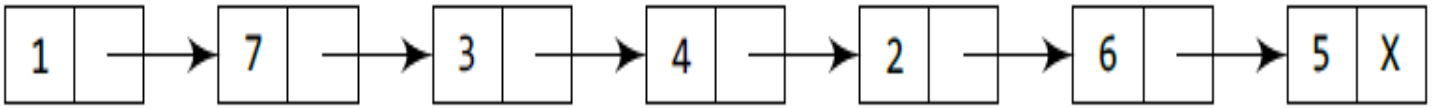
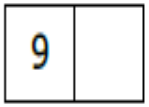
UNIT II

Insert node at beginning

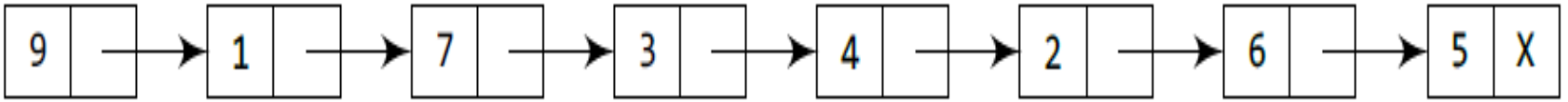


START

Allocate memory for the new node and initialize its DATA part to 9.

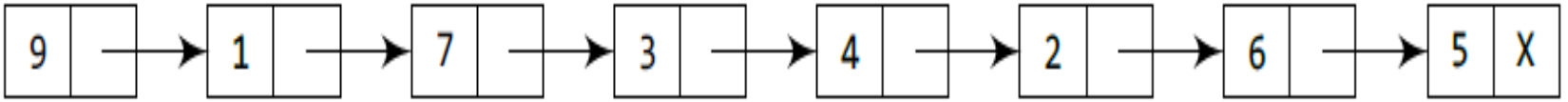


Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START.



START

Now make START to point to the first node of the list.

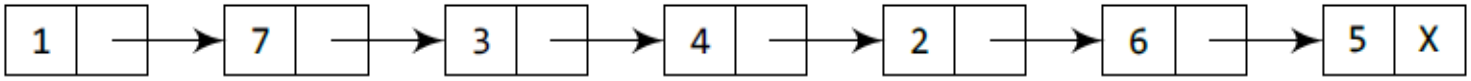


START

Alg to Insert node at beginning

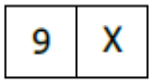
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

Insert node at the end

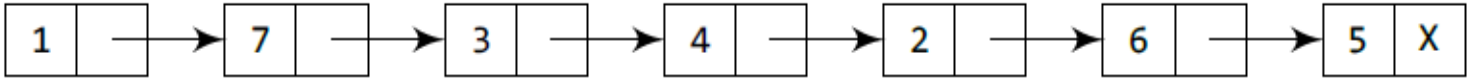


START

Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.

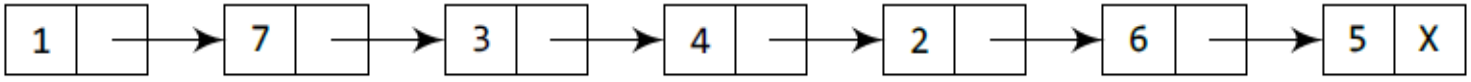


Take a pointer variable PTR which points to START.



START, PTR

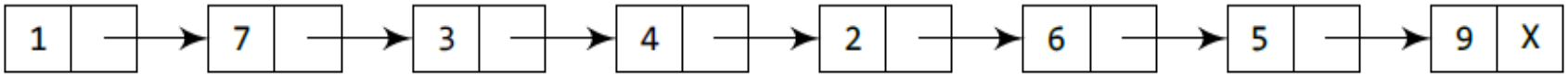
Move PTR so that it points to the last node of the list.



START

PTR

Add the new node after the node pointed by PTR. This is done by storing the address of the new node in the NEXT part of PTR.



START

PTR

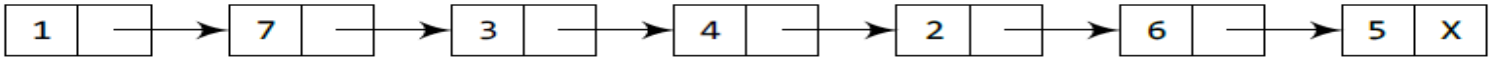
Alg to Insert node at the end

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [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
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

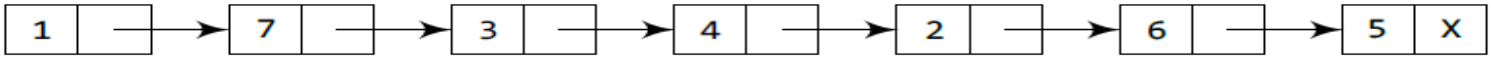
Insert node after the given node

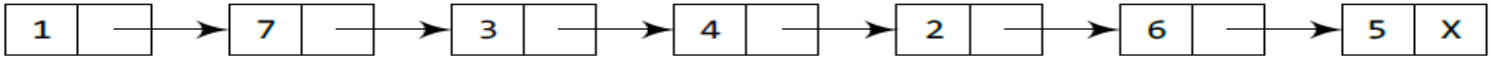
START
Allocate memory for the new node and initialize its DATA part to 9.

Take two pointer variables PTR and PREPTR and initialize them with START so that START, PTR, and PREPTR point to the first node of the list.

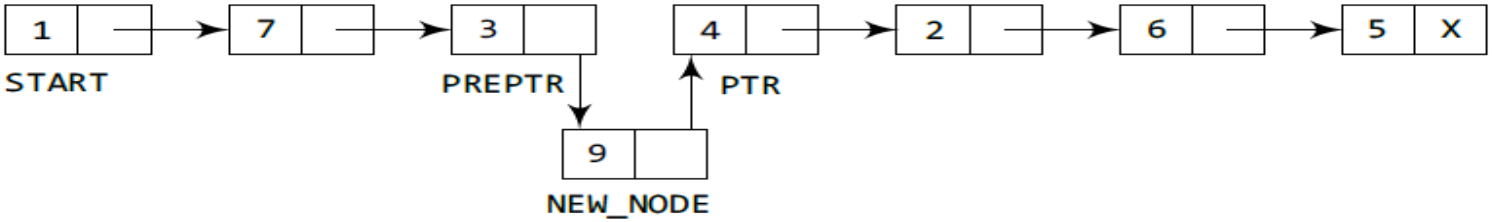


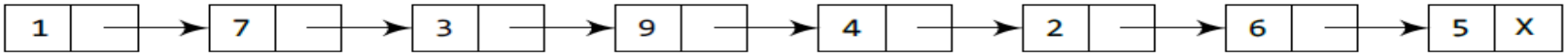
Move PTR and PREPTR until the DATA part of PREPTR = value of the node after which insertion has to be done. PREPTR will always point to the node just before PTR.





Add the new node in between the nodes pointed by PREPTR and PTR.

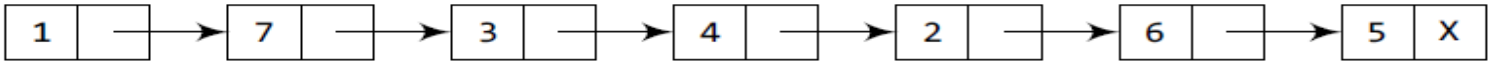




Alg to Insert node after the given node

```
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
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
         != NUM
Step 8:      SET PREPTR = PTR
Step 9:      SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

Insert node before the given node



START

Allocate memory for the new node and initialize its DATA part to 9.

Initialize PREPTR and PTR to the START node.

START

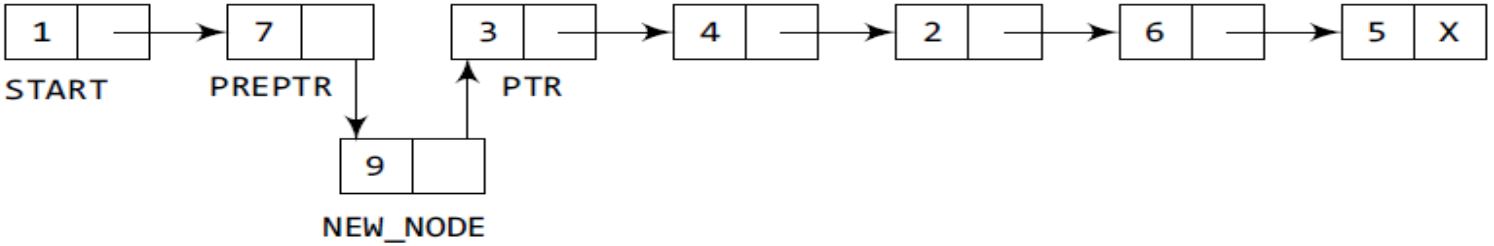
PTR

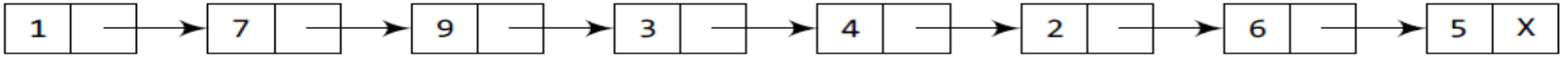
PREPTR

Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.

START PREPTR PTR

Insert the new node in between the nodes pointed by PREPTR and PTR.





START

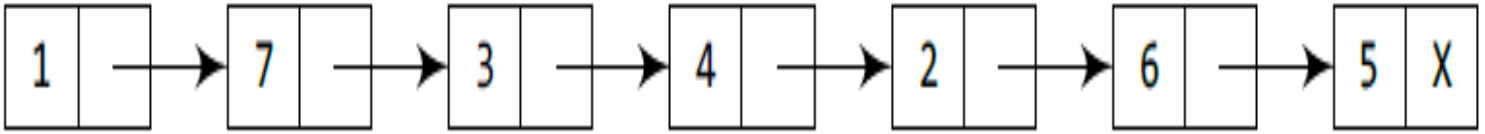
Alg to Insert node before the given node

```
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
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
         [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

Delete node in the SLL

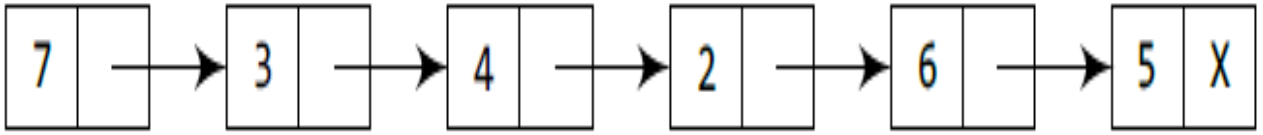
- Delete the first node
- Delete the last node
- the node after the given node is deleted

Delete the first node



START

Make START to point to the next node in sequence.

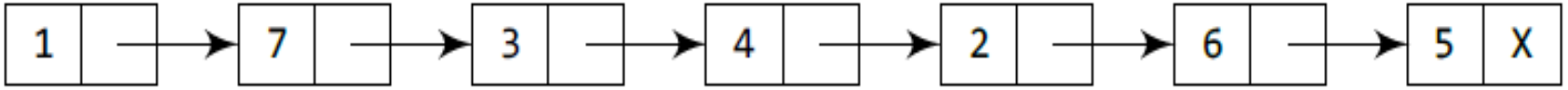


START

Alg to Delete the first node

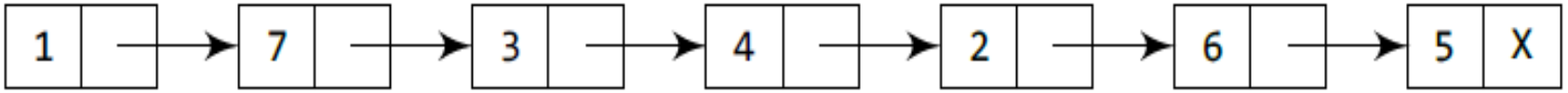
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

Delete the last node



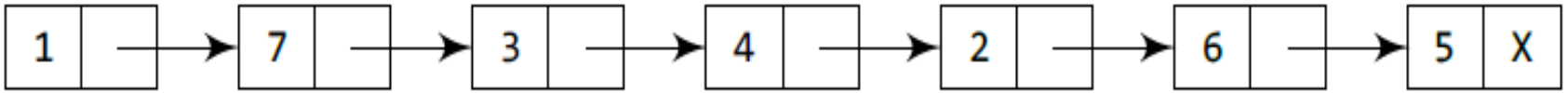
START

Take pointer variables PTR and PREPTR which initially point to START.



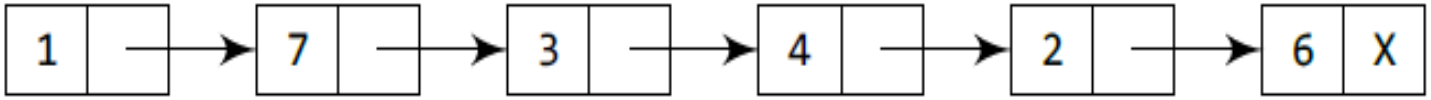
START
PREPTR
PTR

Move PTR and PREPTR such that NEXT part of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR.



START PREPTR PTR

Set the NEXT part of PREPTR node to NULL.

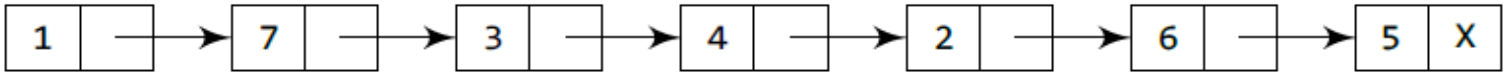


START

Alg to Delete the last node

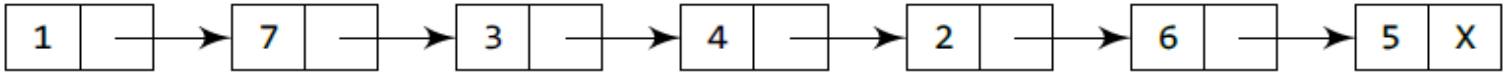
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step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

node after the given node is deleted



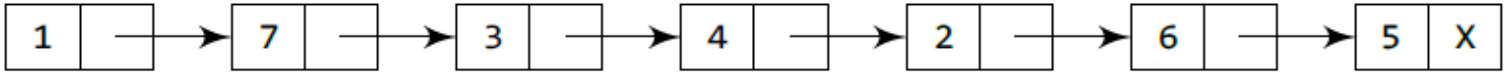
START

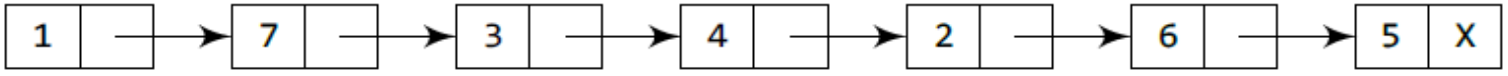
Take pointer variables PTR and PREPTR which initially point to START.

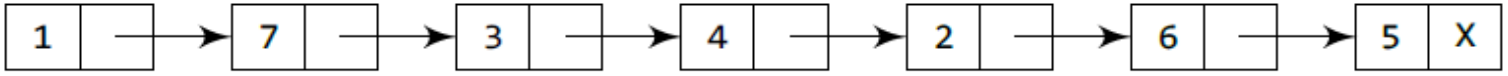


START
PREPTR
PTR

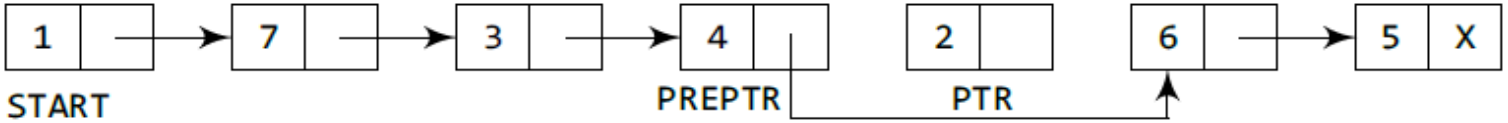
Move PREPTR and PTR such that PREPTR points to the node containing VAL and PTR points to the succeeding node.







Set the NEXT part of PREPTR to the NEXT part of PTR.



Alg- the node after the given node is deleted

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

CLL

Case 1: The new node is inserted at the beginning of the circular linked list.

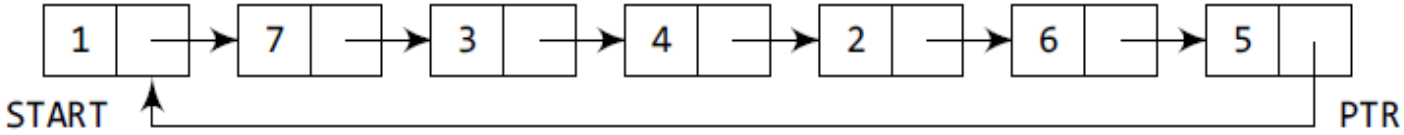
Case 2: The new node is inserted at the end of the circular linked list.

Insert node at the beginning (CLL)

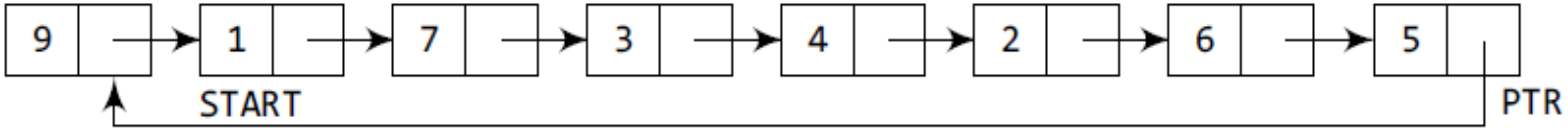
Allocate memory for the new node and initialize its DATA part to 9.

Take a pointer variable PTR that points to the START node of the list.

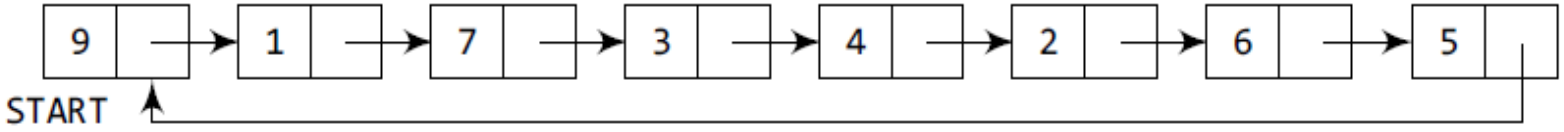
Move PTR so that it now points to the last node of the list.



Add the new node in between PTR and START.



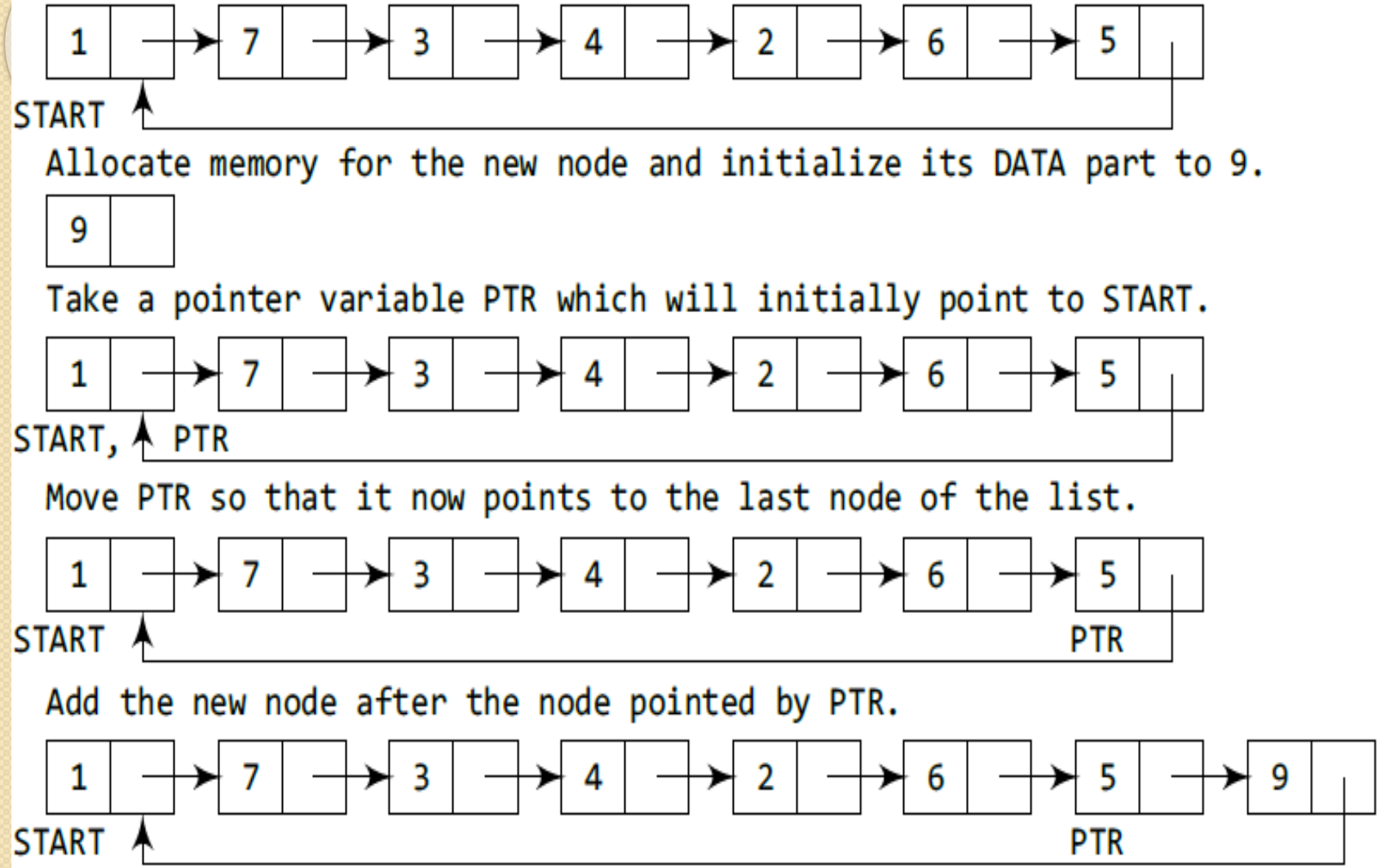
Make START point to the new node.



Alg to Insert node at the beginning

```
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 PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:      PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

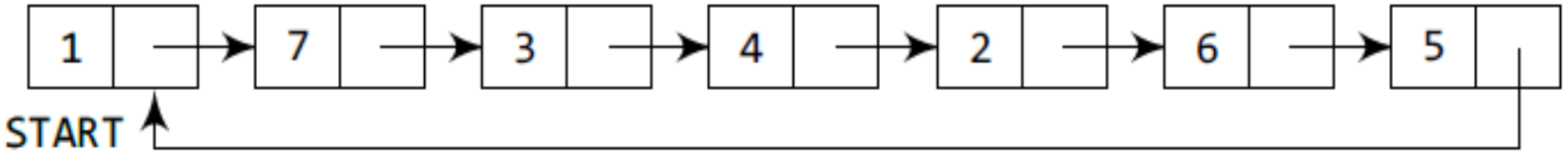
Insert node at the end (CLL)



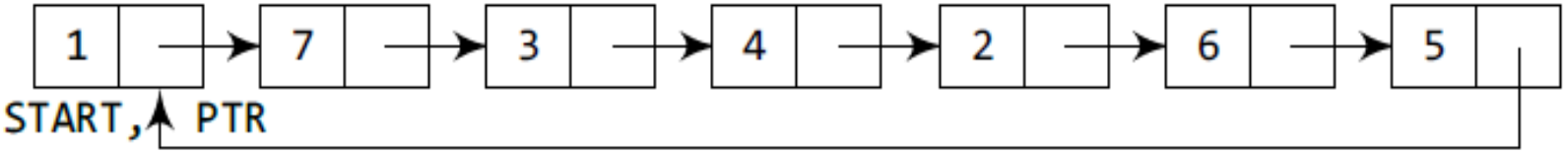
Alg to Insert node at the end

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [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 = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:      SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

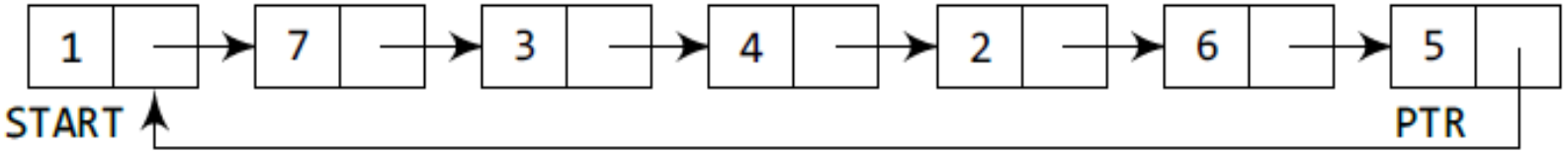
Delete the first node (CLL)



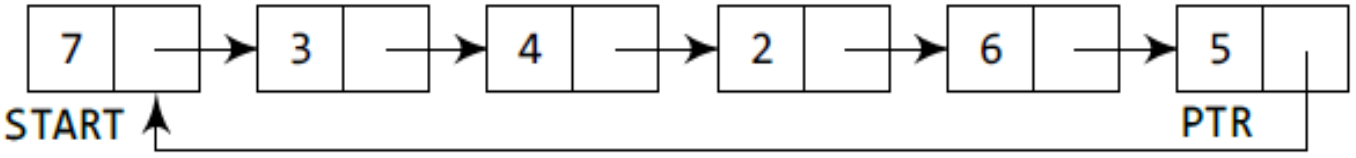
Take a variable PTR and make it point to the START node of the list.



Move PTR further so that it now points to the last node of the list.



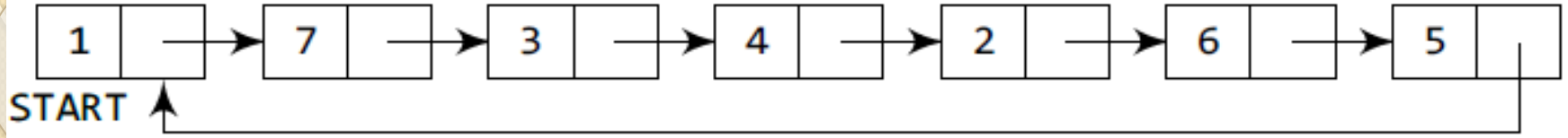
The NEXT part of PTR is made to point to the second node of the list and the memory of the first node is freed. The second node becomes the first node of the list.



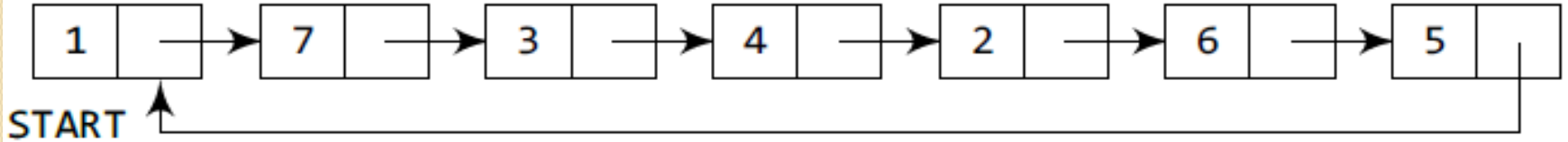
Alg to Delete the first node

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

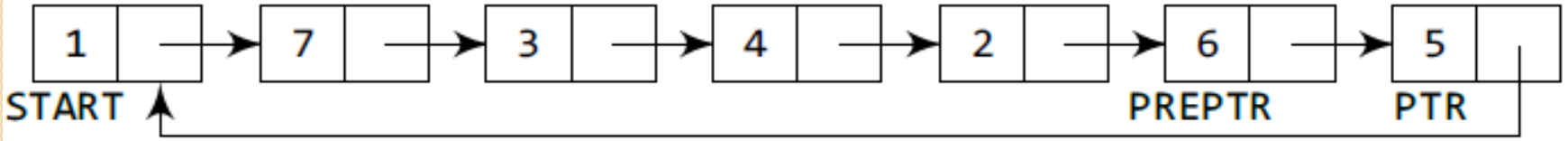
Delete the last node (CLL)



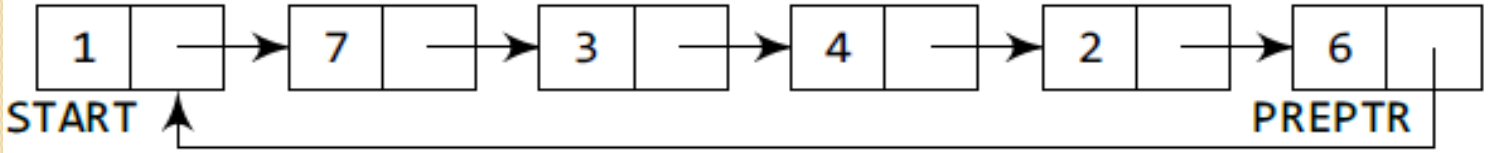
Take two pointers PREPTR and PTR which will initially point to START



Move PTR so that it points to the last node of the list. PREPTR will always point to the node preceding PTR.



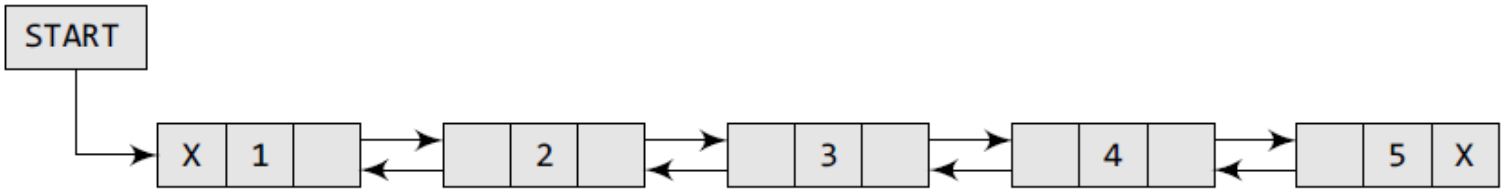
Make the PREPTR's next part store START node's address and free the space allocated for PTR. Now PREPTR is the last node of the list.



Alg to Delete the last node

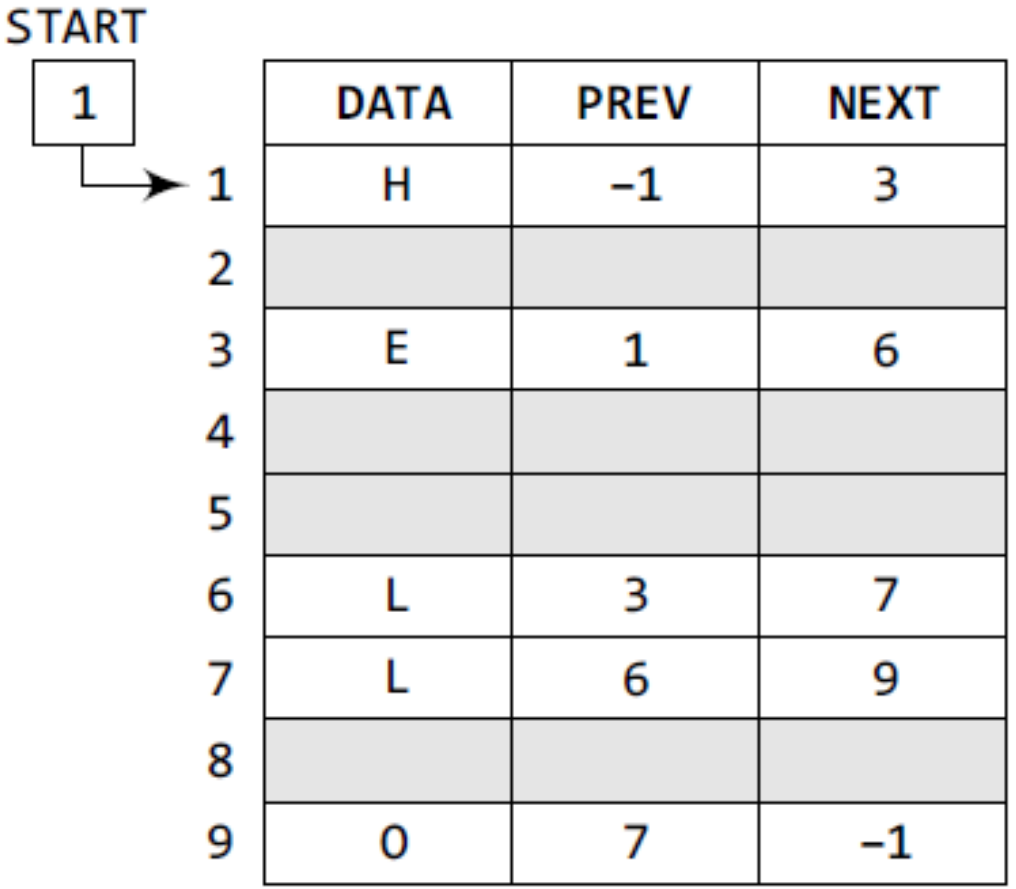
```
Step 1: IF START = NULL
                Write UNDERFLOW
                Go to Step 8
          [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START
Step 4:         SET PREPTR = PTR
Step 5:         SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET PREPTR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

DLL




```
struct node
{
        struct node *prev;
        int data;
        struct node *next;
};
```

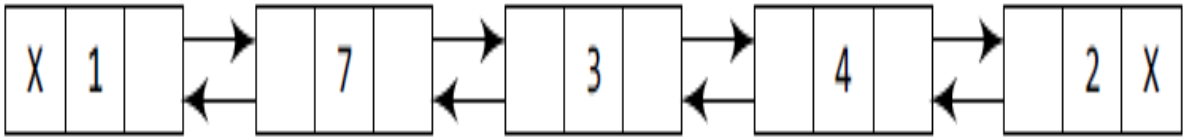
Memory representation of a doubly linked list



	DATA	PREV	NEXT
1	H	-1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	-1

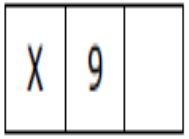
- Case 1: The new node is inserted at the beginning
- Case 2: The new node is inserted at the end.
- Case 3: The new node is inserted after a given node.
- Case 4: The new node is inserted before a given node.

Inserting a Node at the Beginning of a Doubly Linked List

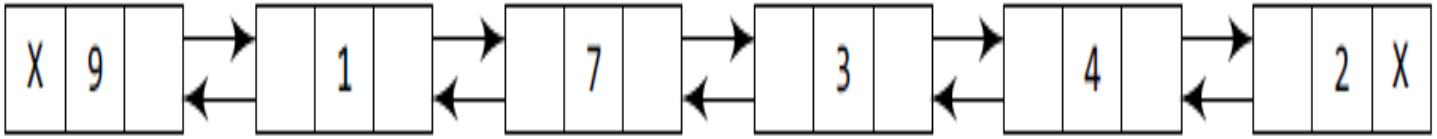


START

Allocate memory for the new node and initialize its DATA part to 9 and PREV field to NULL.



Add the new node before the START node. Now the new node becomes the first node of the list.

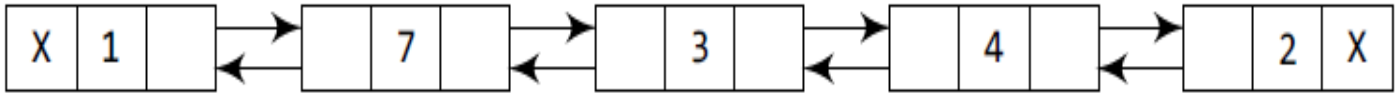


START

Algorithm to insert a new node at the beginning (DLL)

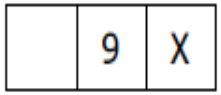
```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

Inserting a Node at the End end of a Doubly Linked List

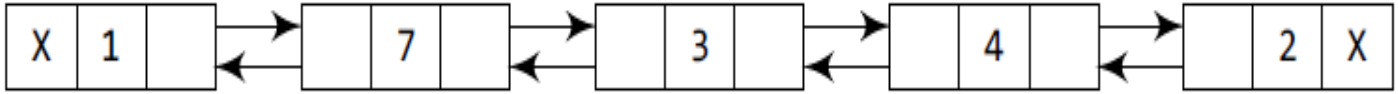


START

Allocate memory for the new node and initialize its DATA part to 9 and its NEXT field to NULL.

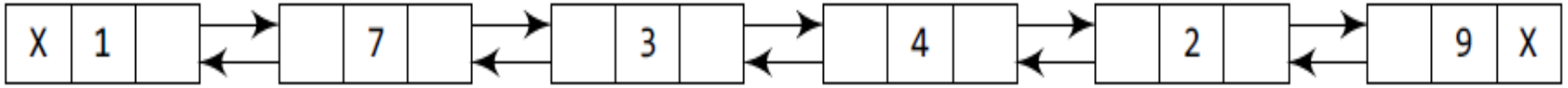


Take a pointer variable PTR and make it point to the first node of the list.



START,PTR

Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.

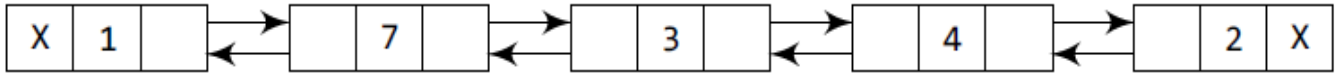


START PTR

Alg to Insert node at the end (DLL)

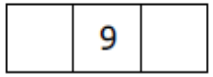
```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != 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
```

Inserting a Node After a Given Node in a Doubly Linked List

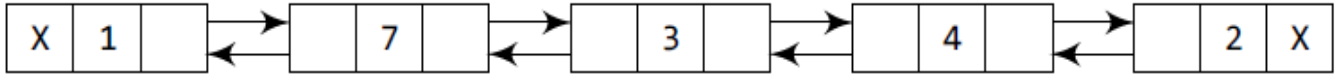


START

Allocate memory for the new node and initialize its DATA part to 9.

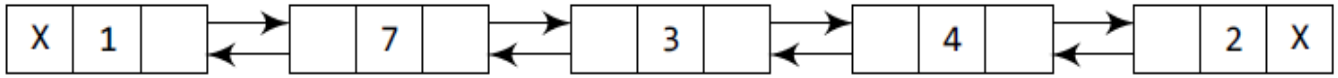


Take a pointer variable PTR and make it point to the first node of the list.



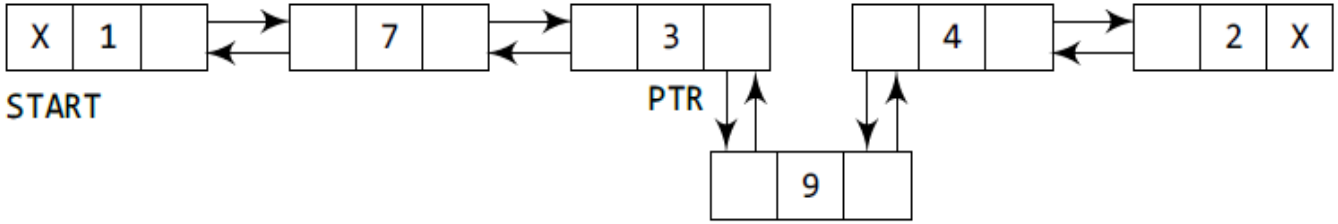
START,PTR

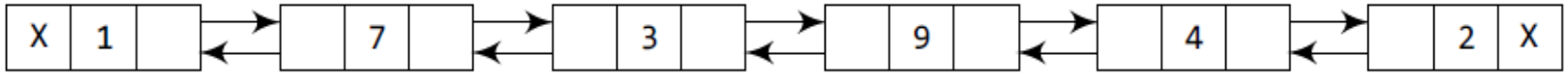
Move PTR further until the data part of PTR = value after which the node has to be inserted.



START PTR

Insert the new node between PTR and the node succeeding it.





START

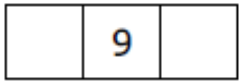
Algorithm to insert a new node after a given node

```
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
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = 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
```

Inserting a new node before a given node in a doubly linked list

START

Allocate memory for the new node and initialize its DATA part to 9.



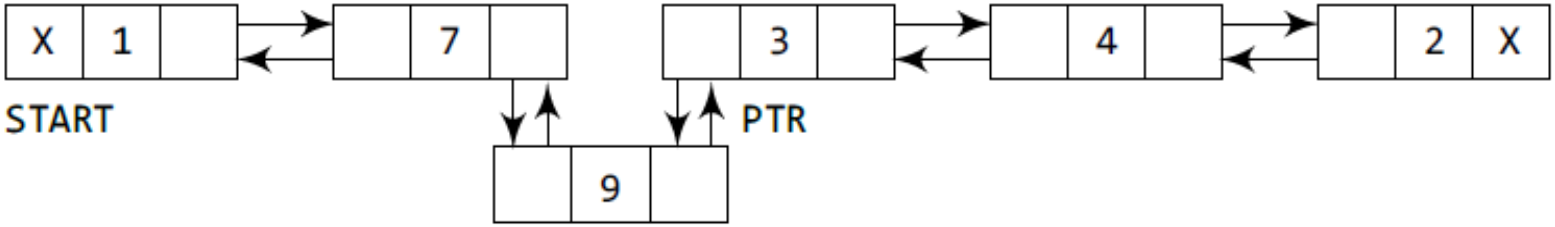
Take a pointer variable PTR and make it point to the first node of the list.

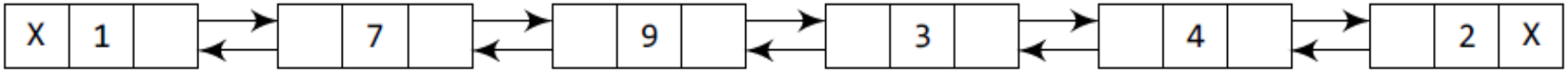
START, PTR

Move PTR further so that it now points to the node whose data is equal to the value before which the node has to be inserted.

START

Add the new node in between the node pointed by PTR and the node preceding it.





START

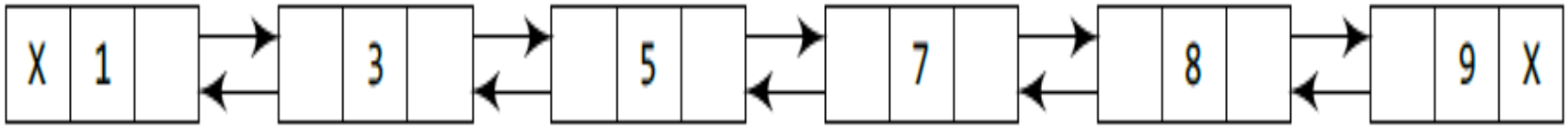
Algorithm to insert a new node before a given node

```
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
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = 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
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10: SET PTR -> PREV = NEW_NODE
Step 11: SET PTR  -> PREV -> NEXT = NEW_NODE
Step 12: EXIT
```

Deleting a Node from a Doubly Linked List

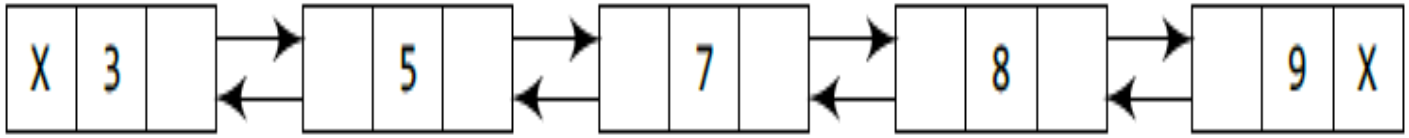
- Case 1: The first node is deleted.
- Case 2: The last node is deleted.
- Case 3: The node after a given node is deleted.
- Case 4: The node before a given node is deleted.

Deleting the First Node from a Doubly Linked List



START

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

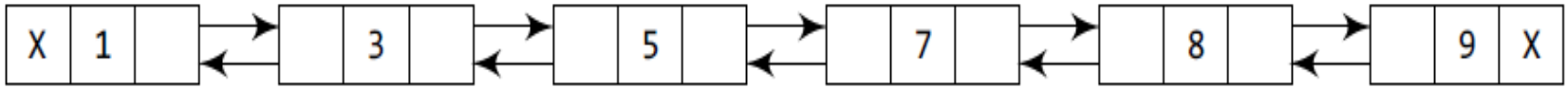


START

Algorithm to delete the first node

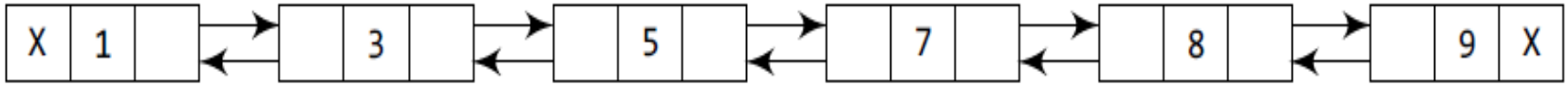
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

Deleting the Last Node from a Doubly Linked List



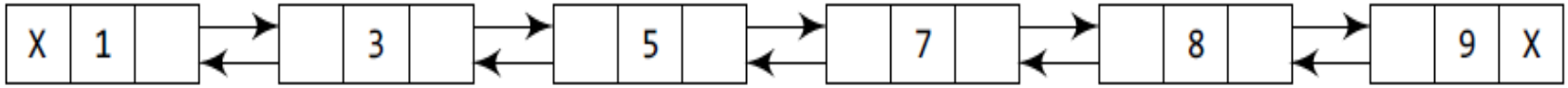
START

Take a pointer variable PTR that points to the first node of the list.



START,PTR

Move PTR so that it now points to the last node of the list.



START PTR

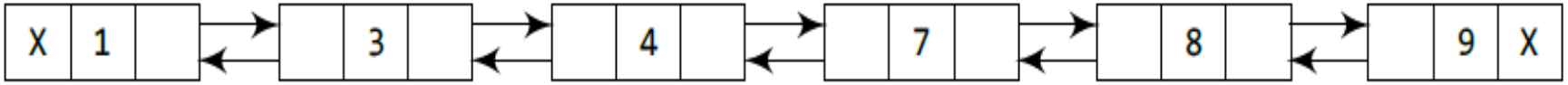
Free the space occupied by the node pointed by PTR and store NULL in NEXT field of its preceding node.

START

Algorithm to delete the last node

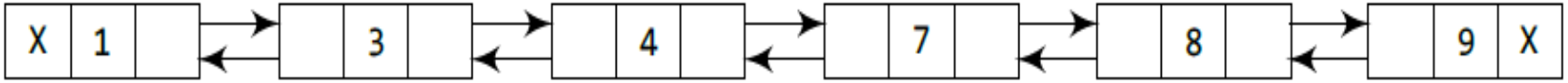
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != NULL
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

Deleting the Node After a Given Node in a Doubly Linked List



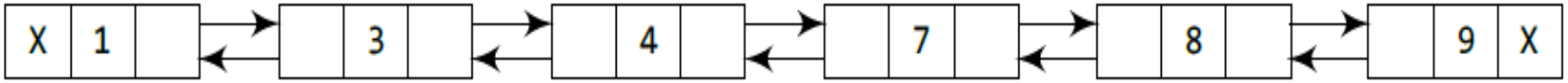
START

Take a pointer variable PTR and make it point to the first node of the list.



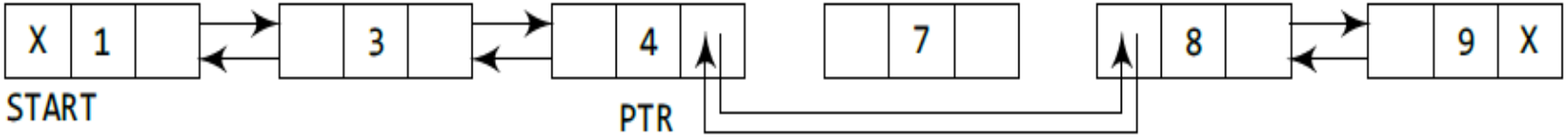
START,PTR

Move PTR further so that its data part is equal to the value after which the node has to be inserted.

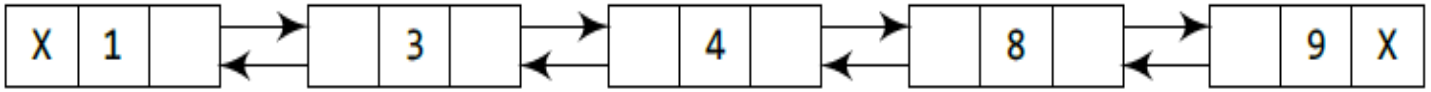


START PTR

Delete the node succeeding PTR.



START PTR

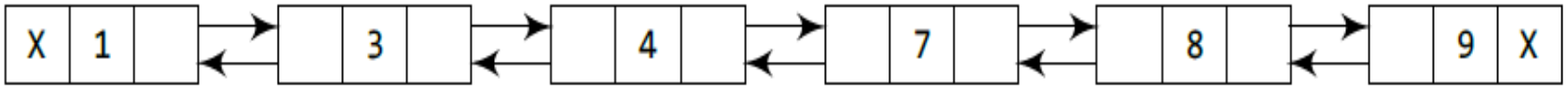


START

Algorithm to delete a node after a given node

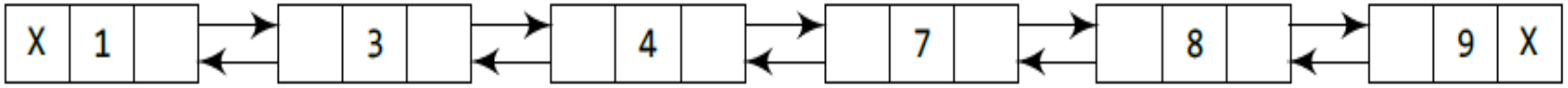
```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step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

Deleting the Node Before a Given Node in a Doubly Linked List



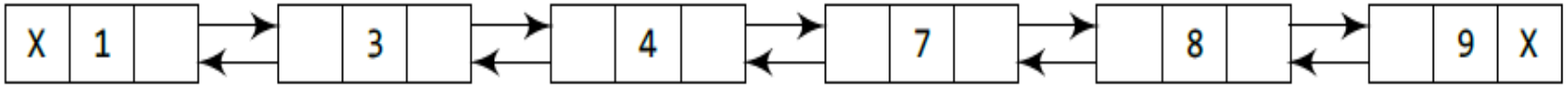
START

Take a pointer variable PTR that points to the first node of the list.



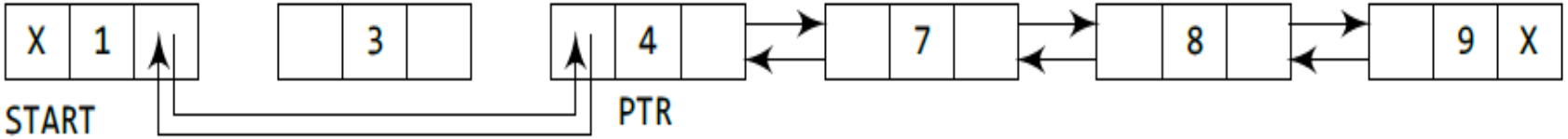
START,PTR

Move PTR further till its data part is equal to the value before which the node has to be deleted.

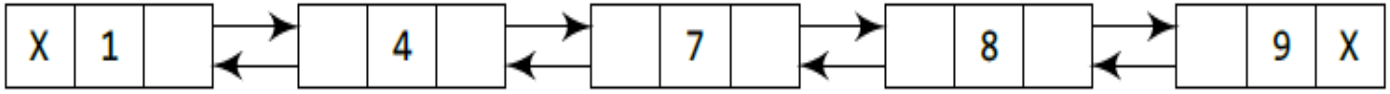


START PTR

Delete the node preceding PTR.



START PTR



START

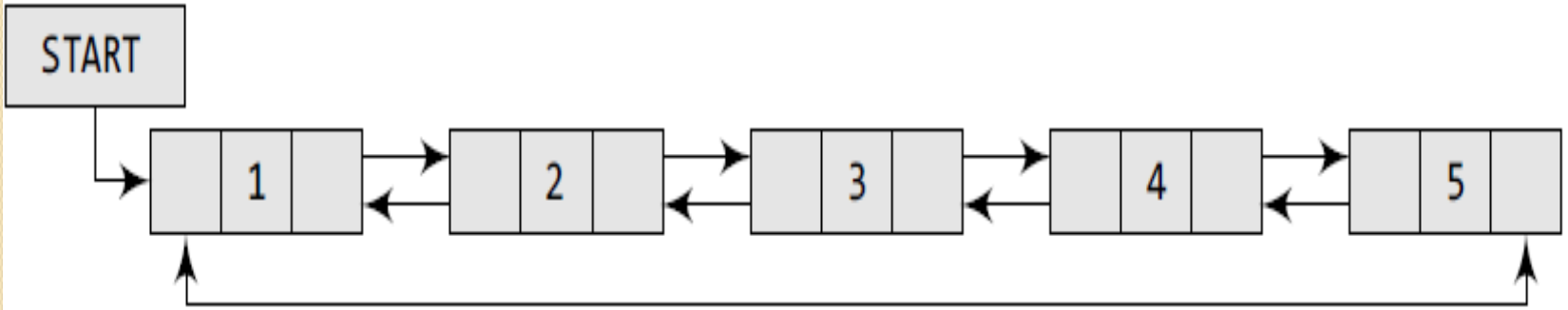
Algorithm to delete a node before a given node

```
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 -> PREV -> NEXT = PTR
Step 7: SET PTR -> PREV = TEMP -> PREV
Step 8: FREE TEMP
Step 9: EXIT
```

CIRCULAR DOUBLY LINKED LISTs

- circular doubly linked list or a circular 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
- The circular doubly linked list does not contain NULL in the previous field of the first node and the next field of the last node.
-

- Rather, the next field of the last node stores the address of the first node of the list, i.e., START. Similarly, the previous field of the first field stores the address of the last node


START
1
2
3
4
5

START

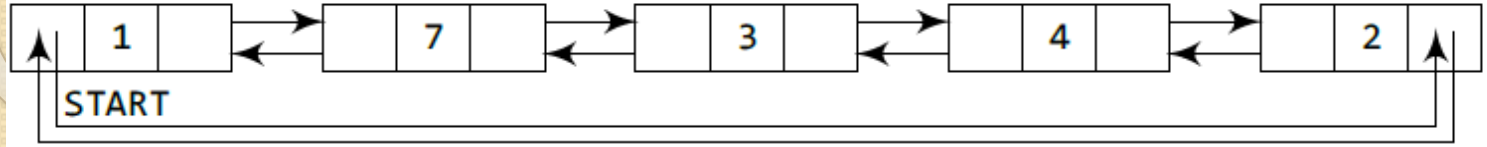
1

	DATA	PREV	Next
1	H	9	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	1

Inserting a New Node in a Circular Doubly Linked List

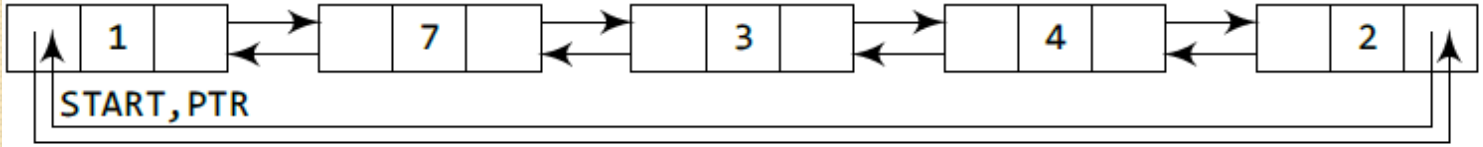
- new node is inserted at the beginning.
- new node is inserted at the end

Inserting a Node at the Beginning of a Circular Doubly Linked List

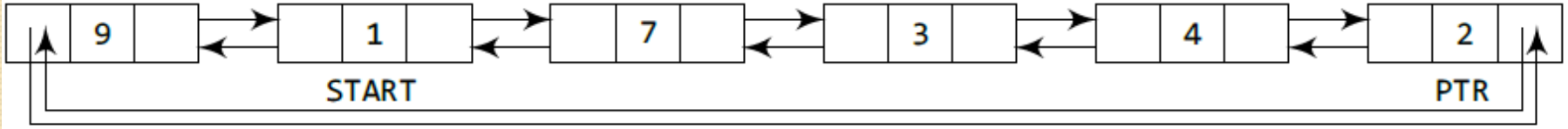


Allocate memory for the new node and initialize its DATA part to 9.

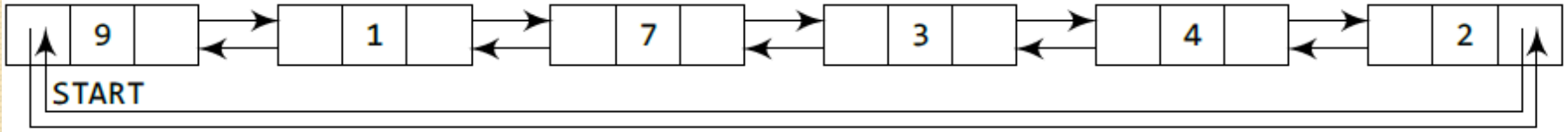
Take a pointer variable PTR that points to the first node of the list.



Move PTR so that it now points to the last node of the list. Insert the new node in between PTR and the START node.



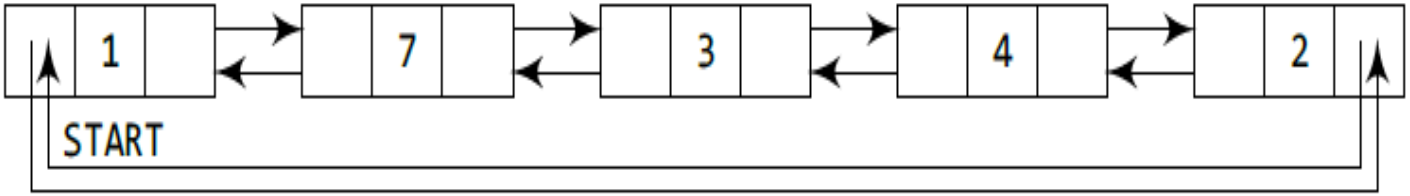
START will now point to the new node.



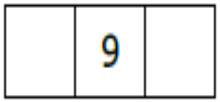
Algorithm to insert a new node at the beginning

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 13
        [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 PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

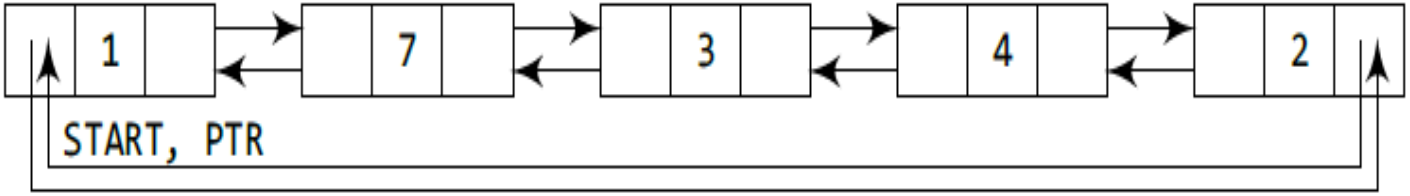
Inserting a Node at the End of a Circular Doubly Linked List



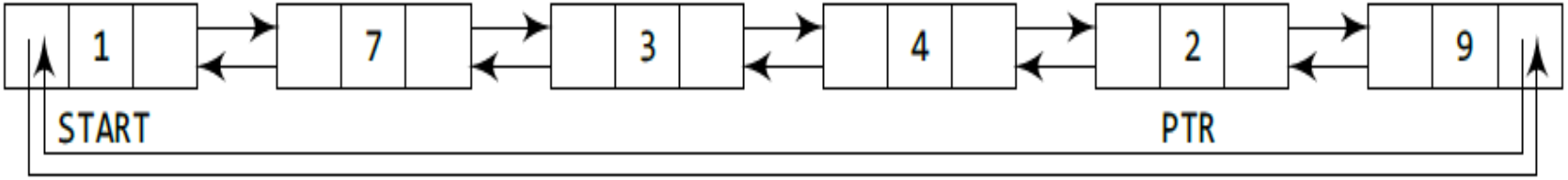
Allocate memory for the new node and initialize its DATA part to 9.



Take a pointer variable PTR that points to the first node of the list.



Move PTR to point to the last node of the list so that the new node can be inserted after it.



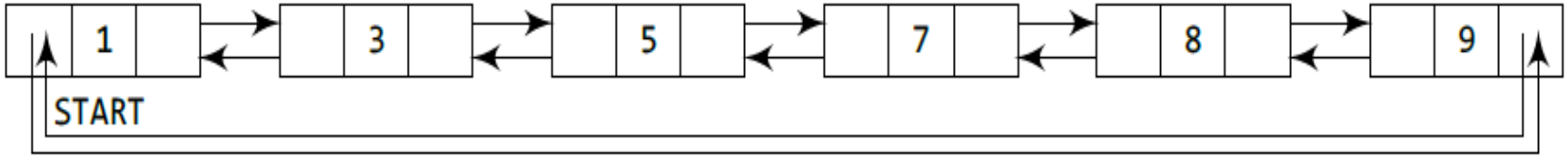
Algorithm to insert a new node at the end

```
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
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
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: SET START -> PREV = NEW_NODE
Step 12: EXIT
```

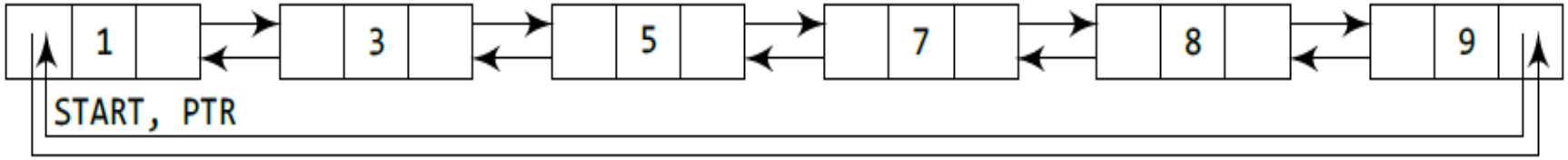
Deleting a Node from a Circular Doubly Linked List

- The first node is deleted.
- The last node is deleted.

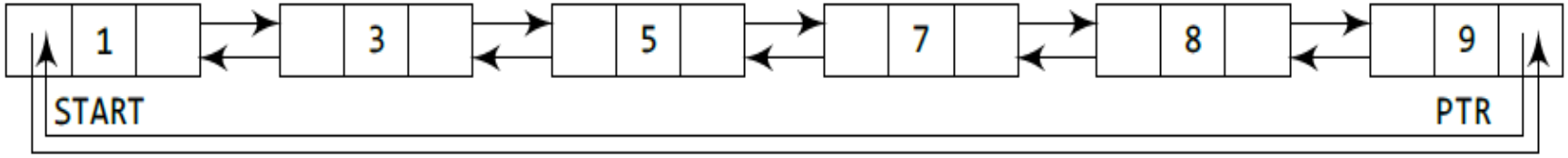
Deleting the First Node from a Circular Doubly Linked List



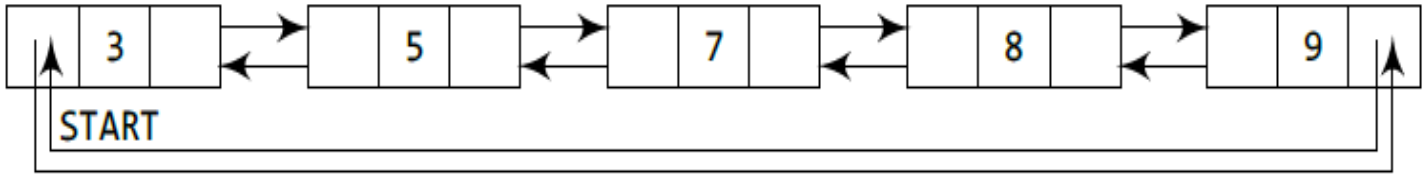
Take a pointer variable PTR that points to the first node of the list.



Move PTR further so that it now points to the last node of the list.



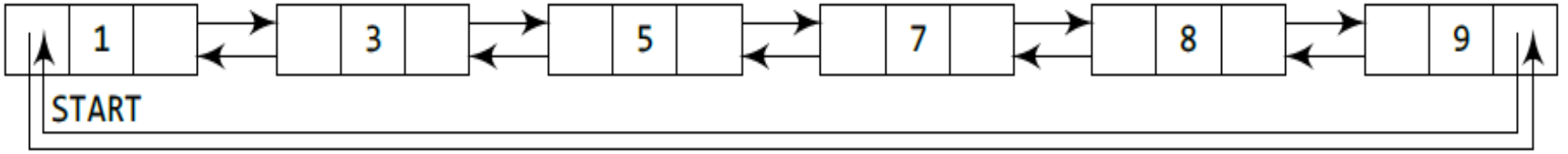
Make START point to the second node of the list. Free the space occupied by the first node.



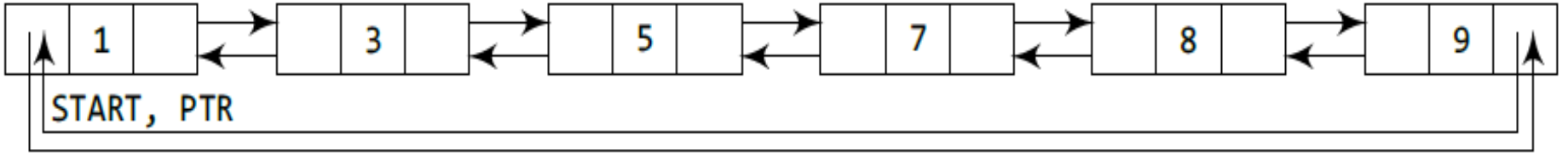
Algorithm to delete the first node

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

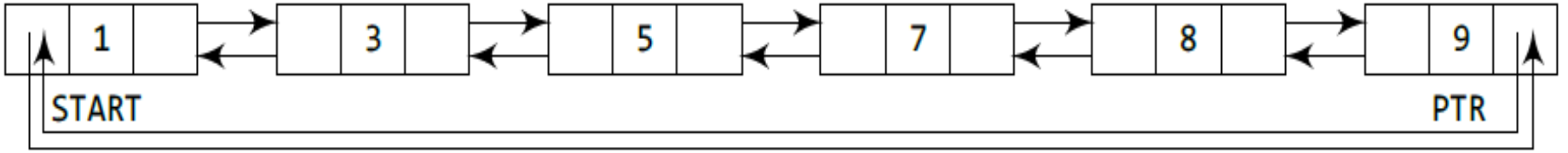
Deleting the Last Node from a Circular Doubly Linked List



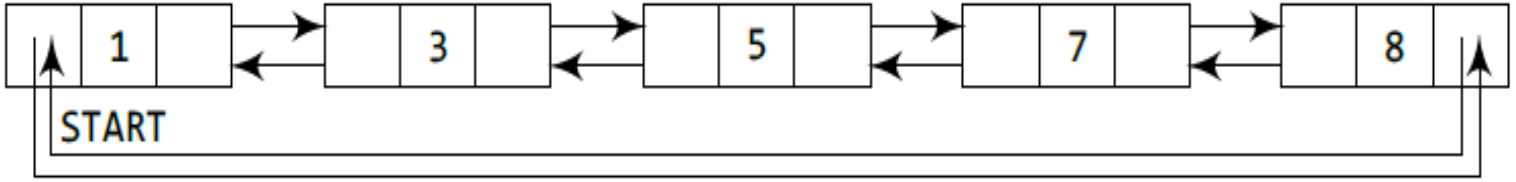
Take a pointer variable PTR that points to the first node of the list.



Move PTR further so that it now points to the last node of the list.



Free the space occupied by PTR.



Algorithm to delete the last node

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