DOUBLY LINKED LISTS

Definition

- A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence.
- Therefore, it consists of three parts—data, a pointer to the next node, and a pointer to the previous node as shown in Figure

	NOWEE	RIA		
START				
\rightarrow X 1 \rightarrow	2	3 →	4 →	5 X
The structure of a doubly linked list ca	n be given as, struc	t node	SHALL I	
		\mathbb{R}	121	
struct node *prev; int data;			٩ ٩	
struct node *next;				
};	ULAM, KANYI	KUM		

- The PREV field of the first node and the NEXT field of the last node will contain NULL.
- The PREV field is used to store the address of the preceding node, which enables us to traverse the list in the backward direction.
- Doubly linked list calls for more space per node and more expensive basic operations.
- However, a doubly linked list provides the ease to manipulate the elements of the list as it maintains pointers to nodes in both the directions (forward and backward).
- The main advantage of using a doubly linked list is that it makes searching twice as efficient.

Memory representation of a doubly linked list

1	DATA	PREV	NEXT
□ →1	Н	-1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	0	7	-1

- Variable START is used to store the address of the first node.
- In this example, START = 1, so the first data is stored at address 1, which is H.
- Since this is the first node, it has no previous node and hence stores NULL or -1 in the PREV field.
- We will traverse the list until we reach a position where the NEXT entry contains –1 or NULL. This denotes the end of the linked list.
- When we traverse the DATA and NEXT in this manner, we will finally see that the linked list in the above example stores characters that when put together form the word HELLO.

Inserting a New Node in a Doubly Linked List

Case 1: The new node is inserted at the beginning.

Case 2: The new node is inserted at the end.

Case 3: The new node is inserted after a given node.

Case 4: The new node is inserted before a given node.

Case 1: The new node is inserted at the beginning.

We want to add a new node with data 9 as the first node of the list. Then the following changes will be done in the linked list.

Х	1	\leftarrow	7	\leftarrow	3	\checkmark	4	\leftarrow	2)	(
STAF Allo		memory	for the	new node	and in	itialize	its	DATA part	to 9 a	nd PREV	field to	NULL
Х	9											
	the list		before	the STAR	T node.	Now the	new	node beco	mes the	first	node of	
X	9	 	1	\rightarrow	7	\rightarrow	3	\rightarrow	4	₹	2 X]
STAF	RT	1993 T										

Algorithm to insert a new node at the beginning

		10 M (10		
	Step	1:	IF AVAIL = NULL	
10			Write OVERFLOW	ŝ
1			Go to Step 9	
14			[END OF IF]	F
~	Step	2:	SET NEW_NODE = AVAIL	1
ч.	Step	3:	SET AVAIL = AVAIL -> NEXT	ľ
0	Step	4:	SET NEW_NODE -> DATA = VAL	
0	Step	5:	SET NEW_NODE -> PREV = NULL	
	Step	6:	SET NEW_NODE -> NEXT = START	
Z	Step	7:	SET START -> PREV = NEW_NODE	
7	Step	8:	SET START = NEW_NODE	ł
	Step	9:	EXIT	Ļ
		10.00		

- In Step 1, we first check whether memory is available for the new node.
- If the free memory has exhausted, then an OVERFLOW message is printed.
- Otherwise, if free memory cell is available, then we allocate space for the new node.
- Set its DATA part with the given VAL and the NEXT part is initialized with the address of the first node of the list, which is stored in START.
- Now, since the new node is added as the first node of the list, it will now be known as the START node, that is, the START pointer variable will now hold the address of NEW NODE.

Case 2: The new node is inserted at the end.

We want to add a new node with data 9 as the last node of the list. Then the following changes will be done in the

linked list.

x	$1 \xrightarrow{} 7 \xrightarrow{} 3 \xrightarrow{} 4 \xrightarrow{} 2 X$
ST	ART
	locate memory for the new node and initialize its DATA part to 9 and its XT field to NULL.
	9 X
Та	ke a pointer variable PTR and make it point to the first node of the list.
x	$1 \xrightarrow{} 7 \xrightarrow{} 3 \xrightarrow{} 4 \xrightarrow{} 2 \times$
ST	ART, PTR
	ve PTR so that it points to the last node of the list. Add the new node after the de pointed by PTR.
x	$1 \xrightarrow{} 7 \xrightarrow{} 3 \xrightarrow{} 4 \xrightarrow{} 2 \xrightarrow{} 9 X$
ST	ART PTR
gorithm to	insert a new node at the end
0	
	Step 1: IF AVAIL = NULL
	Write OVERFLOW
	Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 11 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET NEW_NODE -> NEXT = NULL Step 6: SET PTR = START
	[END OF IF]
	Step 2: SET NEW_NODE = AVAIL
	Step 3: SET AVAIL = AVAIL -> NEXT
	<pre>Step 4: SET NEW_NODE -> DATA = VAL</pre>
	Step 5: SET NEW_NODE -> NEXT = NULL
	Step 8: SET PTR = PTR -> NEXT [END OF LOOP]
	Step 9: SET PTR -> NEXT = NEW_NODE
	Step 10: SET NEW NODE -> PREV = PTR
	Step 11: EXIT

- In Step 6, we take a pointer variable PTR and initialize it with START.
- In the while loop, we traverse through the linked list to reach the last node.
- Once we reach the last node, in Step 9, we change the NEXT pointer of the last node to store the address of the new node.
- Remember that the NEXT field of the new node contains NULL which signifies the end of the linked list.
- The PREV field of the NEW_NODE will be set so that it points to the node pointed by PTR (now the second last node of the list).

Case 3: The new node is inserted after a given node.

We want to add a new node with value 9 after the node containing 3.

	$X 1 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 2 X$
	START
	Allocate memory for the new node and initialize its DATA part to 9.
	9
	Take a pointer variable PTR and make it point to the first node of the list.
	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$
	START, PTR
	Move PTR further until the data part of PTR = value after which the
	node has to be inserted.
	X 1 7 3 4 2 X START PTR
	Insert the new node between PTR and the node succeeding it.
	$X 1 \rightarrow 7 \rightarrow 3 4 \rightarrow 2 X$
	START
Algorithm	to insert a new node after a given node
	Step 1: IF AVAIL = NULL
	Write OVERFLOW
	Go to Step 12
	[END OF IF]
	Step 2: SET NEW_NODE = AVAIL
	Step 3: SET AVAIL = AVAIL -> NEXT
	<pre>Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET PTR = START</pre>
	Step 5: SET FIX - START Step 6: Repeat Step 7 while PTR -> DATA != NUM
	Step 7: SET PTR = PTR -> NEXT
	[END OF LOOP]
	Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
	Step 9: SET NEW_NODE -> PREV = PTR
	Step 10: SET PTR -> NEXT = NEW_NODE
	Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
	Step 12: EXIT

- In Step 5, we take a pointer PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list. In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM.
- We need to reach this node because the new node will be inserted after this node.
- Once we reach this node, we change the NEXT and PREV fields in such a way that the new node is inserted after the desired node.

Case 4: The new node is inserted before a given node.

We want to add a new node with value 9 before the node containing 3.

	x	1		\rightarrow	10	7	+	>	3	-	\rightarrow		4	2	\rightarrow		2	х	
	STAR	Т		83 SH							93 - 94				18. J				
	A110	cat	e me	mory	for	the	new	node	and	init	iali	ze	its	DATA	A par	t to	9.		
		9																	
	Take	a	poin	ter v	varia	able	PTR	and r	nake	it p	oint	to:	the	fir	st n	ode	of 1	the list	
	х	1		\rightarrow	3	7	4	*	3	ļ	\rightarrow		4		\rightarrow		2	х	
	STAR			13 NF		Xir	-13	10	100		2.9	8	-		, L	- 2			
								now p							e dat	a is	s equ	ual	
	1		vaiu	e bei	10	. 1		ne noo	0.3	s to	b be	ins	erte						
	X	1		<u> </u>	8	7	+	-	3		~		4	1	<u> </u>		2	x	
	STAR Add		new	node	e in	betw	veen	the r	node	poir	nted	by I	PTR	and	the	node	e pro	eceding	it.
	X	1		~	0	7	- 05		3	Ţ	\rightarrow		4		\rightarrow		2	х	
	STAR	т		213 SH-	30	4	•	¥1	PTR		3 J.	i.			- L	2.1	12		
						ſ	9	ə [1										
	V	1	1	→		-		→		-	→		2	-	→		4	_ →	
	x	1		<		7	_ ←		9		< _	8	3		← [4		2 X
	STAR	Т	1				11		6	2		2	Æ		0.1		11	TO L	
				Z					1.1		(梁)			\mathbb{O}^{n}	H١			\sim 1	
Algorithm	to in	sert	: a n	ew n	ode k	befo	re a j	given	node	2	靜精		Å				/	ġ/	
				٠. ١	Ste	ep 1	: 1	AVA	IL =	NU	LL							17	
						85		Wr	ite	OVE	RFLO	W							
				1	1		10-110		to		p 12							1	
					C 1			ND OF	1.										
					1			ET NE	1000									r	
				_	10000	1.1		ET AV											
						0.5		ET PT				IA	- VA	41					
						10		epeat				le	PTR	-> D	ATA	!=	NUM		
						ep 7		100	TPT	Second					0.000000	Stocard	10.70.8		
						100		ND OF											
					Ste	ep 8	: SI	ET NE	W_NO	DE -	> NE	XT	= P1	٢R				ΒV	
			1		Ste	ep 9	: 51	ET NE	W_NO	DE -	> PR	EV	= P1	rr –	> PRE	V			
					Ste	ep 1	0: 9	SET P	TR ->	PRE	EV =	NEI	W_NC	DE				1	
				-		1		SET P	TR -	-> PI	REV -	-> N	EXT	= N	IEW_N	ODE			-
					Ste	ep 1	2: 1	EXIT											

- In Step 1, we first check whether memory is available for the new node.
- In Step 5, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list.
- In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM.

- We need to reach this node because the new node will be inserted before this node.
- Once we reach this node, we change the NEXT and PREV fields in such a way that the new node is inserted before the desired node.

Deleting a Node from a Doubly Linked List

In this section, we will see how a node is deleted from an already existing doubly linked list

Case 1: The first node is deleted.

Case 2: The last node is deleted.

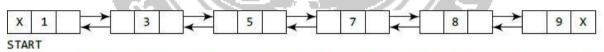
Case 3: The node after a given node is deleted.

Case 4: The node before a given node is deleted.

Case 1: The first node is deleted.

When we want to delete a node from the beginning of the list, then the following changes will be done in the linked

list



Free the memory occupied by the first node of the list and make the second node of the list as the START node.



START

Algorithm to delete the first node SERVE OPTIMIZE OUTSPREE

Step 1: IF START = NULL Write UNDERFLOW Go to Step 6 [END OF IF] Step 2: SET PTR = START Step 3: SET START = START -> NEXT Step 4: SET START -> PREV = NULL Step 5: FREE PTR Step 6: EXIT

- In Step 1 of the algorithm, we check if the linked list exists or not.
- If START = NULL, then it signifies that there are no nodes in the list and the control is transferred to the last statement of the algorithm.
- However, if there are nodes in the linked list, then we use a temporary pointer variable PTR that is set to point to the first node of the list. For this, we initialize PTR with START that stores the address of the first node of the list.
- In Step 3, START is made to point to the next node in sequence and finally the memory occupied by PTR (initially the first node of the list) is freed and returned to the free pool.

Case 2: The last node is deleted.

We want to delete the last node from the linked list, then the following changes will be done in the linked list.

7 <u>00 - 0000 - 000</u>										- C.2
X 1	\rightarrow	3	*	5	*	7	\checkmark	8		9 X
START							-	A 384	-	
Take a po	inter var	iable	PTR tha	at point	ts to the	first	node of	the lis	t.	
X 1	\rightarrow	3	\rightarrow	5	*	7	*	8		9 X
START, PTR		h- 52		- Con - 18				92		20 20 0
Move PTR	so that i	t now	points	to the	last node	e of th	ne list.			
X 1	\rightarrow	3	\rightarrow	5	\rightarrow	7	\rightarrow	8		9 X
START										PTR
		S	by the	node po	pinted by	PTR ar	d store	NULL in	NEXT fi	eld of
X 1	\rightarrow	3	\rightarrow	5	*	7	\rightarrow	8)	K	
START		_								
to delete t	he last no	de						<	2	
			IF ST	ART =	NULL			<	2	2
			IF ST W		NULL	W		~	-	•
			W	rite l		W		~		,
		2p 1:	W	rite L o to S	INDERFLO	W				,
	Ste	ep 1:	W	rite l o to S F IF]	INDERFLO	W				,
	Ste	ep 1: ep 2:	W G END O SET P	rite U o to S F IF] TR = S	INDERFLO		-> NEXT	!= NUI		,
	Ste Ste	ep 1: ep 2:	W G END O SET P Repea	rite U o to S F IF] TR = S t Step	INDERFLO Step 7	e PTR		!= NUI		,
	Ste Ste	2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2	W G END O SET P Repea	rite U o to S F IF] TR = S t Step ET PTF	NDERFLO Step 7 START 0 4 whil R = PTR-	e PTR		!= NUI		,
	Ste Ste Ste	ep 1: ep 2: ep 3: ep 4:	W G END O SET P Repea S [END O	rite U o to S F IF] TR = S t Step ET PTF F LOOF	NDERFLO Step 7 START 0 4 whil R = PTR-	e PTR -> NEX	Г	!= NUI	u	,
	Ste Ste Ste	2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2: 2	W G END O SET P Repea S [END O	rite U o to S F IF] TR = S t Step ET PTF F LOOF TR -> F	NDERFLO Step 7 START 0 4 whil R = PTR - 2]	e PTR -> NEX	Г	!= NUI	u	
	START Take a po X 1 START,PTR Move PTR X 1 START Free the its prece	START Take a pointer var X 1 START, PTR Move PTR so that i X 1 START Free the space occu its preceding node	START Take a pointer variable X 1 3 START,PTR Move PTR so that it now X 1 3 START Free the space occupied its preceding node.	START Take a pointer variable PTR that X 1 3 5TART, PTR Move PTR so that it now points X 1 3 5TART Free the space occupied by the its preceding node.	START Take a pointer variable PTR that point X 1 3 5 START, PTR Move PTR so that it now points to the X 1 5 START Free the space occupied by the node points its preceding node.	START Take a pointer variable PTR that points to the X 1 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	START Take a pointer variable PTR that points to the first X 1 3 5 7 START, PTR Move PTR so that it now points to the last node of th X 1 5 7 START Free the space occupied by the node pointed by PTR ar its preceding node.	START Take a pointer variable PTR that points to the first node of X 1 3 5 7 START, PTR Move PTR so that it now points to the last node of the list. X 1 5 7 START Free the space occupied by the node pointed by PTR and store its preceding node.	START Take a pointer variable PTR that points to the first node of the lis X 1 3 5 7 8 START, PTR Move PTR so that it now points to the last node of the list. X 1 3 5 7 8 START Free the space occupied by the node pointed by PTR and store NULL in its preceding node.	START Take a pointer variable PTR that points to the first node of the list. X 1 3 5 7 8 START, PTR Move PTR so that it now points to the last node of the list. X 1 3 5 7 8 Move PTR so that it now points to the last node of the list. START Free the space occupied by the node pointed by PTR and store NULL in NEXT fi its preceding node.

- In Step 2, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list. The while loop traverses through the list to reach the last node.
- Once we reach the last node, we can also access the second last node by taking its address from the PREV field of the last node.
- To delete the last node, we simply have to set the next field of second last node to NULL, so that it now becomes the (new) last node of the linked list. The memory of the previous last node is freed and returned to the free pool.

Case 3: The node after a given node is deleted.

We want to delete the node that succeeds the node which contains data value 4. Then the following changes will be done in the linked list.

\sim 1	N. 9 9.	V 11		
$X 1 \rightarrow 3 \rightarrow$	4	7	8	9 X
START				
Take a pointer variable PTR and	make it point	to the <mark>fir</mark> st nod	e of the list.	
$X 1 \rightarrow 3 \rightarrow 1$	4	7	8	9 X
START, PTR				
Move PTR further so that its da	ta part is equa	l to the value a	fter which the	node has
to be inserted.				
$X 1 \rightarrow 3 \rightarrow 1$	4	7	8	9 X
START	PTR	10 10 30% dt		100 - 200 - 30
Delete the node succeeding PTR.				
$X 1 \rightarrow 3 \rightarrow$	4	7	8	9 X
START	PTR	the barrier have been		16 26 <u>1</u>
$X 1 \rightarrow 3 \rightarrow 1$	4	8	9 X	
START				

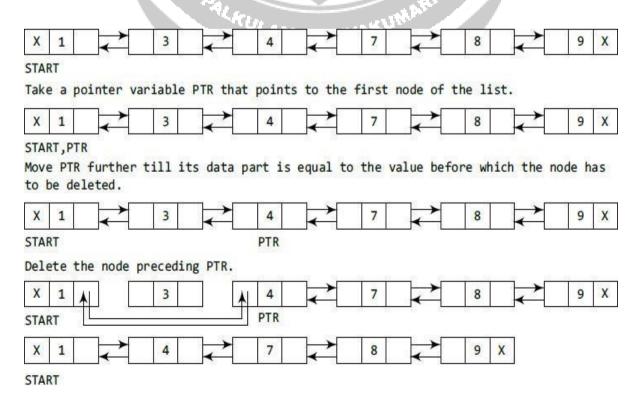
Algorithm to delete a node after a given node

```
Step 1: IF START = NULL
Write UNDERFLOW
Go to Step 9
[END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4: SET PTR = PTR -> NEXT
[END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step 7: SET TEMP -> NEXT = TEMP -> NEXT
Step 8: FREE TEMP
Step 9: EXIT
```

In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the doubly linked list. The while loop traverses through the linked list to reach the given node. Once we reach the node containing VAL, the node succeeding it can be easily accessed by using the address stored in its NEXT field. The NEXT field of the given node is set to contain the contents in the NEXT field of the succeeding node. Finally, the memory of the node succeeding the given node is freed and returned to the free pool.

Case 4: The node before a given node is deleted.

Suppose we want to delete the node preceding the node with value 4



Algorithm to delete a node before a given node

```
Step 1: IF START = NULL
Write UNDERFLOW
Go to Step 9
[END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR->DATA != NUM
Step 4: SET PTR = PTR->NEXT
[END OF LOOP]
Step 5: SET TEMP = PTR->PREV
Step 6: SET TEMP = PTR->PREV
Step 7: SET PTR->PREV = TEMP->PREV
Step 8: FREE TEMP
Step 9: EXIT
```

- In Step 2, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list.
- The while loop traverses through the linked list to reach the desired node.
- Once we reach the node containing VAL, the PREV field of PTR is set to contain the address of the node preceding the node which comes before PTR.

ULAM, KANYA

OBSERVE OPTIMIZE OUTSPREP

• The memory of the node preceding PTR is freed and returned to the free pool.