Step 14:- Let link of S = link of ptr
Step 15:- Let V = Data of ptr.
Step 16:- ~~Declare~~ Release ptr.
Step 17:- Return with V

Doubly linked list :-

A doubly linked list is a also a collection of nodes scattered within the memory and connected together logically by holding the address of consequence nodes. The differente between SLL and DLL is that it can be traversed in two ways from the beginning this two way traversal provides a lot of advantages over the SLL It reduces a remarkable processing time particularly when we have to search a value and have to append a node. for the two way traversal, the nodes of DLL holds before and coming after. The nodes of DLL have at least three parts - the datapart, the previous link pointer and the next link pointer. The previous

link of the first node has a null value while the next link of the last node has a null value. To traverse the list we need to keep account the address of both first and last nodes. for which we declare two pointers commonly called as start-end or head-tail.

* Creation of DLL :-

A DDLL is created by difining the structure of node which have three members - data, P-link, N-link. After that two variables are declared with defined datatype and assigned with null to make the empty list. The algorithm is as given below-

Step 1:- Create DDLL
Step 2:- Define structure node with P-link and N-link as pointer of node type.
Step 3:- Declare pointers start and end of node type
Step 4:- Let start = end = Null

Step 5:- let end = Null

Step 6:- end

* Insertion of first node - To insert a node as the first node in a DLL as usual we need to allocate memory for the node by holding the address in a pointer. The value is assigned to the data member and the node is connected to the start pointer. But with the just node of DLL we have to assign the prierious link with null pointer and to check either it is the only one node then it have to be link with the end pointer also. It follows the algorithm given below -

Step 1:- InsFirst (v)

Step 2:- Declare pointer ptr of node type

Step 3:- Allocate memory to ptr

Step 4:- Let Data of ptr = v

Step 5:- If start = Null then

5.A :- let P-link of ptr = Null
5.B :- let N-link of ptr = Null
5.C :- let start = ptr
5.D :- let end = ptr

Step 6 :- otherwise

6.A :- let P-link of ptr = Null
6.B :- let N-link of ptr = start
6.C :- let P-link of start = ptr
6.D :- let start = ptr

step 7 :- end of If

step 8 :- end of function.

* Insertion of last node - A node at the end of a DLL can be inserted by allocating memory for the node and then attaching it with the end pointer and current last node. But here we also need to consider that either it would be the only one node of a list. If it is we need to connected with the start also.

The algorithm is as given below —

Step 1:- Start insLast(v)

Step 2:- Declare pointer ptr of node type

Step 3:- Allocate memory to ptr.

Step 4:- Let data of ptr = v

Step 5:- If start = Null then,

5.A : Let P.Link of Ptr = Null
5.B : Let N Link of Ptr = Null
5.C : Let start = Null
5.D : Let end = Null.

Step 6:- Otherwise

6.A : Let N link of Ptr = Null
6.B : Let P link of Ptr = end
6.C : Let N link of end = Ptr
6.D : Let end = Ptr

Step 7:- end of If

Step 8:- end

* Insertion of node at a given position

In general to insert a node at a given position we need to traverse to the node comming before the given position and adjust the new node with the link of traversed node. But with the doubly linked list we should consider that to get the position from where we will traverse either from the start or from the end. In addition we need to check for the empty list, for not founding the position and probability of position to be one or last. This insertion follows the given algoritm.

Step 1:- Start Ins Pos (v, p)
Step 2:- Declare pointer ptr and S of node type and counter x = 1
Step 3:- If P > count of node +1 OR start = Null then

3.A: Print "Position not found or list empty"
3.B: Return
Step 4:- end of if
Step 5:- Allocate memory to ptr
Step 6:- Let data of Ptr = v
Step 7:- If P = 1 then,
7.A: call insfirst(v)
7.B: Return
Step 8:- end of if
Step 9:- If P count+1 then
9.A:- Call insdast(v)
9.B:- Return
Step 10:- end of if
Step 11:- If P < count/2 then,
11.A:- Let S = start
11.B:- while x < P-1
11.C:- Let S = N link of S
11.D:- let x = x+1
11.E:- end of while
11.F:- Let N link of Ptr = N link of S
11.G:- Let P Link of Ptr = S
11.H:- let N Link of S'P link = Ptr
11.I:- Let N link of S = Ptr.
Step 12:- else
12.A:- Let S = end
B:- let x = count
C:- while x > P
D:- Let S = P link of S

E :- let x = x - 1
F :- end of while
G :- let P link of Ptr = P-link of S
H :- end of while
H :- let N.link of Ptr = S
I :- let P link of S'd N link = Ptr
Step13 :- end of If
Step14 :- Return

* Deletion of 1st node — when deleting the first node we need to consider that either the list is empty as well as the list has a single node if the nodes is the only one node we need to assign both the start and end pointers with null to make the list empty. Otherwise, the start is assigned with the next pointer of the node to be deleted. The algorithm is as given below:

Step 1 :- DelFirst () and variable v
Step 2 :- Declare pointer ptr of node type
Step 3 :- If start = Null then,
3.A : Print list is empty
3.B : Return
Step 4 :- end of if
Step 5 :- Let Ptr : start

Step 6 :- let v = data of Ptr
Step 7 :- if start = end then;
7.A :- let start = Null
7.B :- let end = Null
Step 8 :- else
8.A :- let start = N link of Ptr
8.B :- let P link of start = Null
Step 9 :- end of if
Step 10 :- Deallocate Ptr
Step 11 :- Return with v.

* Deleting the last node from DLL :-
As like the deletion of first node we need consider about the empty list and presence of a single node in the list when we delete the last node of the DLL. If both these conditions are false we deallocate the node after assigning the end with the second node and assigning the next link of second last node with null. The following algorithm is used.

Step 1 :- Start Del last ()
Step 2 :- Declare pointer of node type and var = v
Step 3 :- If start = Null then
3.A :- Print empty List

3.B:- Return with Zero
Step 4:- end of if
Step 5:- let ptr = end
Step 6:- if start = end then,
6.A:- let start = Null
6.B:- let end = Null
Step 7:- else
7.A:- let end = P link of ptr
7.B:- let N link of end = Null
Step 8:- end of if
Step 9:- let v = data of ptr
Step 10:- Deallocate ptr
Step 11:- Return with v

Deletion of node from a particular position:—

To delete a node from a particular position, we preceed with the checking for the empty list and the presence of that number of node. After then we check the position either it is first or last and call the corresponding function. If all the conditions go false then, we traverse to the node and connect the nodes comming before and after before deallocating the node. The algorithm is as follows:-

Step 1:- Del Pos(P)

Step 2:- Declare pointer ptr and tmp as node type and the variable v and n=p.

Step 3:- If start = Null or C<P then,

3.A:- Print list empty or position not found

3.B:- Return with 0

Step 4:- If P=1 then,

4.A:- v = call del 1st

4.B:- Return with v

Step 5:- end of if

Step 6:- If P=C

6.A:- v = call del last

6.B:- Return with v

Step 7:- end of if

Step 8:- If C, g>P then,

8.A:- Let Ptr = start

8.B:- while (n<P-1)

8.C:- Let Ptr = N link → Ptr

8.D:- Let n = n+1

8.E:- end of while

Step 9:- else

9.A:- Let Ptr = end

9.B:- Let n = C

9.C:- while n>P+1

9.D:- Let Ptr = P link → Ptr

9.E:- Let n = n-1

9.f:- end of while

9.G :- tmp = P link → Ptr
9.H :- N link of tmp's P link = Ptr
9.I :- P link of ptr = P link of tmp.

Step 10 :- end of if
Step 11 :- v = data of tmp
Step 12 :- Deallocate tmp
Step 13 :- let C = C - 1
Step 14 :- Return with v