in the Double Linked List is as follows:

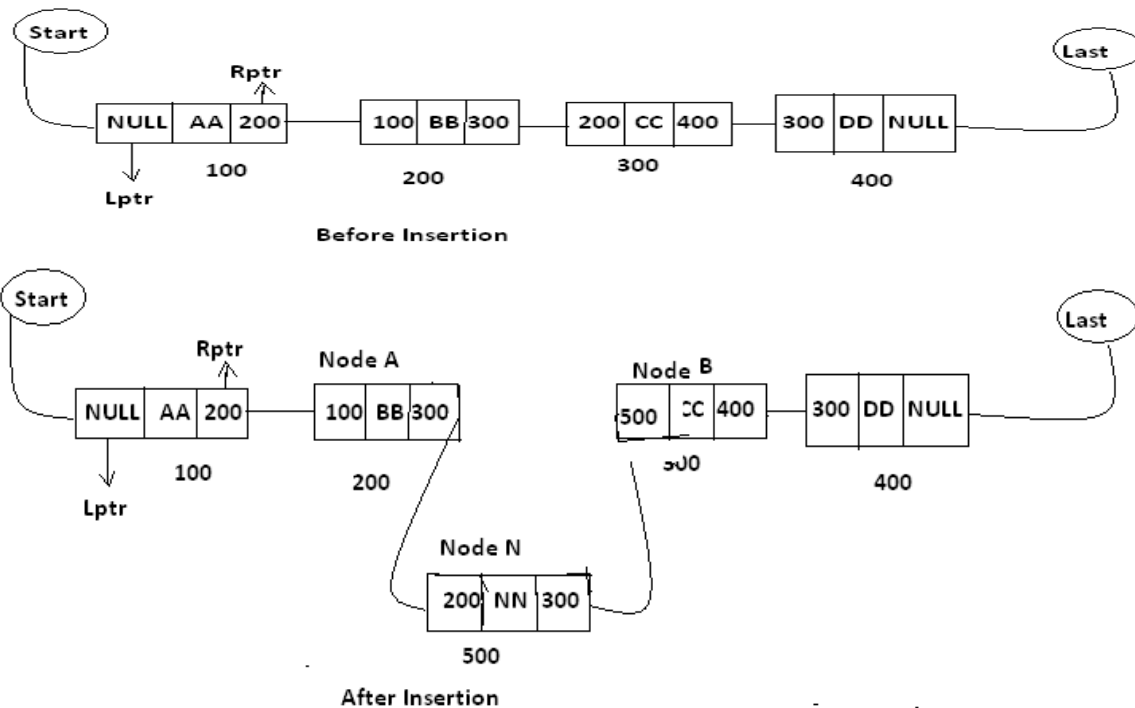


Lptr contains the address of the before node. Rptr contains the address of next node. Data Contains the Linked List is as follows.



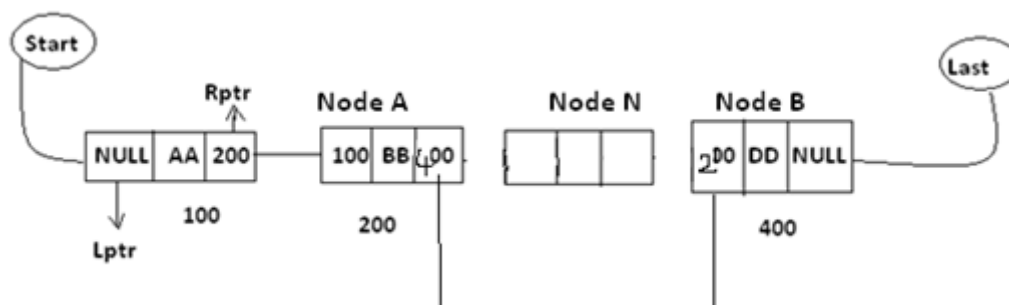
In the above diagram Last and Start are pointer variables which contains the address of last node and starting node respectively.

Insertion in to the Double Linked List: Let list be a double linked list with successive modes A and B as shown in the following diagram. Suppose a node N is to be inserted into the list between the node s A and B this is shown in the following diagram.



As in the new list the right pointer of node A points to the new node 'N', the Lptr of the node 'N' points to the node A and Rptr of node 'N' points to the node 'B' and Lptr of node B points to the new node 'N'.

Deletion Of Double Linked List :- Let list be a linked list contains node N between the nodes A and B as shown in the following diagram.

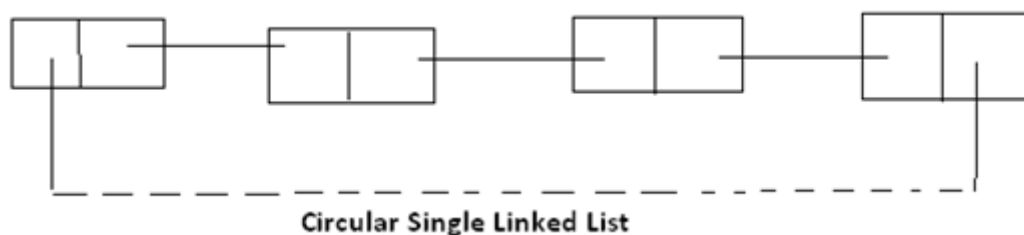


Support node N is to be deleted from the list diagram will appear as the above mentioned double linked list. The deletion occurs as soon as the right pointer field of node A is changed, so that it points to node B and the left pointer field of node B is changed. So that it points to node A.

Circular Linked List:- Circular Linked List is a special type of linked list in which all the nodes are linked in continuous circle. Circular list can be singly or doubly linked list. Note that, there are no Nulls in Circular Linked Lists. In these types of lists, elements can be added to the back of the list and removed from the front in constant time.

Both types of circularly-linked lists benefit from the ability to traverse the full list beginning at any given node. This avoids the necessity of storing first Node and last node, but we need a special representation for the empty list, such as a last node variable which points to some node in the list or is null if it's empty. This representation significantly simplifies adding and removing nodes with a non-empty list, but empty lists are then a special case. Circular linked lists are most useful for describing naturally circular structures, and have the advantage of being able to traverse the list starting at any point. They also allow quick access to the first and last records through a single pointer (the address of the last element).

Circular single linked list:



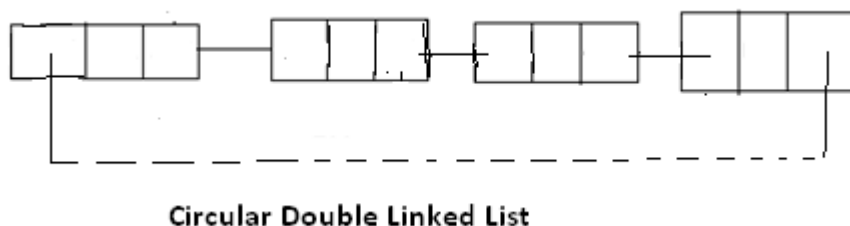
Circular linked list are one they of liner linked list. In which the link fields of last node of the list contains the address of the first node of the list instead of contains a null pointer.

Advantages:- Circular list are frequency used instead of ordinary linked list because in circular list all nodes contain a valid address. The important feature of circular list is as follows.

- (1) In a circular list every node is accessible from a given node.
- (2) Certain operations like concatenation and splitting becomes more efficient in circular list.

Disadvantages: Without some conditions in processing it is possible to get into an infinite Loop.

Circular Double Linked List :- These are one type of double linked list. In which the rpt field of the last node of the list contain the address of the first node ad the left points of the first node contains the address of the last node of the list instead of containing null pointer.



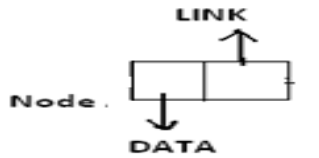
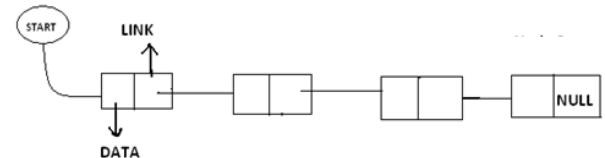
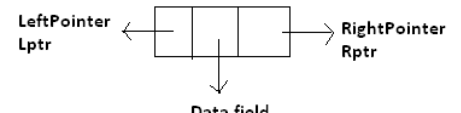
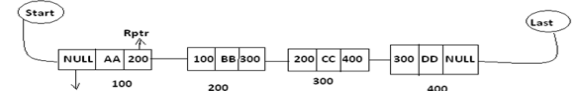
Advantages:- circular list are frequently used instead of ordinary linked list because in circular list all nodes contained a valid address. The important feature of circular list is as follows.

- (1) In a circular list every node is accessible from a given node.
- (2) Certain operations like concatenation and splitting becomes more efficient in circular list.

Disadvantage:- Without some conditions in processes it is possible to get in to an infant glad.

Difference between single linked list and double linked list?

Single linked list(SLL)	Double linked list(DLL)
1. In Single Linked List the list will be traversed in only one way ie; in forward. 2. In Single Linked List the node contains one link field only. 3. Every node contains the address of next node. 4. The node structure in Single linked list is as follows:	1. In Double Linked List the list will be traversed in two way ie; either forward and backward 2. In Double Linked List the node contains two link fields. 3. Every node contains the address of next node as well as preceding node. 4. the node structure in double linked list is as follows:

 <p>5. The conceptual view of SLL is as follows:</p>  <p>6. SLL are maintained in memory by using two arrays.</p>	 <p>5.the conceptual view of DLL is as follows:</p>  <p>6. DLL is maintained in memory by using three arrays.</p>
---	---

2. Difference between sequential allocation and linked allocation?

OR

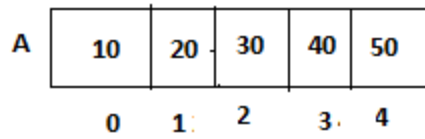
Difference between Linear List and Linked List?

OR

Difference between Arrays and Linked List?

Arrays	Linked List
<ol style="list-style-type: none"> 1. Arrays are used in the predictable storage requirement ie; extent amount of data storage required by the program can be determined. 2. In arrays the operations such as insertion and deletion are done in an inefficient manner. 3. The insertion and deletion are done by moving the elements either up or down. 4. Successive elements occupy adjacent space on memory. 5. In arrays each location contains DATA only 6. The linear relationship between the data elements of an array is reflected by the physical relationship of data in the memory. 7. In array declaration a block of memory space is required. 8. There is no need of storage of pointer or lines 	<ol style="list-style-type: none"> 1. Linked List are used in the unpredictable storage requirement ie; extent amount of data storage required by the program can't be determined. 2. In Linked List the operations such as insertion and deletion are done more efficient manner ie; only by changing the pointer. 3. The insertion and deletion are done by only changing the pointers. 4. Successive elements need not occupy adjacent space. 5. In linked list each location contains data and pointer to denote whether the next element present in the memory. 6. The linear relationship between the data elements of a Linked List is reflected by the Linked field of the node. 7. In Linked list there is no need of such thing.

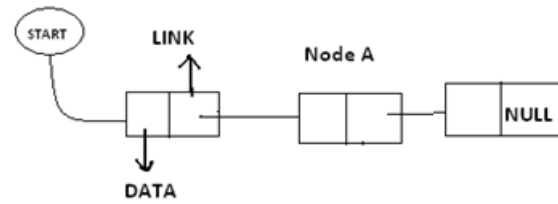
9. The Conceptual view of an Array is as follows:



10. In array there is no need for an element to specify whether the next is stored

8. In Linked list a pointer is stored along into the element.

9. The Conceptual view of Linked list is as follows:



10. There is need for an element (node) to specify whether the next node is formed.