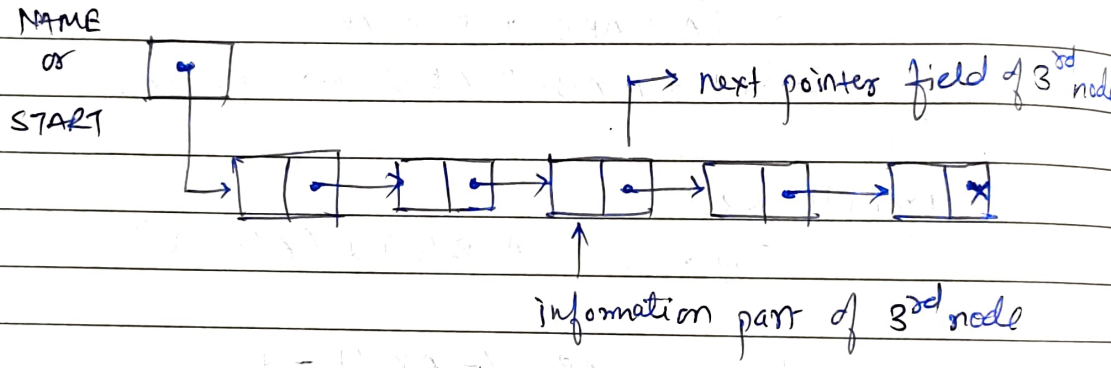
LINKED LIST.

A linked list or a one-way list is a linear collection of data elements called "nodes" where the linear order is given by means of "pointers". That is each node is divided into two parts, the first part contains the information of the element and the second part, called the link field, contains the address of the next node in the list.

Schematic diagram of a linked list with 5 nodes.



The pointer of the last node contains a special value, called the "null" pointer, which is any invalid address. In practice, 0 or a negative number is used for the null pointer.

X - to indicate null pointer in list.
START - contains address of the first node.
We need only this address to trace through the list.

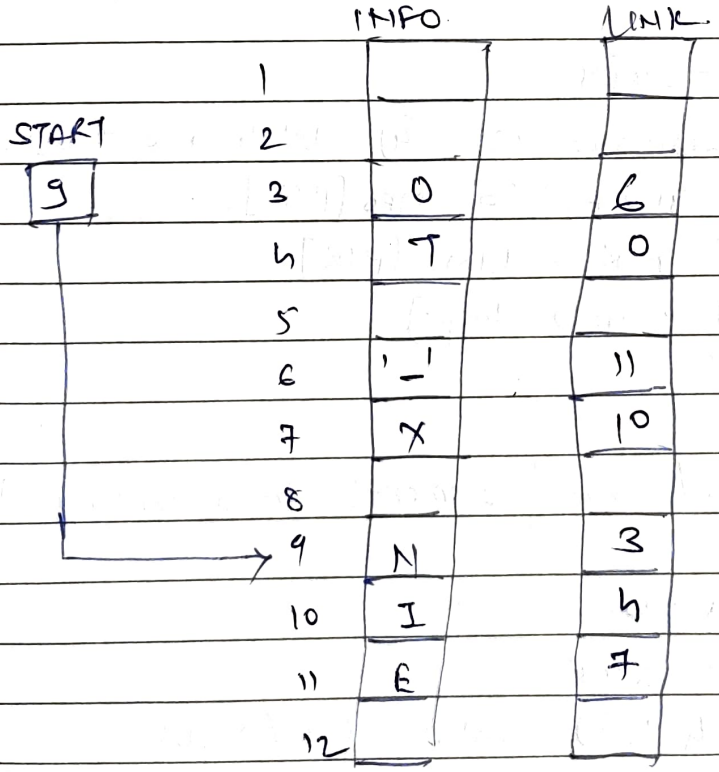
A special case is the list that has no nodes. Such a list is called the null list or empty list and is denoted by the null pointer in START.

* Representation of LINKED LIST IN MEMORY.

Let LIST, be a linked list. First, LIST requires two linear arrays — INFO and LINK — such that INFO[K] and LINK[K] contain, respectively the information part and next pointer field of a node of LIST.

LIST also requires a variable - such as START - which contains the location of the beginning of the list and a next nextpointer sentinel denoted by NULL - which indicates the end of the list. In general, we choose NULL = 0.

Example :-



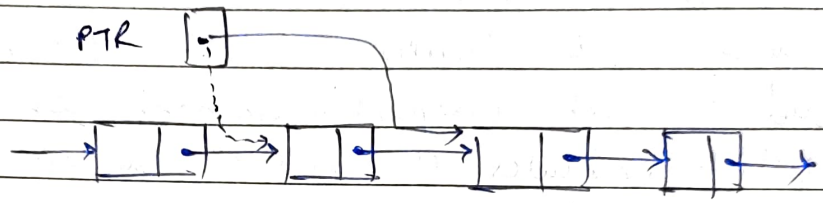
Similarly, we can maintain two linked list (or more) using these two arrays.

Example :- Given in Introduction of linked list.

Traversing a linked list

Here, for traversing algorithm, we used a pointer variable PTR which points to the node. Accordingly, LINK[PTR] points to the next node.

Thus, PTR = LINK[PTR]



PTR = LINK[PTR].

Algorithm :- TRVERSE(INFO, LINK, PTR, START).

1. Set PTR = START
2. Repeat ~~step~~ step 3 & 4 while PTR ≠ NULL.
3. Apply PROCESS to INFO[PTR].
4. Set PTR := LINK[PTR].
5. [End of step 2 loop]
6. Exit.

Algo to find no. of elements in a linked list

1. Set NUM := 0
2. Set PTR := START.
3. Repeat 4 & 5 while PTR ≠ NULL
4. Set NUM := NUM + 1
5. Set PTR := LINK[PTR]
   [End of step 3 loop]
6. Exit.

* Searching a Linked List

LIST is Unsorted

SEARCH (INFO, LINK, START, ITEM, LOC).

1. Set PTR = START.
2. Repeat 3 while PTR ≠ NULL
3. If ITEM = INFO[PTR] then:
   Set LOC = PTR and Exit.
   Else
   Set PTR = LINK[PTR]
   [End of if structure]
   [End of step 2 loop]
4. Set LOC := NULL
5. Exit.

List is Sorted

SEARCHSL (INFO, LINK, START, ITEM, LOC)

1. Set PTR := START
2. Repeat step 3 while PTR ≠ NULL
3. If ITEM > INFO[PTR] then:
   Set PTR := LINK[PTR]
   Else If ITEM = INFO[PTR] then
   set LOC := PTR and Exit.
   Else:
   Set LOC := NULL and Exit
   [End of if structure]
   [End of step 2 loop]
4. Set LOC = NULL
5. Exit.

* Memory Allocation ; Garbage Collection.

⇒ Maintenance of linked list assume possibility of inserting new nodes in lists and hence, some mechanism which provides unused memory space for these new nodes.

⇒ So, some mechanism is required whereby the memory space of deleted nodes becomes available for future use.

⇒ Thus, a special list is maintained having its own pointer, called as "list of available space" or "free storage list" or the "free pool"
We call this list as "AVAIL"

Thus linked list will be denoted by writing -
LIST (INFO, LINK, START, AVAIL).

Ex :-

START = 5

AVAIL = 10

	INFO	LINK
1	E	7
2		6
3	B	11
4	G	12
5	A	3
6		10
7	F	4
8	D	1
9	I	0
10		2
11	C	8
12	H	9

⇒ Suppose some memory space becomes reusable because a node is deleted from list or a list is deleted from program.

⇒ We want this space to be available for future use. So, we immediately reinsert this space into the "free storage list".

⇒ This is what we do when we implement linked list using arrays. However, this method can be too time-consuming for ~~os~~ Operating system, which may choose to use following alternative -

⇒ OS periodically collect all deleted space onto the free storage list. Any technique, which does this collection is called garbage collection.

⇒ It usually takes place in two steps -

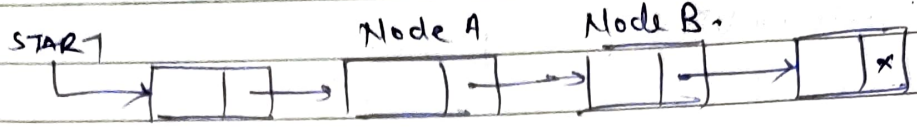
i) First, computer runs through all lists tagging those cells that are currently in use.

ii) Then computer runs through memory, collecting all untagged cells into free storage list.

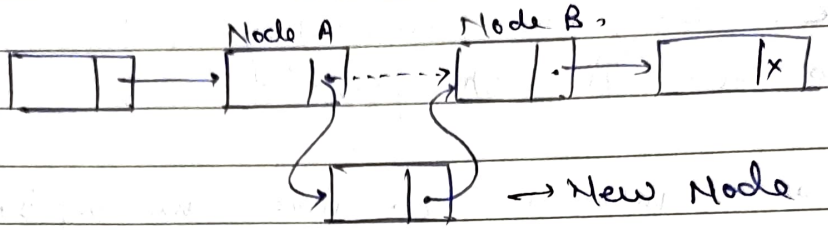
⇒ Garbage collection takes place when there there is little space or no space left in the storage list. Or when CPU is idle.

⇒ Generally speaking, garbage collection is invisible to the programmer.

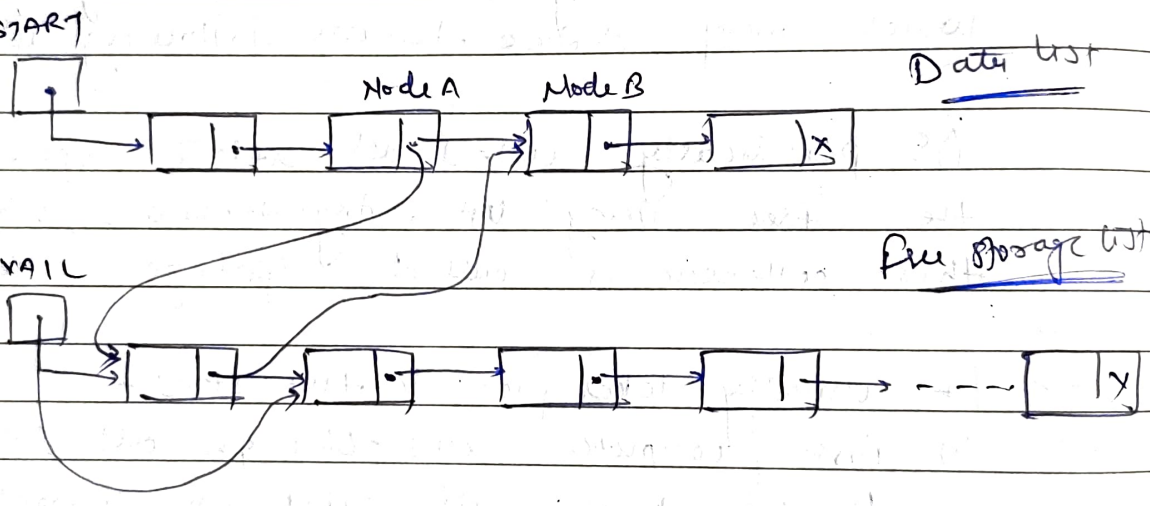
* Insertion into a linked list



Before insertion



After insertion



NEW = AVAIL and AVAIL := LINK [AVAIL]

INFO [NEW] = ITEM.

Removing first node from AVAIL.

Allocating item to new node

This new node is now ready to be inserted at its correct position.

Insert at the beginning.

* INSERT (INFO, LINK, START, AVAIL, ITEM)

① If AVAIL = NULL then:
write: overflow and Exit.

② [Remove first node from avail]
set NEW := AVAIL , AVAIL := LINK [AVAIL].

③ Set INFO [NEW] := ITEM

④ Set LINK [NEW] = START.

⑤ Set START := NEW.

* Insert after given node:
INSERT LOC (INFO, LINK, STAR, AVAIL, LOC, ITEM).

① If AVAIL = NULL : then overflow and Exit

② NEW = AVAIL and AVAIL := LINK[AVAIL].

③ INFO [NEW] = ITEM.

④ If LOC = NULL then : [Insert at first].
LINK [NEW] := START & START = NEW.
Else
⑤ LINK [NEW] = LINK [LOC] and LINK[LOC] = NEW.

⑥ Exit.

Assignment 1

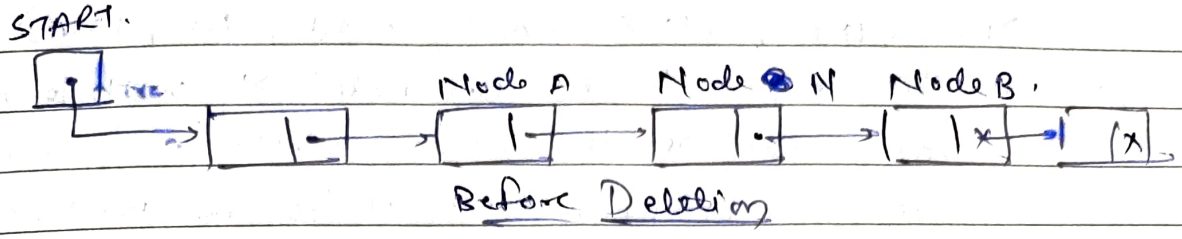
4) Write an algorithm to insert a node into sorted linked list. Assume that linked list is sorted in ascending order.

Hint:- Obtain "LOC" value to get the location where an item is to be inserted. Then call previous algo by passing this loc.

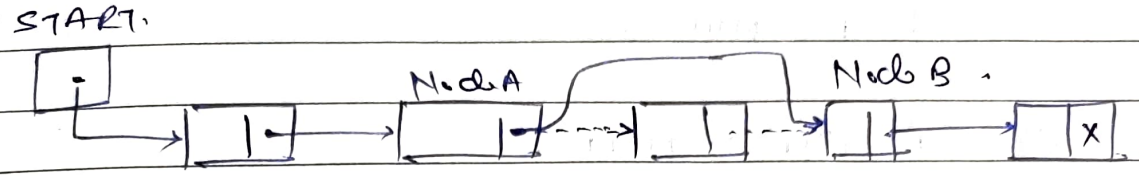
FINDA (INFO, LINK, START, ~~AVAIL~~, ITEM, LOC)

1. If START = NULL then set LOC := NULL & Exit
2. [special case]
   If ITEM < INFO[START] then set LOC := NULL and Exit.
3. Set SAVE := START and PTR := LINK[PTR].
4. Repeat 5 & 6 while PTR ≠ NULL.
5. If ITEM < INFO[PTR] then
   Set LOC := SAVE and Return.
   [End of if].
6. Set SAVE := PTR and PTR := LINK[PTR].
   [End of Step 4 loop].
7. Set LOC := SAVE
8. Exit.

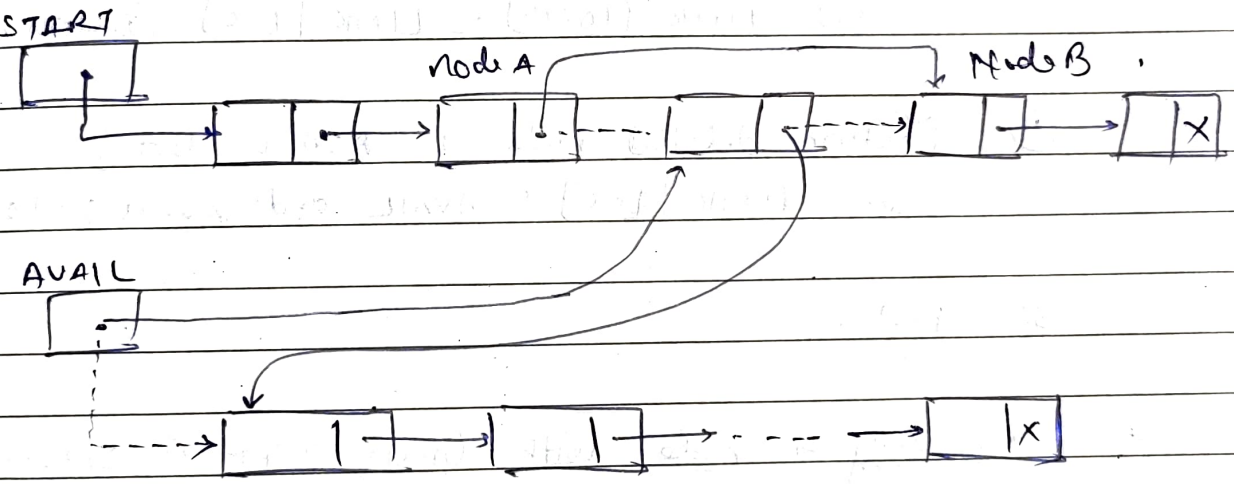
* Deletion from a linked list.



Before Deletion



After Deletion



fw storage list.

LINK [LOC] := AVAIL and AVAIL := LOC.

Add the deleted node into avail and then link node A with node B.

* Deleting a Node following a given node

DEL (INFO, LINK, START, AVAIL, LOC, LOCP)

⇒ This algo deletes the node N with location LOC. LOCP is the location of the node which precedes N or, when N is the first node, LOCP = NULL.

① If LOCP := NULL,
Set START := LINK [START] [Delete 1st node]
Else
Set LINK [LOCP] := LINK [LOC] [Delete node N]

② [Return deleted node to AVAIL list]
Set LINK [LOC] := AVAIL and AVAIL := LOC

③ Exit.

* Deleting a node with Given ITEM of Information

FINDB (INFO, LINK, START, ITEM, LOC, LOCP)

① [List empty] If START := NULL then:
Set LOC := NULL and LOCP := NULL & Return

② [ITEM in first Node]. If INFO [START] = ITEM then
Set LOC := START and LOCP := NULL & Return

③ Set SAVE := START and PTR := LINK [START]

④ Repeat Steps ⑤ and ⑥ while PTR := ≠ NULL.

(5) If INFO[PTR] = ITEM then
Set LOC := PTR and LOCP := SAVE and Return

(6) Set SAVE := PTR and PTR = LINK[PTR]
[End of Step (5) Loop]

(7) Set LOC := NULL (Search Unsuccessful)

(8) Return.

- DELETE (INFO, LINK, START, AVAIL, ITEM)

(1) Call FINDB (INFO, LINK, START, ITEM, LOC, LOCP)

(2) If LOC = NULL then:
Write: ITEM not in LIST and Exit.

(3) [Delete node]
If LOCP = NULL then:
Set START := LINK[START]
Else
Set LINK[LOCP] := LINK[LOC]

(4) Set LINK[LOC] := AVAIL and AVAIL := LOC.

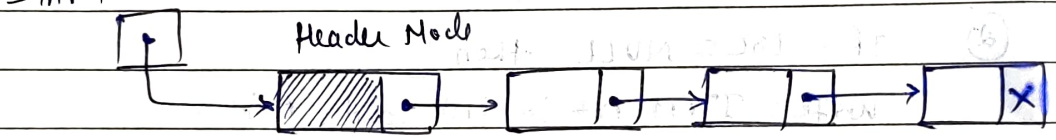
(5) Exit.

* Header Linked List

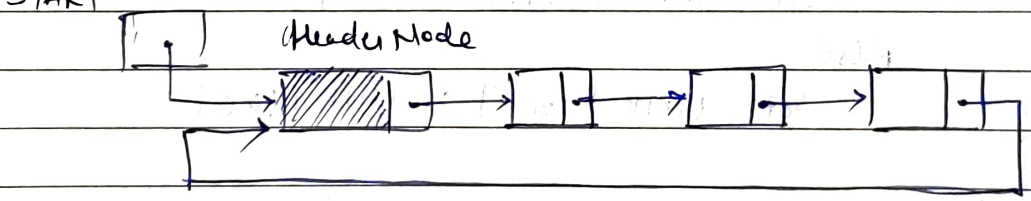
A header linked list is a linked list which always contains a special node, called the "header node" at the beginning of the list.

The following are the two widely used header lists :-

(1) A grounded header list :- It is a header list where the last node contains the null pointer.



(2) A circular header list :- It a header list where the last node points back to the header node



Unless otherwise stated or implied, our header list will always be circular. Accordingly, in such cases, the header node also acts as a sentinel indicating the end of the list.

LINK [START] = NULL ⟶ Grounded list is empty

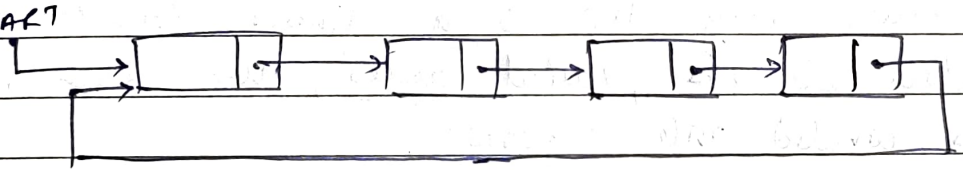
LINK [START] = START ⟶ Circular list is empty.

Example:- Traversing a Circular Header List.

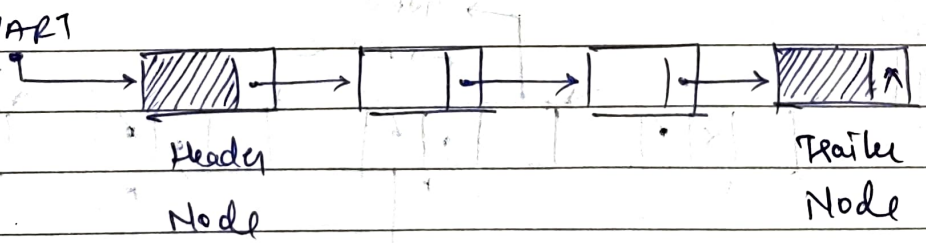
1. Set PTR := LINK[START]
2. Repeat steps 3 & 4 while PTR ≠ START
3. Apply PROCESS to INFO[PTR]
4. Set PTR := LINK[PTR]
5. Exit.

* Variations of Linked List -

1. A Linked List whose last node points back to the first node instead of containing null pointer, is called a circular list.



2. A linked list which contains both a special header node at the beginning of the list and a special trailer node at the end of the list.



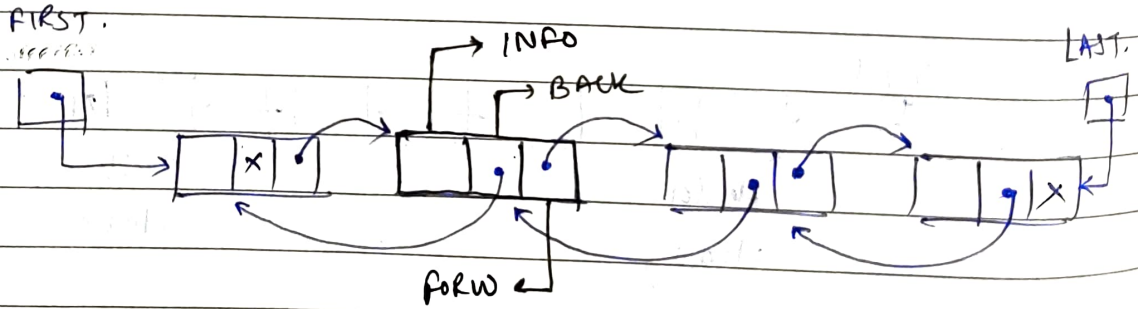
* Two - Way List

It is a list that can be traversed in both the directions. That is, in the forward direction from the beginning of the list to the end, or in the backward direction, from the end of the list to the beginning.

Furthermore, given the location Loc of a node 'N', we have access to both the next node and previous node in the list. That is, one can delete the node without traversing the list.

A two way list is a linear collection of data elements, called 'nodes', where each node is divided into 3 parts :-

1) INFO part -
2) Location of Next node - FORW.
3) Location of previous node - BACK.



The list also requires two list pointers :-

1) FIRST :- points to the first node in the list
2) LAST :- points to the last node in the list.

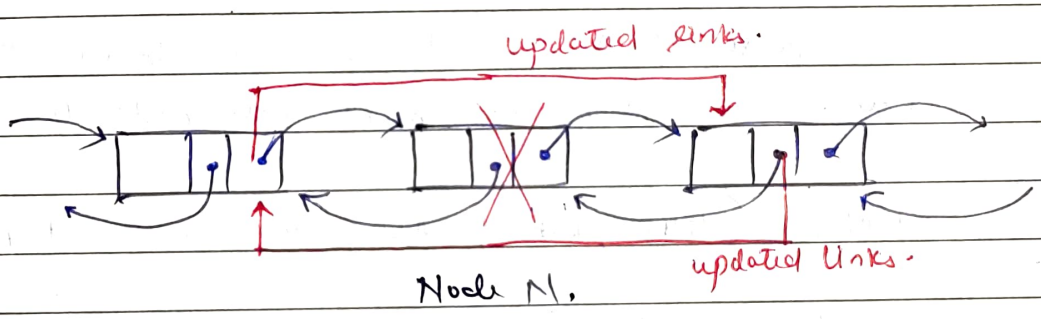
* Operations on Two-way lists.

⇒ **Traversing and Searching**

Both these operations can be performed exactly similar to that of one-way linear list. There is no advantage of having a two-way list for traversing and searching.

→ **Delete :-**

Suppose we want to delete node 'N' with location given as 'LOC' from list -



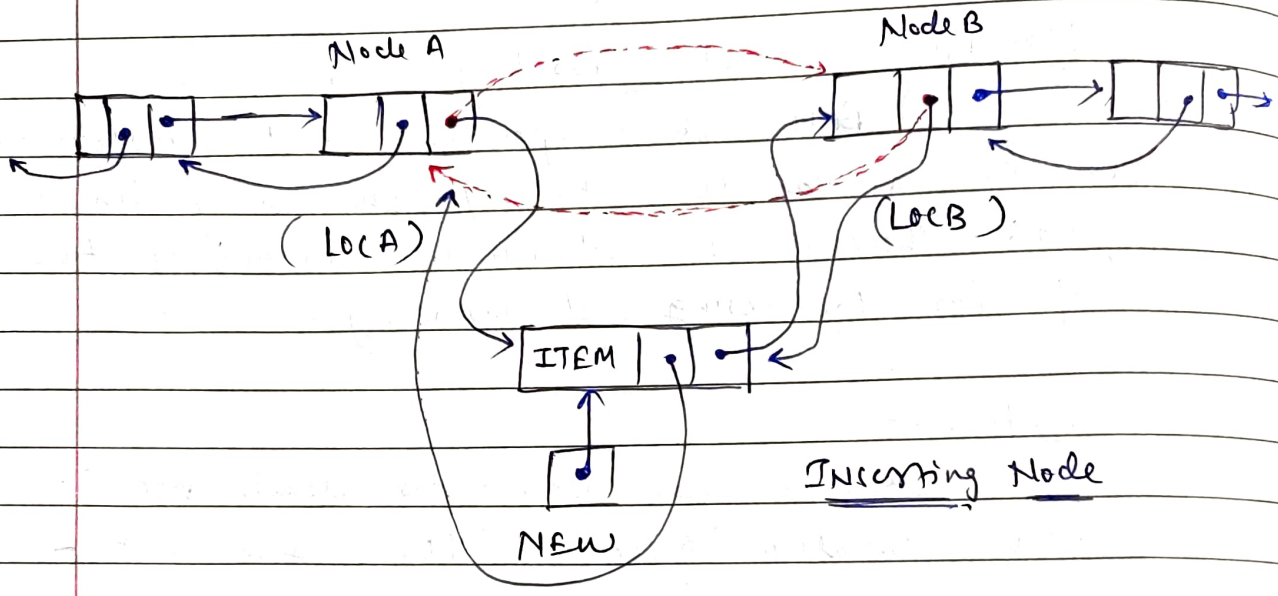
DELETE (INFO, FORW, BACK, START, AVAIL, LOC).

1) [Delete Node],
Set FORW[BACK[LOC]] := FORW[LOC] and
BACK[FORW[LOC]] := BACK[LOC]

2) [Add deleted node to AVAIL list]
Set FORW[LOC] := AVAIL and AVAIL := LOC.

3) Exit.

* Insert a node in two-way list -



Suppose LOCA and LOCB are location of adjacent node. we need to insert new node in between.

INSERT (INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

① [Overflow] If AVAIL := NULL then;
Write: Overflow and Return.

② [Remove node from AVAIL]
Set NEW := AVAIL, AVAIL := FORW[AVAIL]

③ INFO[NEW] := ITEM.

④ [Insert a Node]
Set FORW[LOCA] := NEW, FORW[NEW] := LOCB
BACK[LOCB] := NEW, BACK[NEW] := LOCA

⑤ Exit.

* STACK using Linked List -

PUSH_LINK_STACK (INFO, LINK, TOP, AVAIL, ITEM)

(1) If AVAIL = NULL then
Overflow and Exit

(2) NEW = AVAIL and
AVAIL = LINK[AVAIL]
} Remove first node from AVAIL.

(3) Set INFO[NEW] = ITEM

(4) Set LINK[NEW] = TOP
and TOP = NEW.

(5) Exit.

POP_LINK_STACK (INFO, LINK, TOP, AVAIL, ITEM)

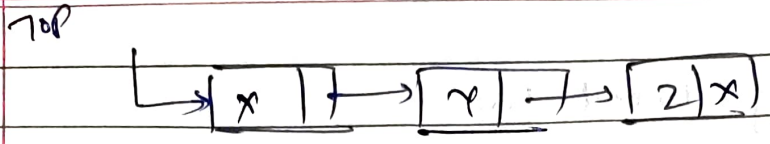
(1) If TOP = NULL then
Underflow and Exit

(2) Set ITEM = INFO[TOP]

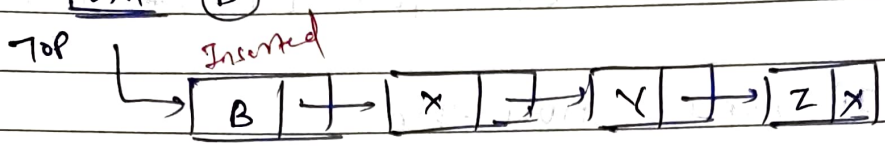
(3) Set TEMP = TOP and
TOP = LINK[TOP]
} Delete node & store it in TEMP.

(4) Set LINK[TEMP] = AVAIL and
AVAIL = TEMP
} Add deleted node to AVAIL.

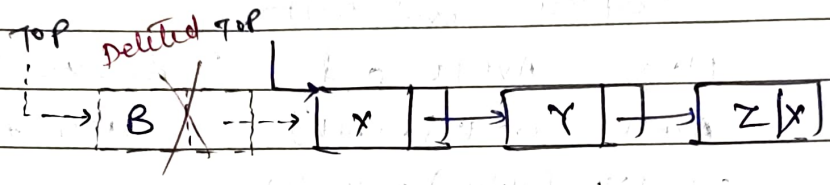
(5) Exit



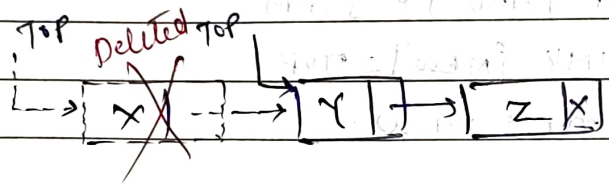
Push (B)



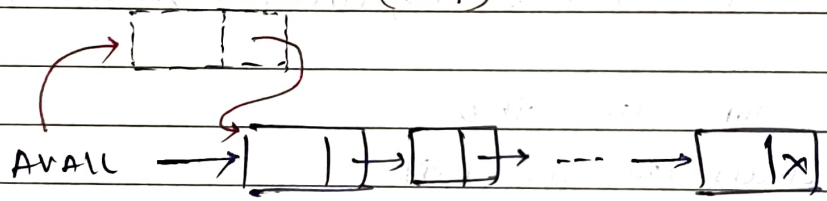
Pop



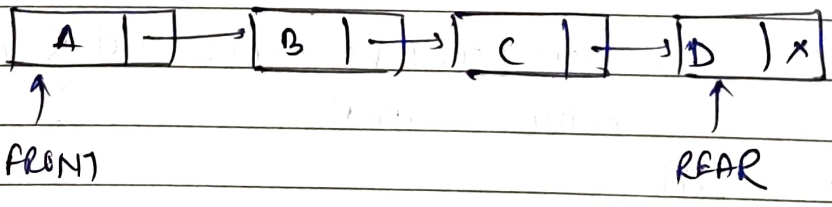
Pop



Add the deleted node to Avail List



Linked list Representation of Queue



Add elements from REAR

Delete elements from FRONT.

LINK_QINSERT (INFO, LINK, FRONT, REAR, AVAIL, ITEM)

1. If AVAIL = NULL then overflow and Exit
2. Set NEW = AVAIL
   AVAIL = LINK[AVAIL]
   } Remove first node from AVAIL.
3. Set INFO[NEW] = ITEM and
   LINK[NEW] = NULL
   } Copy item into new node
4. If FRONT = NULL then
   FRONT = REAR = NEW.
   Else.
   LINK[REAR] = NEW and REAR = NEW.
5. Exit.

LINK_QDELETE (INFO, LINK, FRONT, REAR, AVAIL, ITEM)

1. If FRONT = NULL then: Underflow and Exit.
2. Set TEMP = FRONT
3. ITEM = INFO[TEMP]
4. FRONT = LINK[TEMP] → Delete node
5. LINK[TEMP] = AVAIL and
   AVAIL = TEMP.
   } Add deleted node to AVAIL.
6. Exit.