Traversal (Linked List)

Procedure INSERT (INFO, LINK, START)

1) check for underflow condition
   If (START = NULL), then write "Empty List"
2) Initialization of traversal
   T = start
3) Repeat step 4 while LINK[T] ≠ NULL
4) Write INFO[T]
   T = LINK[T]
5) Return.

Insertion

Procedure INSFRONT (INFO, LINK, START, AVAIL, X)

(1) check for the overflow Condition
   If AVAIL = NULL, then write "Memory overflow"
(2) Allocate memory
   AV ⇐ NODE
(3) Insert the new item at the avail node.
   INFO[AV] = x
   LINK[AV] = START
   START = AV
(4) Return.

Procedure INLAST (INFO, LINK, START, X)

(1) Create a new node called avail.
   AV ⇐ NODE
(2) Insert new item x at the info of AV
   INFO[AV] = X
   LINK[AV] = NULL
(3) If (START = NULL) Then START = AV Return

④ If (START ≠ NULL) Then
T = START
While LINK[T] ≠ NULL
T = LINK[T]

⑤ LINK[T] = AV

6 Return

Procedure INLAST (INFO, LINK, START, X)

1) Create a new node called AVAIL
AV ⇐ NODE
2) Insert new item X at the info of AV
INFO[AV] = X
LINK[AV] = NULL
3) If (START = NULL) Then START = AV Return
4) Traversal of linked list to last position
If (START ≠ NULL) Then
T = START
while LINK[T] ≠ NULL
T = LINK[T]
5) LINK[T] = AV
6) Return.

Insert after nth node

Procedure INSAFTERN (START, N, X)

1) Create a new node called avail.
AV ⇐ NODE
2) Insert the new node ~~after~~ at the avail
INFO[AV] = X
3) If (START = NULL), Then write "empty list"

4) T = START

5) C = 1

6) Check the Position while (C = N) and LINK [T] ≠ NULL

T = LINK [T]

C = C + 1 (Increment of loop counter)

7) If (LINK [T] = NULL and C ≠ N)

then write " the node is absent" (Checking the Presence of nth node)

else

P = LINK [T]

8) Insertion of the new node.

LINK [T] = AV

9) LINK [AV] = P

10) Return.

Insertion of a new node after a node that contains the value D

1) Create a new node.

Av ← NODE

2) Insert the new item x at the [INFO [AV]]

INFO [AV] = X

3) If (START = NULL) Then write " empty list".

4) Traversal till the link node contains the value of D.

T = START

While LINK [T] ≠ NULL and INFO [T] ≠ D

T = LINK [T]

5) (Checking the Presence of the node D.)

If LINK [T] = NULL and INFO [T] ≠ D, Then the node is absent

else P = LINK [T]

LINK [T] = AV

LINK [AV] = P

6) Return.

DELETE FIRST (START, INFO, LINK, X)

1) If (START = NULL) then write "Empty list".
   Return (Checking underflow condition)
2) Assign the start node to a variable T.
   T = START
3) Delete the front node.
   START = LINK[START]
4) Return T.

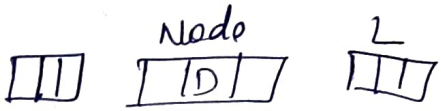
DELETE LAST - DELLAST (START, INFO, LINK, X)

1) If (START = NULL) Then write "Empty List".
   (Check the underflow condition)
2) [Traversal to last]
   If LINK[START] = NULL, then P = START
   START = NULL
   Return info.
   (If there is only one node)
3) Else
   PREVIOUS = START
   T = LINK[START]
4) Traversal till the last node
   PREVIOUS = T
   T = LINK[T]
5) Delete the last node
   LINK[PREV] = NULL
6) Return.

Double Linked List

Procedure LEFTMOST (D)

1) Allocate the memory for new node.
   NODE ⇐ AVAIL
   INFO [NODE] = D
2) Update the new node.



   L LINK [NODE] = NULL
   R LINK [NODE] = L
3) L LINK [L] = NODE
4) L = NODE
5) Return.

Procedure INSRIGHTMOST (D)

1) NODE ⇐ AVAIL
   INFO [NODE] = D
   (Allocate memory for new node)
2) (Update the new node)
   L LINK [NODE] = R
   R LINK [NODE] = NULL

3) R LINK [R] = NODE
4) R = NODE
5) Return.

Insert to the right of a node which contains the value D.

1) If L=R = NULL, Then "Empty list".

2) AV ⇐ NODE
   INFO [AV] = X
3) While INFO [T] ≠ D and R LINK [T] ≠ NULL
   T = R LINK [T]
4) If INFO [T] ≠ D and R LINK [T] = NULL, Then the desired node

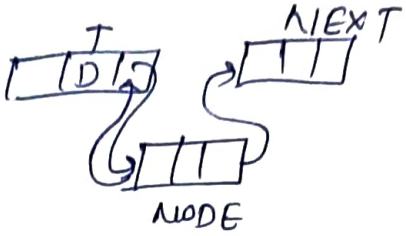
absent

5) i) If INFO[T] = D and L=R

Then L LINK [NODE] = T

R LINK [T] = NODE

R = NODE



(ii) If INFO[T] = D and L ≠ R i.e L = T

Then NEXT = R LINK [T]

R LINK [T] = NODE

L LINK [NODE] = T

R LINK [NODE] = NEXT

L LINK [NEXT] = NODE

6) Return.

DELLEFT (L,R)

Leftmost node

1) If L=R=NULL, Then write the list is empty.

2) (Underflow condition checking)

If L=R Then T=L

L = R = NULL

Return

3) Else T = L

L = R LINK [L]

L LINK [L] = NULL

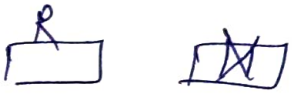
Return T

Rightmost node

1) If L=R=NULL, Then write empty list

2) If L=R Then T=L

L=R=NULL



3) Else T = R

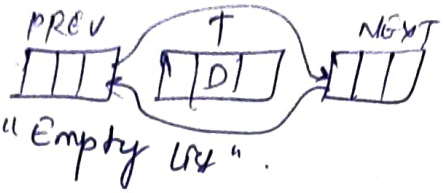
R = L LINK [R]

R LINK [R] = NULL

4) Return

Deletion of node contains value D.

1) (Check the underflow condition) If (L=R= NULL). then write "Empty List".



2) If L=R and INFO[L] = D then T=L
   L=R = NULL Return.
3) While INFO[T] ≠ D and RLINK[T] ≠ NULL
   T = RLINK[T]
4) ~~While~~ If INFO[T] ≠ D and RLINK[T] = NULL
   then the desired node is absent.
5) Else
   INFO[T] = D
   then PREV = LLINK[T]
   NEXT = RLINK[T]
   RLINK[PREV] = NEXT
   LLINK[NEXT] = PREV

Return T.

Deletion of node after the node containing the value D

1) Check the underflow condition.
   If (L=R=NULL) then write "Empty List".
2) If L=R and INFO[L] = D Then T=L
   L=R = NULL Return.
3) While INFO[T] ≠ D and RLINK[T] ≠ NULL
   T = RLINK[T]
4) If INFO[T] ≠ D and RLINK[T] = NULL
   then the desired node is absent.
5) Else

   INFO[T] = D
   ~~RLINK[T]~~ NEXT = RLINK[T]
   NEXT NEXT = RLINK[NEXT]

R LINK [T] = NEXT NEXT
L LINK [NEXT NEXT] = T
Return T.

Delete the node before the node containing the value D.

1) Check the underflow condition.
   If (L=R = NULL) Then Write "Empty List".
2) If L=R and INFO[L] = D Then T=L
   L = R = NULL Return.
3) While INFO [T] ≠ D and R LINK [T] ≠ NULL
   T = R LINK [T]
4) If INFO [T] ≠ D and R LINK [T] = NULL
   Then the desired node is absent.
5) Else

   INFO [T] = D
   PREV = L LINK [T]
   PREV PREV = L LINK [PREV]
   L LINK [T] = PREV PREV
   R LINK [PREV PREV] = T
   Return.

Procedure Binary Tree Traversal Recursive algorithm

① RINODRER (T) : LDR

```
If ( T ≠ NULL) then
{
 Call  R INORDER (L CHILD [T])
 PRINT  INFO [T]
 Call  R INORDER ( R CHILD [T])
}
```

② R PREORDER(T) : DLR

```
If ( T ≠ NULL) , then
{
   PRINT INFO [ T]
   Call  RPREORDER ( L CHILD [T])
   Call  RPREORDER  (R CHILD [T])
}
```

③ RPOSTORDER(T) : LRD

```
If ( T ≠ NULL) , Then
{
  Call  RPOSTORDER (L CHILD [T])
  Call  RPOSTORDER (R CHILD [T])
  PRINT  INFO [ T]
}
```