**LINKED LISTS**

Linked lists and arrays are similar since they both store collections of data. Array is the most common data structure used to store collections of elements. Arrays are convenient to declare and provide the easy syntax to access any element by its index number. Once the array is set up, access to any element is convenient and fast.

**The disadvantages of arrays are:**

* The size of the array is fixed. Most often this size is specified at compile time. This makes the programmers to allocate arrays, which seems "large enough" than required.
* Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.
* Deleting an element from an array is not possible. Linked lists have their own strengths and weaknesses, but they happen to be strong where arrays are weak.
* Generally array's allocates the memory for all its elements in one block whereas linked lists use an entirely different strategy. Linked lists allocate memory for each element separately and only when necessary.

**Linked List Concepts:**

A linked list is a non-sequential collection of data items. It is a dynamic data structure. For every data item in a linked list, there is an associated pointer that would give the memory location of the next data item in the linked list. The data items in the linked list are not in consecutive memory locations. They may be anywhere, but the accessing of these data items is easier as each data item contains the address of the next data item.

**Advantages of linked lists:**

Linked lists have many advantages. Some of the very important advantages are:

1. Linked lists are dynamic data structures. i.e., they can grow or shrink during the execution of a program.
2. Linked lists have efficient memory utilization. Here, memory is not pre-allocated. Memory is allocated whenever it is required and it is de-allocated (removed) when it is no longer needed.
3. Insertion and Deletions are easier and efficient. Linked lists provide flexibility in inserting a data item at a specified position and deletion of the data item from the given position.
4. Many complex applications can be easily carried out with linked lists.

**Disadvantages of linked lists:**

1. It consumes more space because every node requires a additional pointer to store address of the next node.
2. Searching a particular element in list is difficult and also time consuming.
3. Random access to an element is not possible.

**Types of Linked Lists:**

Basically we can put linked lists into the following four items:

1. Single Linked List.
2. Double Linked List.
3. Circular Linked List.

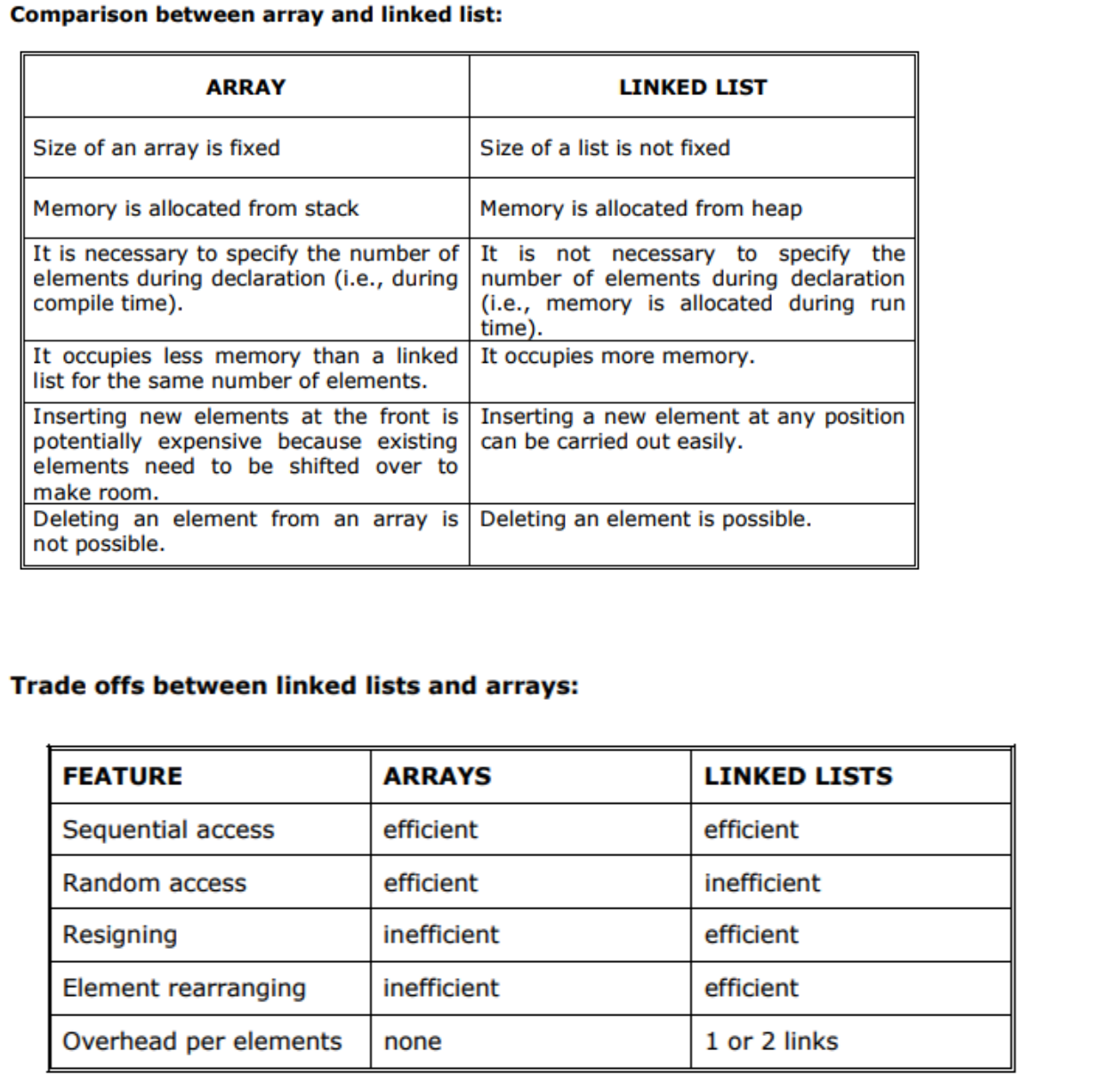
4. Circular Double Linked List.

A single linked list is one in which all nodes are linked together in some sequential manner. Hence, it is also called as linear linked list.

A double linked list is one in which all nodes are linked together by multiple links which helps in accessing both the successor node (next node) and predecessor node (previous node) from any arbitrary node within the list. Therefore each node in a double linked list has two link fields (pointers) to point to the left node (previous) and the right node (next). This helps to traverse in forward direction and backward direction.

A circular linked list is one, which has no beginning and no end. A single linked list can be made a circular linked list by simply storing address of the very first node in the link field of the last node.

A circular double linked list is one, which has both the successor pointer and predecessor pointer in the circular manner.

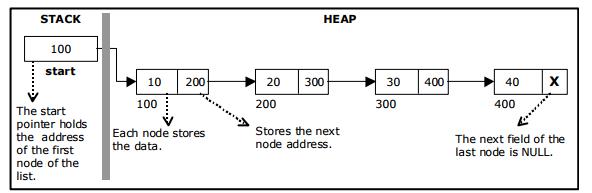


**Applications of linked list:**

1. Linked lists are used to represent and manipulate polynomial. Polynomials are expression containing terms with non zero coefficient and exponents. For example: P(x) = a0 Xn + a1 Xn-1 + …… + an-1 X + an
2. Represent very large numbers and operations of the large number such as addition, multiplication and division.
3. Linked lists are to implement stack, queue, trees and graphs. 4. Implement the symbol table in compiler construction.

**Single Linked List:**

A linked list allocates space for each element separately in its own block of memory called a "node". The list gets an overall structure by using pointers to connect all its nodes together like the links in a chain. Each node contains two fields; a "data" field to store whatever element, and a "next" field which is a pointer used to link to the next node. Each node is allocated in the heap using malloc(), so the node memory continues to exist until it is explicitly de-allocated using free(). The front of the list is a pointer to the “start” node.



A single linked list

The beginning of the linked list is stored in a "start" pointer which points to the first node. The first node contains a pointer to the second node. The second node contains a pointer to the third node, ... and so on.

The last node in the list has its next field set to NULL to mark the end of the list. Code can access any node in the list by starting at the start and following the next pointers.

The start pointer is an ordinary local pointer variable, so it is drawn separately on the left top to show that it is in the stack. The list nodes are drawn on the right to show that they are allocated in the heap.

**The basic operations in a single linked list are:**

* + Creation.
* Insertion.
  + Deletion.
  + Traversing.

**Creating a Single Linked List:**

The data structure for linked list will be a structure.

Struct node

{

int data;

struct node \* next;

} \*Start= NULL;

The empty Linked list has been created.

Start

NULL

There is no node. Linked List is empty.

**Insertion of a Node:**

One of the most primitive operations that can be done in a singly linked list is the insertion of a node. Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty next field. The data field of the new node is then stored with the information read from the user. The next field of the new node is assigned to NULL.

40

Newnode

The new node can then be inserted at three different places namely:

* + Inserting a node at the beginning.
* Inserting a node at the end.
  + Inserting a node at intermediate position.

**Insertions of a node in begin of empty list.**

Start

40

Newnode

**Insertions of a node in begin of non-empty list.**

Start

40

**The existing list.**

**The new node to be inserted at begin**

30

Newnode

**Set Newnode->next=Start.**

Start

40

Newnode

30

**Set Start= Newnode.**

Start

Newnode

40

30

**Algorithm- Insertion at begin:**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL.
3. Set Newnode->next=Start.
4. Set Start=Newnode.
5. Exit.

Start

40

30

Newnode

20

ptr

**Algorithm- Insertion at end:**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL.
3. If Start= = NULL then
4. Set Start= Newnode
5. Exit.
6. Set pointer ptr=Start.
7. While ptr->next!=NULL

Set ptr=ptr->next.

End while

1. Set ptr->next=Newnode.
2. Exit.

Start

40

20

30

Count=3, LOC=3

prev

ptr

50

Newnode

**Algorithm- Insertion at a specific location (LOC):**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL.
3. If Start= = NULL && LOC=1 then
4. Set Start= Newnode
5. Exit.
6. Set pointer ptr=Start and set integer count=1.
7. While ((ptr->next!=NULL) && (count < LOC))
8. Set pointer prev=ptr
9. Set ptr=ptr->next.
10. count=count+1.

End while

1. if count= =LOC then
   1. Set prev->next=Newnode
   2. Set newnode->next=ptr

End if

Else print location is not in range.

1. Exit.

**Algorithm-Traversing the linked list**

1. Set pointer ptr=Start.
2. While ptr!=NULL
3. Print ptr->data.
4. Set ptr=ptr->next.

End while

1. Exit

Start

40

30

temp

20

20

30

**Algorithm-Deletion of a node at begin**

1. Set pointer temp=Start.
2. If temp= =NULL then
   1. Print list empty
   2. Exit

End if.

1. Set Start= temp->next
2. Print temp->data
3. Free(temp).
4. Exit

20

30

40

30

Start

prev

temp

**Algorithm-Deletion of a node at end**

1. Set pointer temp=Start.
2. If temp= =NULL then //empty linked list
   1. Print list empty
   2. Exit.

End if.

1. If temp->next = =NULL then //Linked list having single node only
   1. Set Start= NULL
   2. Print temp->data
   3. Free(temp)
   4. Exit.

End if.

1. While (temp->next!=NULL) //Linked list is having more than one node
2. Set pointer prev=temp
3. Set temp=temp->next.

End while

1. Set prev->next=NULL
2. Print temp->data
3. Free(temp).
4. Exit

20

30

40

30

Start

prev

temp

**Algorithm-Deletion of a node at a specific location**

1. Set pointer temp=Start.
2. If temp= =NULL then //empty linked list
   1. Print list empty
   2. Exit.

End if.

1. If temp->next = =NULL then //Linked list having single node only
   1. Set Start= NULL
   2. Print temp->data
   3. Free(temp)
   4. Exit.

End if.

1. Set integer count=1.
2. While ((temp->next!=NULL) && (count < LOC)) //Suppose LOC=3
3. Set pointer prev=temp
4. Set temp=temp->next.
5. count=count+1.

End while

1. if count= =LOC then
   1. Set prev->next=temp->next
   2. Print temp->data.
   3. Free (temp)

End if

Else print location is not in range.

1. Exit

**Creating a Doubly Linked List:**

The data structure for linked list will be a structure.

Struct node

{

int data;

struct node \* next;

Struct node \* back;

} \*Start= NULL;

The empty Linked list has been created.

Start

NULL

There is no node. Linked List is empty.

**Insertion of a Node:**

One of the most primitive operations that can be done in a singly linked list is the insertion of a node. Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty next field. The data field of the new node is then stored with the information read from the user. The next field of the new node is assigned to NULL.

Newnode

40

The new node can then be inserted at three different places namely:

* + Inserting a node at the beginning.
* Inserting a node at the end.
  + Inserting a node at intermediate position.

**Insertions of a node in begin of empty list.**

Start

Newnode

40

**Insertions of a node in begin of non-empty list.**

Start

40

**The existing list.**

**The new node to be inserted at begin**

30

Newnode

**Set Newnode->next=Start.**

**Set Start->back=Newnode**

**Set Start= Newnode.**

30

Start

40

Newnode

**Algorithm- Insertion at begin:**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL and Newnode->back=NULL .
3. If Start = = NULL then Set start=Newnode and exit.

Else

1. Set Newnode->next=Start.
2. Set Start->back=Newnode.
3. Set Start=Newnode.

End else.

1. Exit.

Start

40

30

ptr

20

Newnode

**Algorithm- Insertion at end:**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL and Newnode->back=NULL.
3. If Start= = NULL then
4. Set Start= Newnode and Exit.
5. Set pointer ptr=Start.
6. While ptr->next!=NULL

Set ptr=ptr->next.

End while

1. Set ptr->next=Newnode.
2. Set Newnode->back=ptr.
3. Exit.

Start

40

20

30

Count=1, LOC=3

prev

ptr

50

Newnode

**Algorithm- Insertion at a specific location (LOC):**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL and Newnode->back=NULL.
3. If Start= = NULL && LOC=1 then
4. Set Start= Newnode
5. Exit.
6. Set pointer ptr=Start and set integer count=1.
7. While ((ptr->next!=NULL) && (count < LOC))
8. Set pointer prev=ptr
9. Set ptr=ptr->next.
10. count=count+1.

End while

1. if count= =LOC then
   1. Set prev->next=Newnode
   2. Set newnode->next=ptr
   3. Set Newnode->back=prev
   4. Set ptr->back=newnode

End if

Else print location is not in range.

1. Exit.

**Algorithm-Traversing the linked list**

1. Set pointer ptr=Start.
2. While ptr!=NULL
3. Print ptr->data.
4. Set ptr=ptr->next.

End while

1. Exit

Start

40

30

20

20

30

temp

**Algorithm-Deletion of a node at begin**

1. Set pointer temp=Start.
2. If temp= =NULL then
   1. Print list empty
   2. Exit

End if.

1. Set Start= temp->next
2. Set Start->back=NULL
3. Print temp->data
4. Free(temp).
5. Exit

20

50

40

30

Start

prev

temp

**Algorithm-Deletion of a node at end**

1. Set pointer temp=Start.
2. If temp= =NULL then //empty linked list
   1. Print list empty
   2. Exit.

End if.

1. If temp->next = =NULL then //Linked list having single node only
   1. Set Start= NULL
   2. Print temp->data
   3. Free(temp)
   4. Exit.

End if.

1. While (temp->next!=NULL) //Linked list is having more than one node
2. Set pointer prev=temp
3. Set temp=temp->next.

End while

1. Set prev->next=NULL
2. Set temp->back=NULL
3. Print temp->data
4. Free(temp).
5. Exit

20

30

40

30

Start

prev

temp

Temp->next

**Algorithm-Deletion of a node at a specific location**

1. Set pointer temp=Start.
2. If temp= =NULL then //empty linked list
   1. Print list empty
   2. Exit.

End if.

1. If temp->next = =NULL then //Linked list having single node only
   1. Set Start= NULL
   2. Print temp->data
   3. Free(temp)
   4. Exit.

End if.

1. Set integer count=1.
2. While ((temp->next!=NULL) && (count < LOC)) //Suppose LOC=3
3. Set pointer prev=temp
4. Set temp=temp->next.
5. count=count+1.

End while

1. if count= =LOC then
   1. Set prev->next=temp->next
   2. Set temp->next->back=prev
   3. Set temp->next=temp->back=NULL
   4. Print temp->data.
   5. Free (temp)

End if

Else print location is not in range.

1. Exit

**Creating a Circular Linked List:**

The data structure for linked list will be a structure. In circular linked list the last node next pointer points to the first node of the linked list.

Struct node

{

int data;

struct node \* next;

} \*Start= NULL;

The empty Linked list has been created.

Start

NULL

There is no node. Linked List is empty.

**Insertion of a Node:**

One of the most primitive operations that can be done in a Circular singly linked list is the insertion of a node. Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty next field. The data field of the new node is then stored with the information read from the user. The next field of the new node is assigned to NULL.

Newnode

40

The new node can then be inserted at three different places namely:

* + Inserting a node at the beginning.
* Inserting a node at the end.
  + Inserting a node at intermediate position.

**Insertions of a node in begin of empty list.**

Start

Newnode

40

**Insertions of a node in begin of non-empty list.**

Start

**The existing list.**

40

**The new node to be inserted at begin**

30

Newnode

**Set Newnode->next=Start.**

Start

Newnode

40

30

ptr

Start

Newnode

40

30

**Algorithm- Insertion at begin:**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL.
3. If Start= = NULL then
4. Set Start= Newnode
5. Set Start->next=Start.
6. Exit.
7. Set pointer ptr=Start.
8. While ptr->next!=Start

Set ptr=ptr->next.

End while

1. Set Newnode->next=Start.
2. Set ptr->next=Newnode.
3. Set Start=Newnode.
4. Exit.

Start

40

30

20

20

Newnode

ptr

**Algorithm- Insertion at end:**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL.
3. If Start= = NULL then
4. Set Start= Newnode
5. Set Start->next=Start.
6. Exit.
7. Set pointer ptr=Start.
8. While ptr->next!=Start

Set ptr=ptr->next.

End while

1. Set ptr->next=Newnode.
2. Set Newnode->next=Start.
3. Exit.

Start

20

40

30

Count=1, LOC=3

prev

ptr

50

Newnode

**Algorithm- Insertion at a specific location (LOC):**

1. Allocate memory to Newnode.
2. Read Newnode->data and set Newnode->next=NULL.
3. If Start= = NULL && LOC=1 then
4. Set Start= Newnode
5. Set Start->next=Start.
6. Exit.
7. Set pointer ptr=Start and set integer count=1.
8. While ((ptr->next!=Start) && (count < LOC))
9. Set pointer prev=ptr
10. Set ptr=ptr->next.
11. count=count+1.

End while

1. if ((ptr->next ! = Start) && (count= =LOC)) then
   1. Set prev->next=Newnode
   2. Set newnode->next=ptr
   3. exit

End if

1. Else if ((ptr->next = = Start) && (count= =LOC)) then
   1. Set ptr->next=Newnode
   2. Set newnode->next=Start
   3. exit

Else print location is not in range.

1. Exit.

**Algorithm-Traversing the linked list**

1. Set pointer ptr=Start.
2. While ptr!=Start
3. Print ptr->data.
4. Set ptr=ptr->next.

End while

1. Exit

temp

prev

40

30

Start

20

20

30

**Algorithm-Deletion of a node at begin**

1. Set pointer temp=Start.
2. If temp= =NULL then
   1. Print list empty
   2. Exit

End if.

1. If (temp->next== Start)
   1. Print temp->data
   2. Set Start= NULL
   3. Free (temp)

End if

Else if (temp->next!= Start)

1. Set prev=Start
2. While( prev->next!=start)

Set prev=prev->next.

End while

1. Prev->next=temp->next
2. Set Start=temp->next
3. Print temp->data
4. Free(temp).
5. Exit

20

30

40

30

Start

prev

temp

**Algorithm-Deletion of a node at end**

1. Set pointer temp=Start.
2. If temp= =NULL then //empty linked list
   1. Print list empty
   2. Exit.

End if.

1. If temp->next = =Start then //Linked list having single node only
   1. Set Start= NULL
   2. Print temp->data
   3. Free(temp)
   4. Exit.

End if.

1. While (temp->next!=Start) //Linked list is having more than one node
2. Set pointer prev=temp
3. Set temp=temp->next.

End while

1. Set prev->next=Start
2. Print temp->data
3. Free(temp).
4. Exit

20

30

40

30

Start

prev

temp

**Algorithm-Deletion of a node at a specific location**

1. Set pointer temp=Start.
2. If temp= =NULL then //empty linked list
   1. Print list empty
   2. Exit.

End if.

1. If temp->next = =Start then //Linked list having single node only
   1. Set Start= NULL
   2. Print temp->data
   3. Free(temp)
   4. Exit.

End if.

1. Set integer count=1.
2. While ((temp->next!=Start) && (count < LOC)) //Suppose LOC=3
3. Set pointer prev=temp
4. Set temp=temp->next.
5. count=count+1.

End while

1. if count= =LOC then
   1. Set prev->next=temp->next
   2. Print temp->data.
   3. Free (temp)

End if

Else print location is not in range.

1. Exit