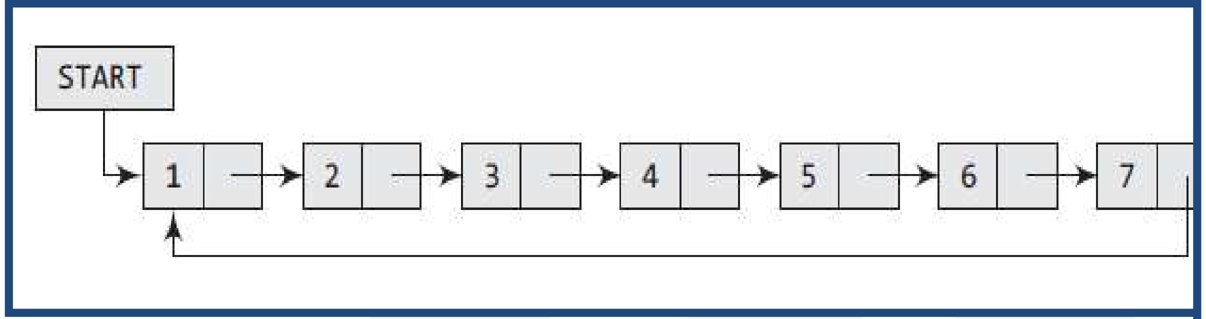
Circular Linked List



START
1

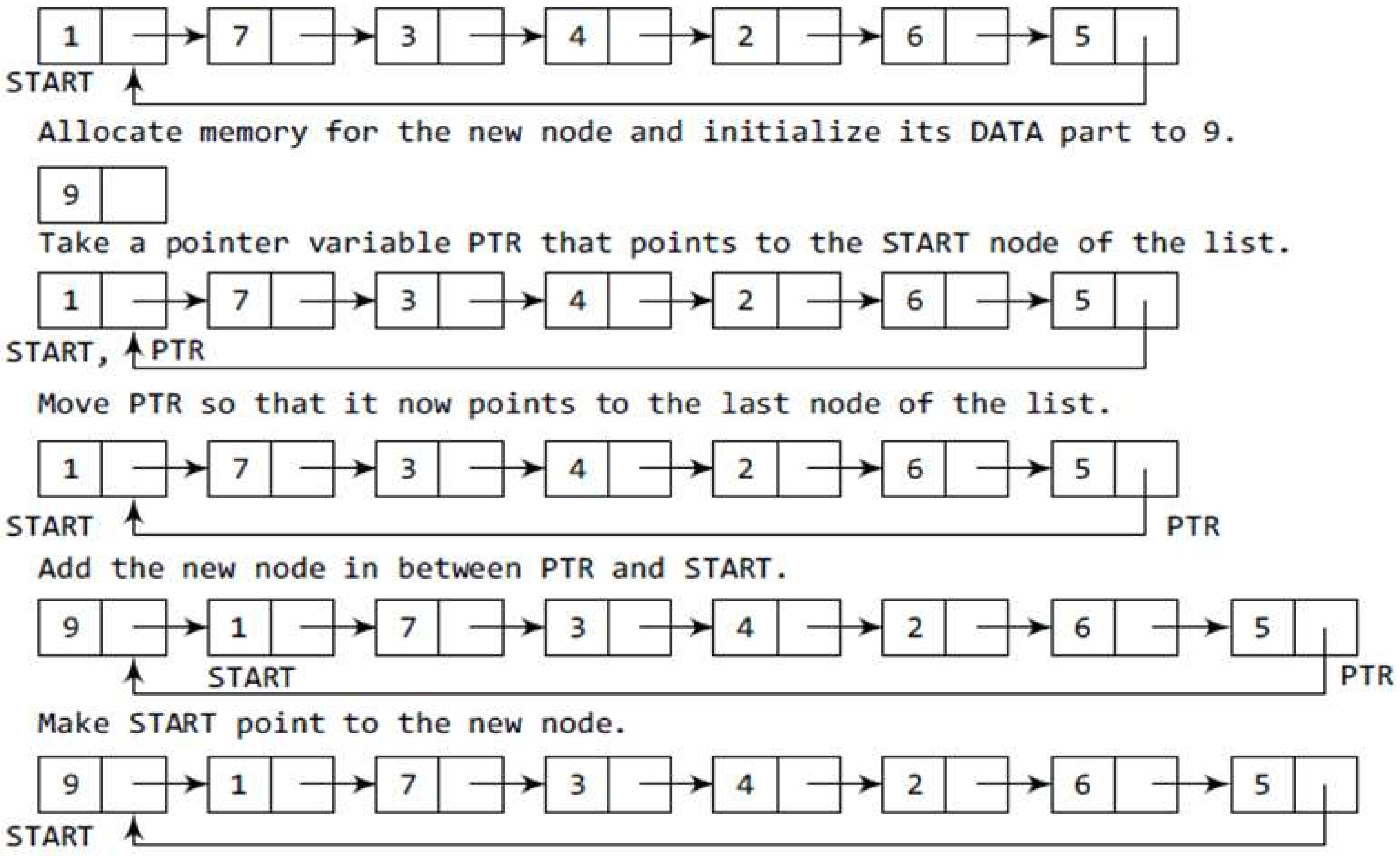
	DATA	NEXT
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	1

In a circular linked list, the last node contains a pointer to the first node of the list.

Circular Link List- Insertion

- 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.

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



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

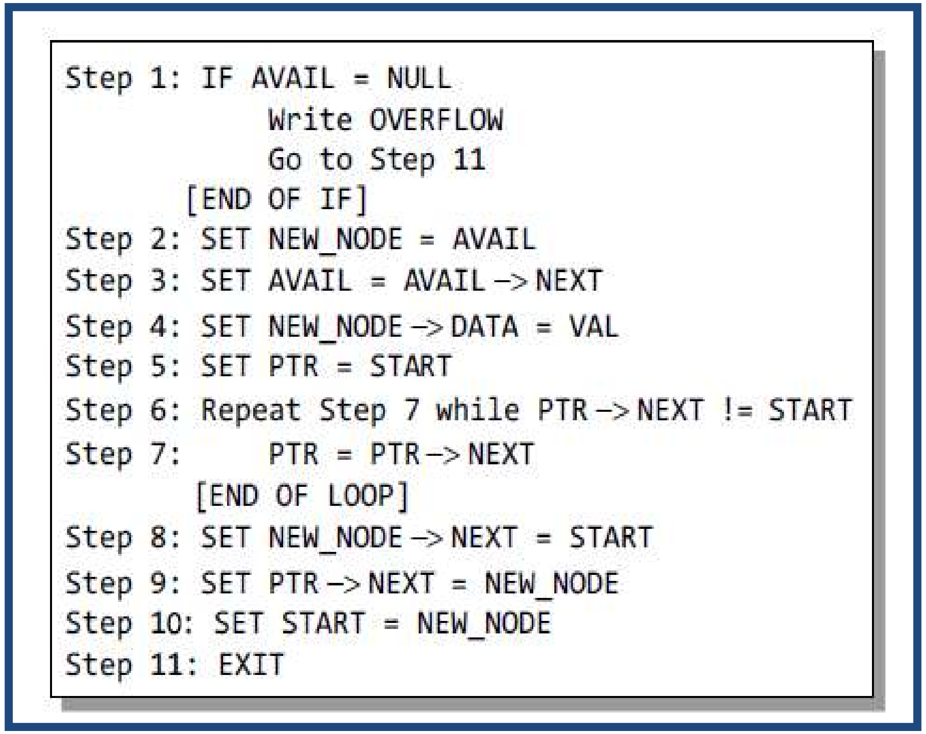
Take a pointer variable PTR that points to the START node of the list.

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.

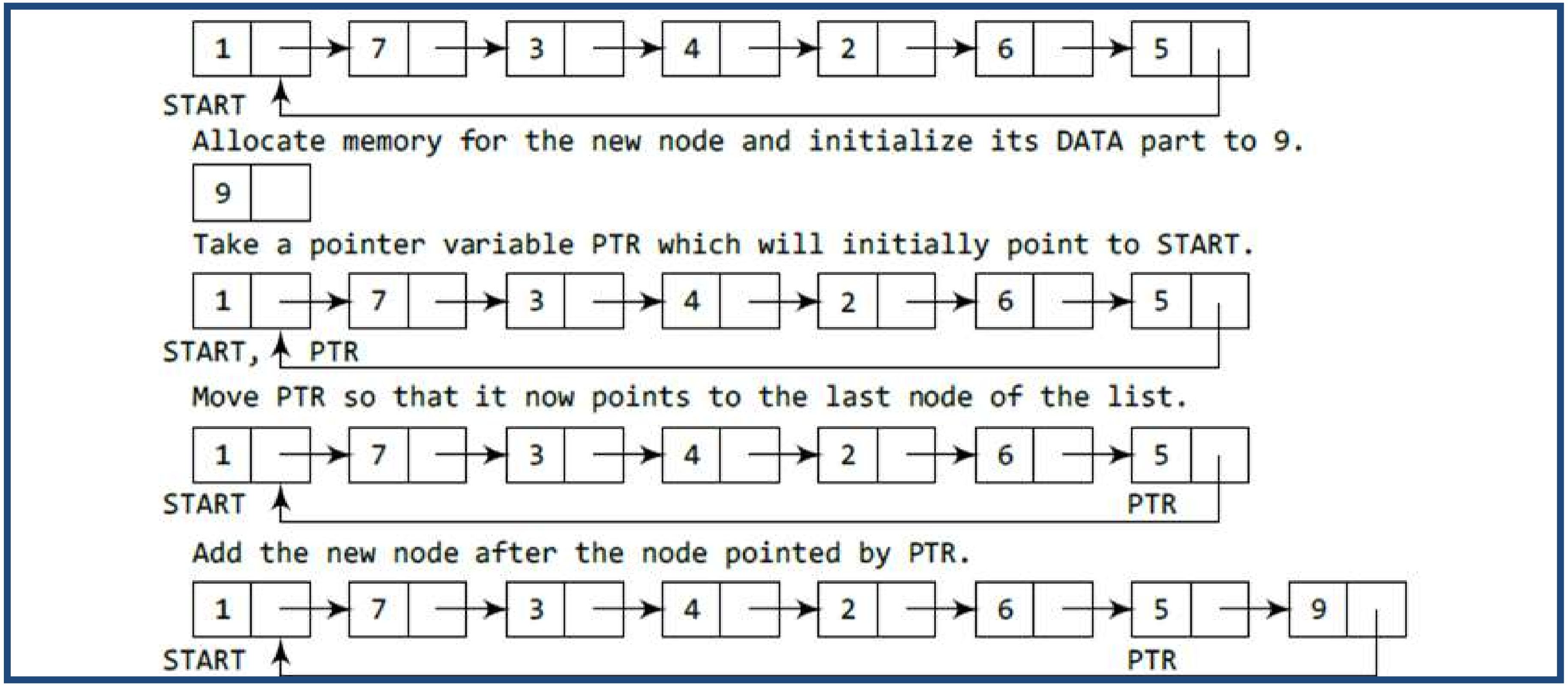
Case 1: The new node is inserted at the beginning of the circular linked list.



```
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
```

- Step 1, we first check whether memory is available for the new node
- 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.
- The START pointer variable will now hold the address of the NEW_NODE.
- While inserting a node in a circular linked list, we have to use a while loop to traverse to the last node of the list. Because the last node contains a pointer to START, its NEXT field is updated so that after insertion it points to the new node which will be now known as START.

Insert node at end of circular link list



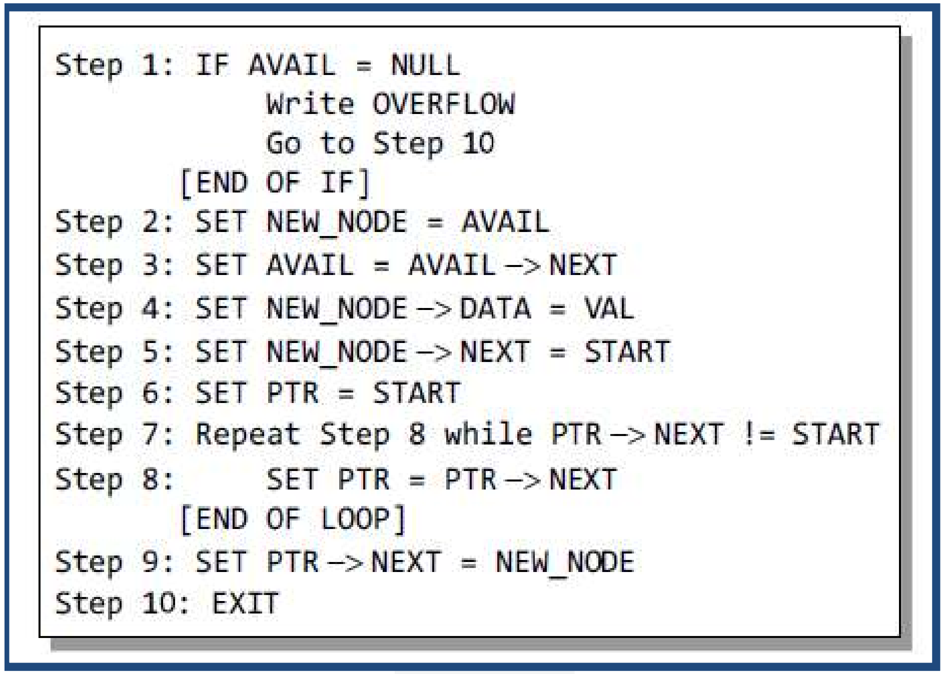
Allocate memory for the new node and initialize its DATA part to 9.

Take a pointer variable PTR which will initially point to START.

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

Add the new node after the node pointed by PTR.

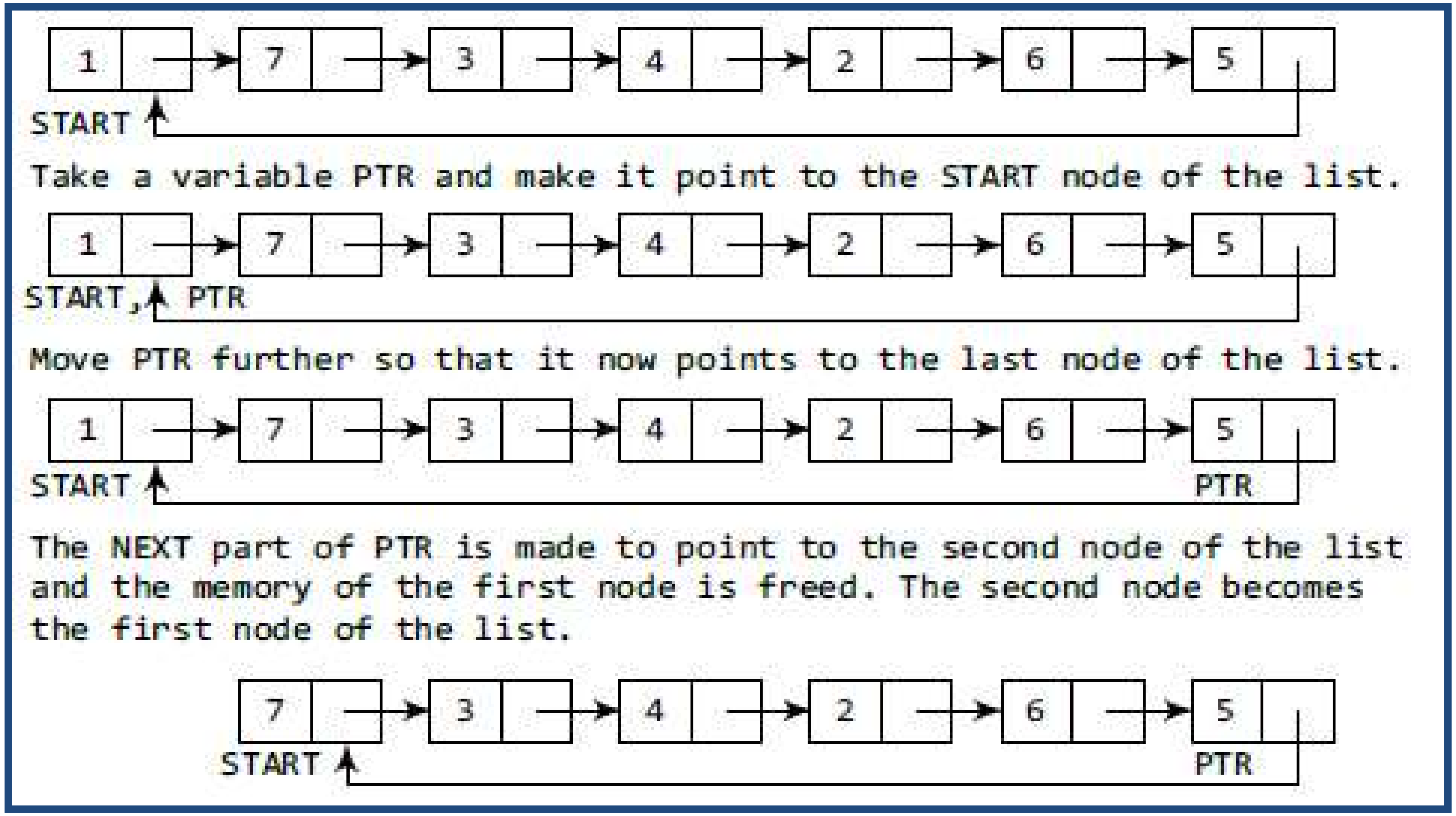
Insert node at end of circular link list



```
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
```

- In Step 6, 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 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 first of the new node contains the address of the fi node which is denoted by START.

Delete first node of circular link list

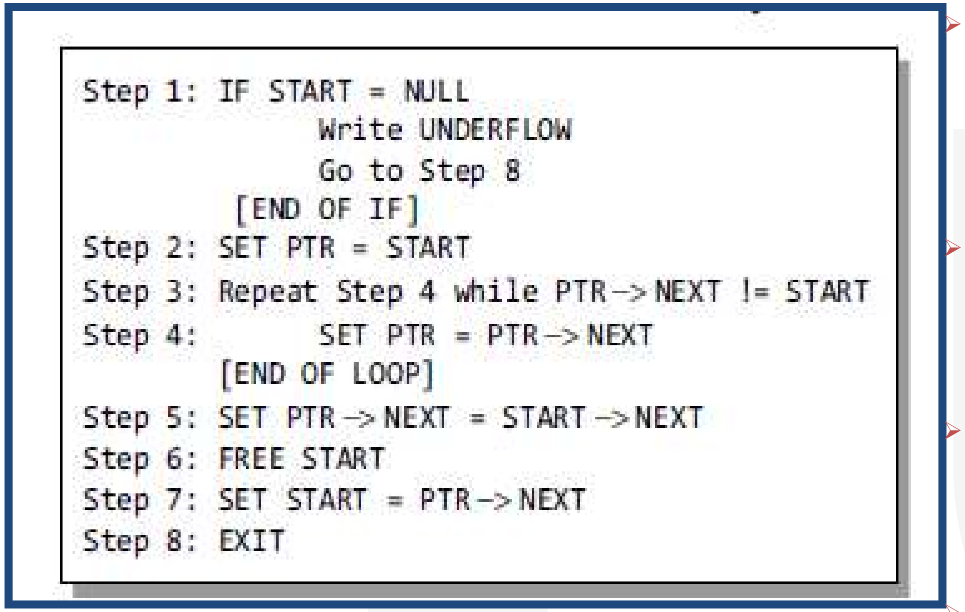


Take a variable PTR and make it point to the START node of the list.

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.

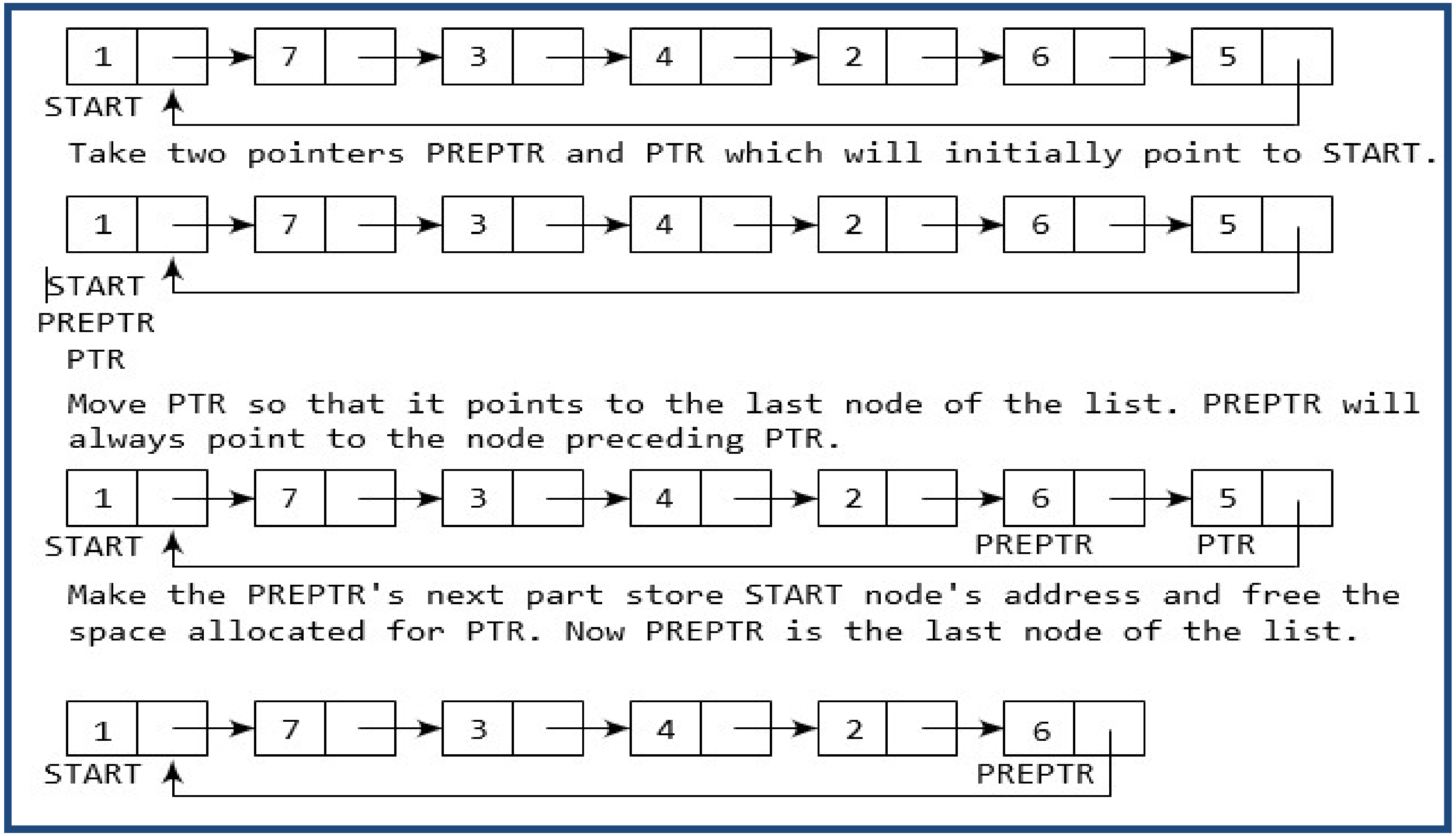
Delete first node of circular link list



```
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
```

- 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 pointer variable PTR which will be used to traverse the list to ultimately reach the last node.
- In Step 5, we change the next pointer of the last node to point to the second node of the circular linked list. In Step 6, the memory occupied by the first node is freed.
- Finally, in Step 7, the second node now becomes the first node of the list and its address is stored in the pointer variable START

Delete last node of circular link list

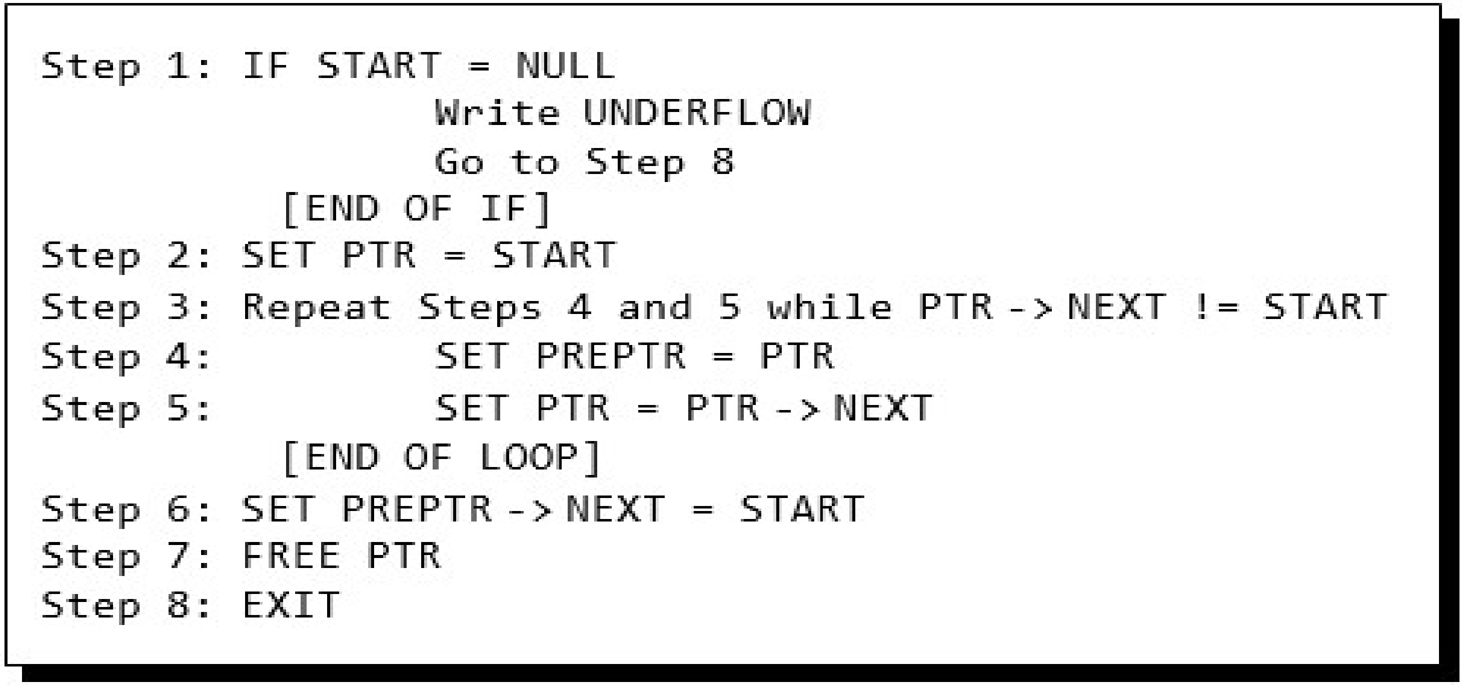


Take two pointers PREPTR and PTR which will initially point to START.

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.

Delete last node of circular link list



```
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
```

Delete last node of circular link list

- In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the fi node of the linked list. In the while loop, we take another pointer variable PREPTR such that PREPTR always points to one node before PTR.
- Once we reach the last node and the second last node, we set the next pointer of the second last node to START, 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.