CIRCULAR LINKED LIST

Circular Linked List

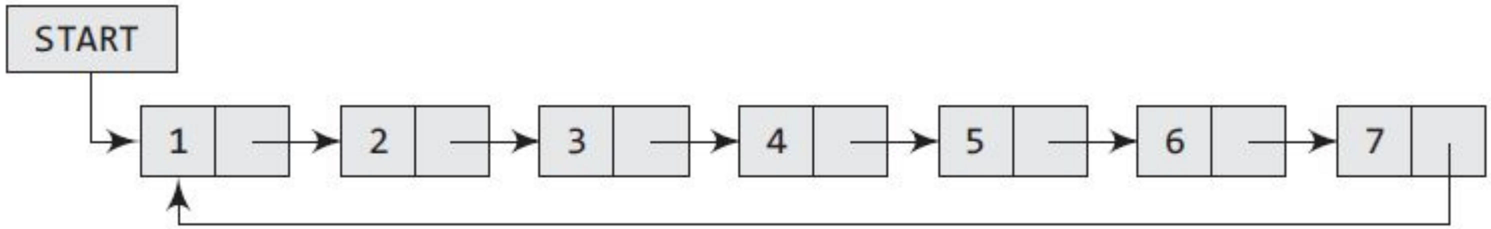


Figure 6.26 Circular linked list

- In the last node of a list, the link field often contains a **null** reference, a special value used to indicate the lack of further nodes.

- A less common convention is to make it point to the first node of the list; in that case the list is said to
be **circular** or **circularly linked**.

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

Figure 6.27 Memory representation of a circular linked list

Circular Linked List

- Advantages

1. In linear linked list it is not possible to go to previous node but within Circular LL possible.
2. It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes. But in double linked list, we will have to go through in between nodes.
3. Browsing of web pages.

Circular Linked List

- Disadvantages

1. If proper care is not taken, then the problem of infinite loop can occur.
2. If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

TRAVERSING
CIRCULAR LINKED LIST

Algorithm

Step 1: [INITIALIZE] SET PTR = START

Step 2: Repeat Steps 3 and 4 while PTR -> NEXT != START

Step 3: Apply Process to PTR -> DATA

Step 4: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 5: EXIT

INSERTIONS IN CIRCULAR LINKED LIST

Insert new node in Circular LL

Case 1: New node is inserted at the beginning

Case 2: New node is inserted at the end

Insertion Case 1:

New node is inserted at the beginning

Example

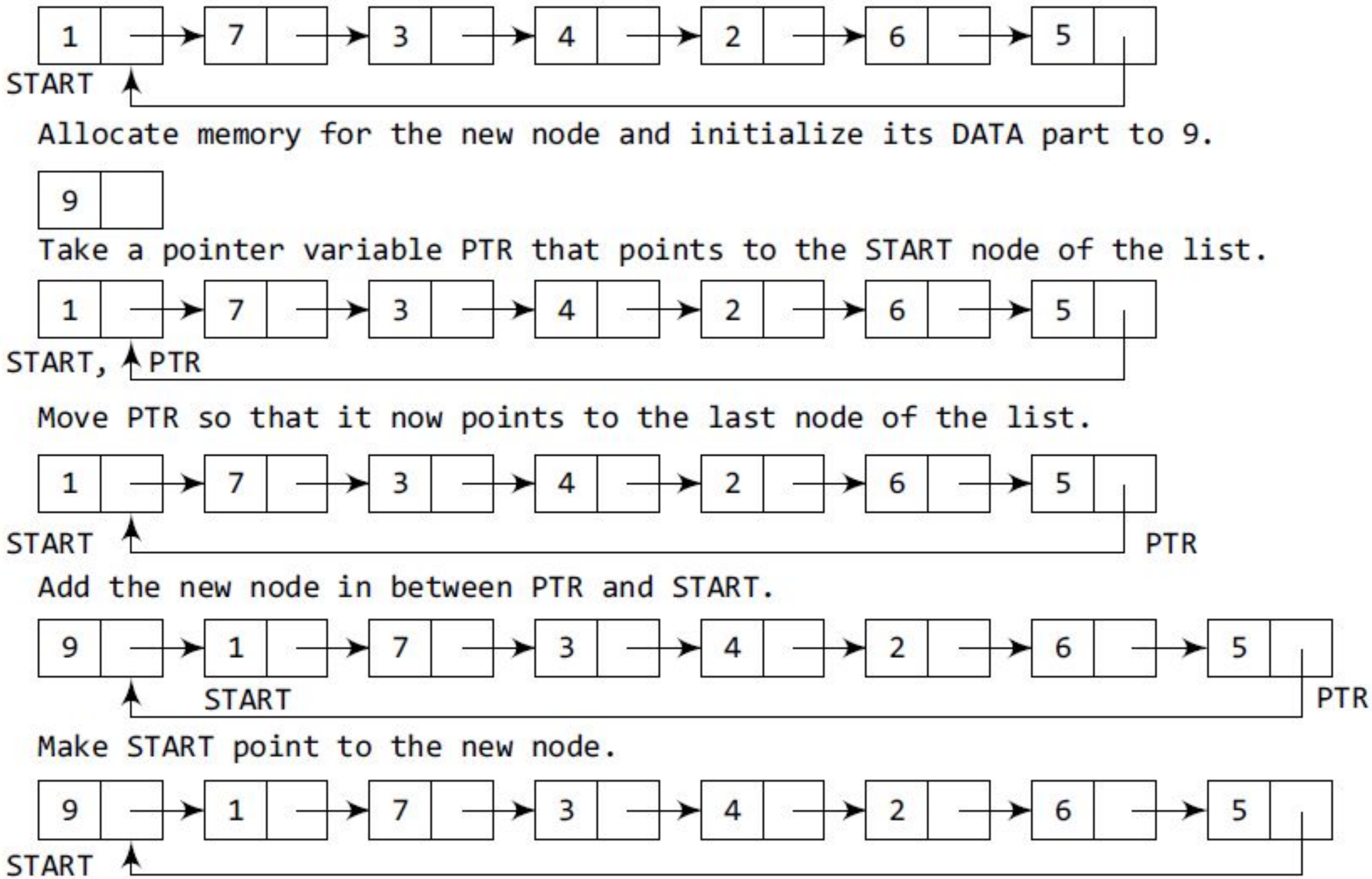


Figure 6.29 Inserting a new node at the beginning of a circular linked list

Algorithm

Step 1: IF AVAIL = NULL then
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 , New_Node -> NEXT = NULL
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

Insertion Case 2:

New node is inserted at the end

Example

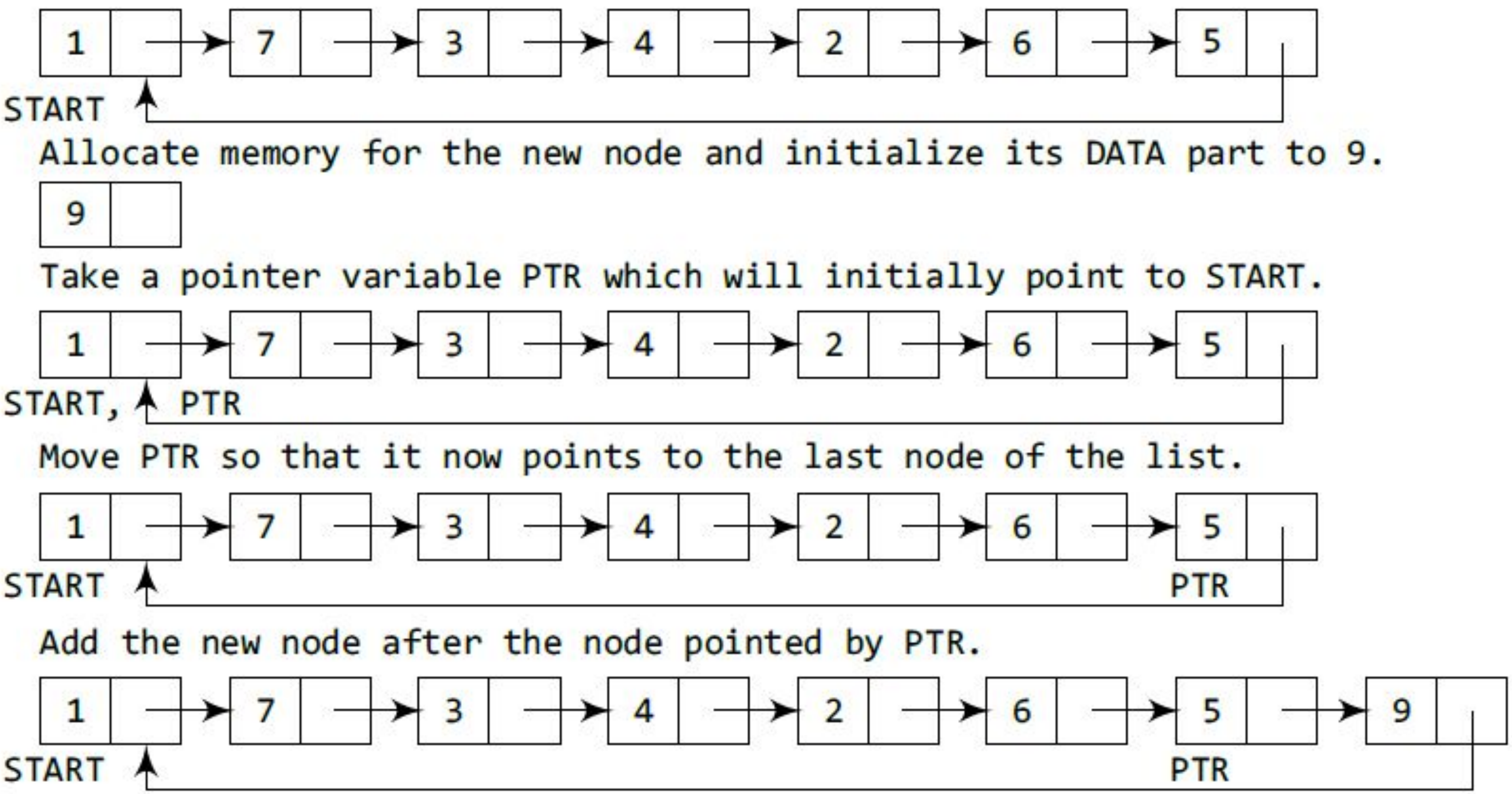


Figure 6.31 Inserting a new node at the end of a circular linked list

Algorithm

Step 1: IF AVAIL = NULL then
    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:   PTR = PTR -> NEXT
  [END OF LOOP]
Step 9: SET PTR -> NEXT = New_Node
Step 10: EXIT

DELETIONS IN CIRCULAR LINKED LIST

Delete node from Circular LL

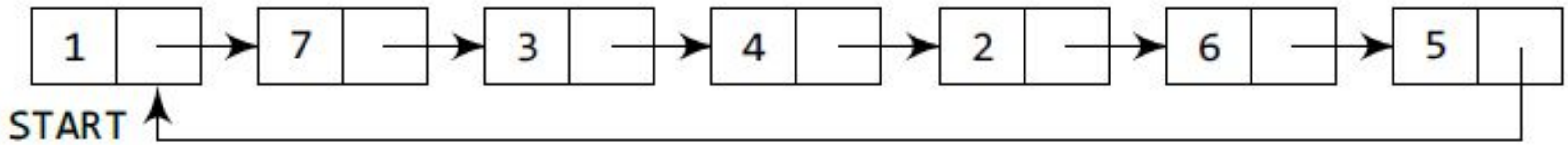
Case 1: The first node is deleted

Case 2: The last node is deleted

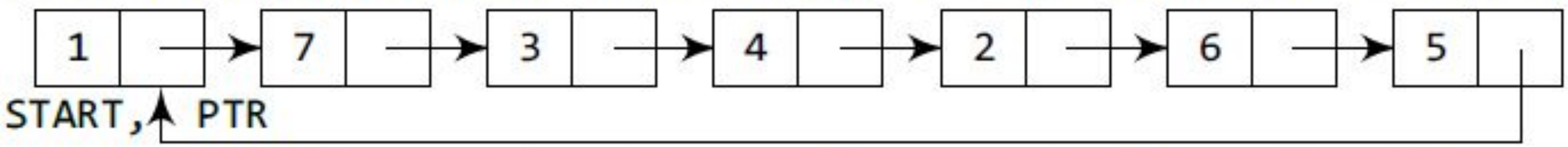
Deletion Case 1:

The first node is deleted

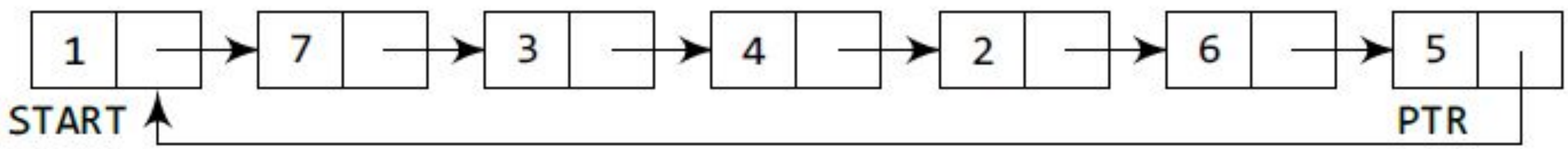
Example



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.

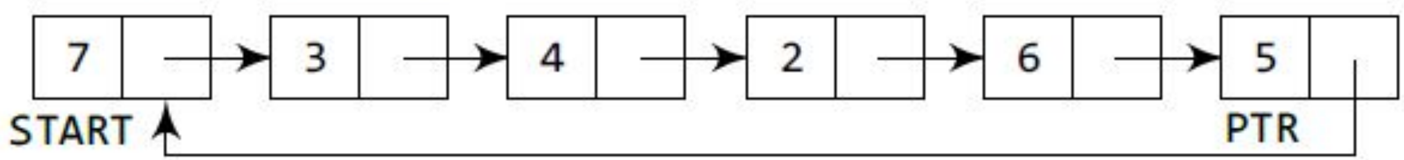


Figure 6.33 Deleting the first node from a circular linked list

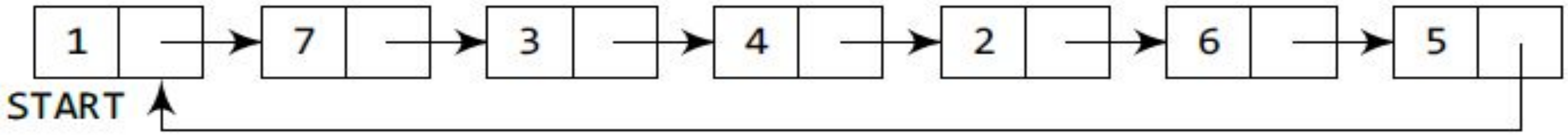
Algorithm

Step 1: IF START = NULL then
    Write UNDERFLOW
    GO TO Step 9
  [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: START->NEXT = NULL
Step 8: SET START = PTR -> NEXT
Step 9: EXIT

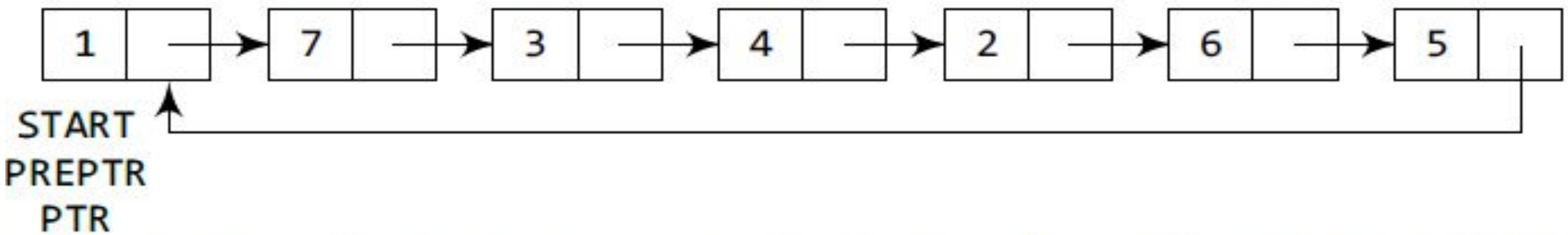
Deletion Case 2:

The last node is deleted

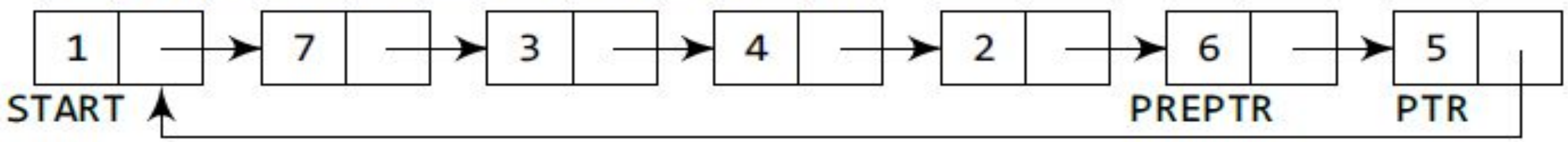
Example



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.

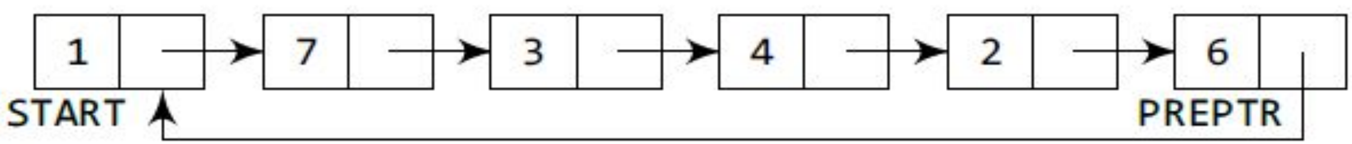


Figure 6.35 Deleting the last node from a circular linked list

Algorithm

Step 1: IF START = NULL then
    Write UNDERFLOW
    GO TO Step 10
  [END OF IF]
Step 2: SET PTR = START , PREPTR = PTR
Step 3: Repeat Step 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: PTR->NEXT = NULL
Step 9: PTR = NULL
Step 10: EXIT

DOUBLY LINKED LIST

Doubly Linked List

- In Doubly-linked list each node has two links: one points to previous node and one points to next node.

- The previous link of first node in the list points to a Null and the next link of last

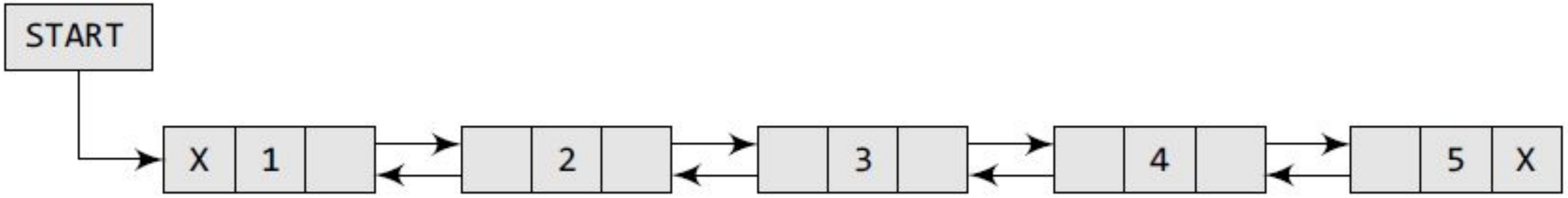


Figure 6.37 Doubly linked list

Advantages

- Traversal in *either direction* becomes convenient.
- *Reduces time* requirement for the program execution.
- The doubly linked lists can be used to represent other data-structure. The hierarchical structure of the *tree* can be easily represented using a doubly linked list. *Graphs* can be represented using doubly linked list.

Disadvantages

- Each node requires *extra space* for storing the additional pointer.
- While manipulating the lists, extra care should be taken to manipulate *both links*.

In C, the structure of a doubly linked list can be given as,

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

START

1

	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

Figure 6.38 Memory representation of a doubly linked list

INSERTIONS IN DOUBLY LINKED LIST

Inserting a new node in DLL

Case 1: New node is inserted at the beginning

Case 2: New node is inserted at the end

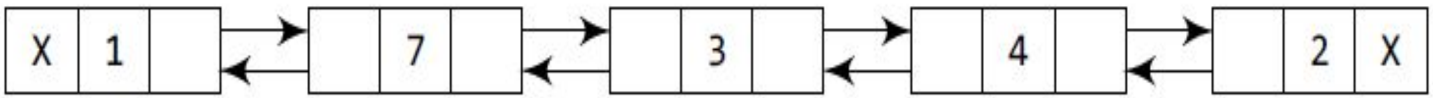
Case 3: New node is inserted after a given node

Case 4: New node is inserted before a given node

Insertion Case 1:

New node is inserted at the beginning

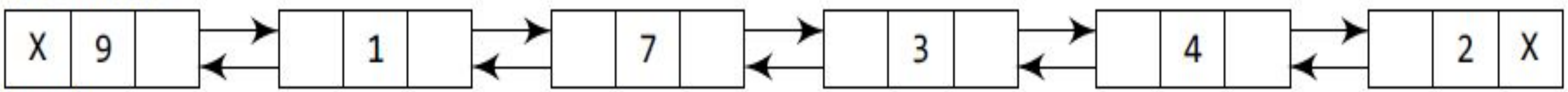
Example



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

Figure 6.39 Inserting a new node at the beginning of a doubly linked list

Algorithm

Step 1: IF AVAIL = NULL, then
 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

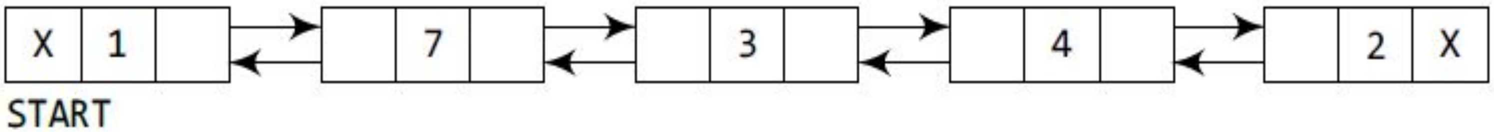
Step 8: SET START = New_Node

Step 9: Exit

Insertion Case 2:

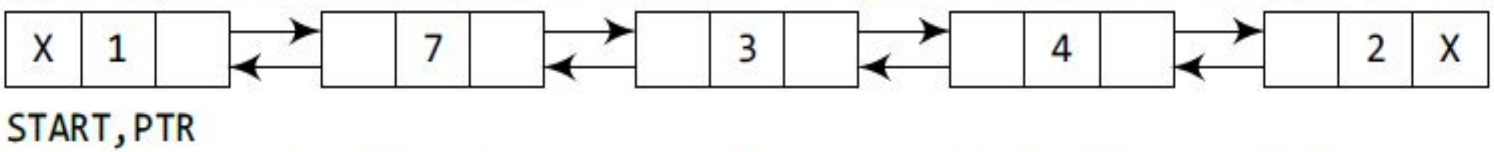
New node is inserted at the end

Example



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.



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

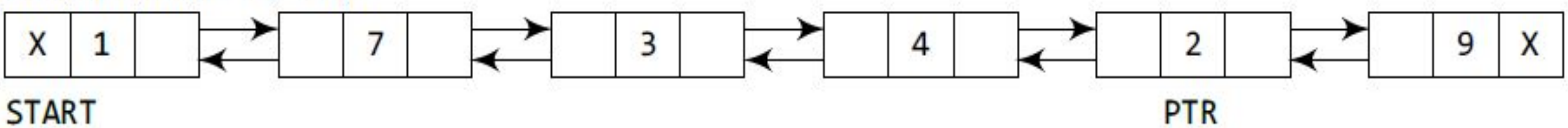


Figure 6.41 Inserting a new node at the end of a doubly linked list

Algorithm

Step 1: IF AVAIL = NULL, then
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, New_Node -> PREV= 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: New_Node -> PREV = PTR
Step 11: Exit

Insertion Case 3:

New node is inserted after a given node

Example

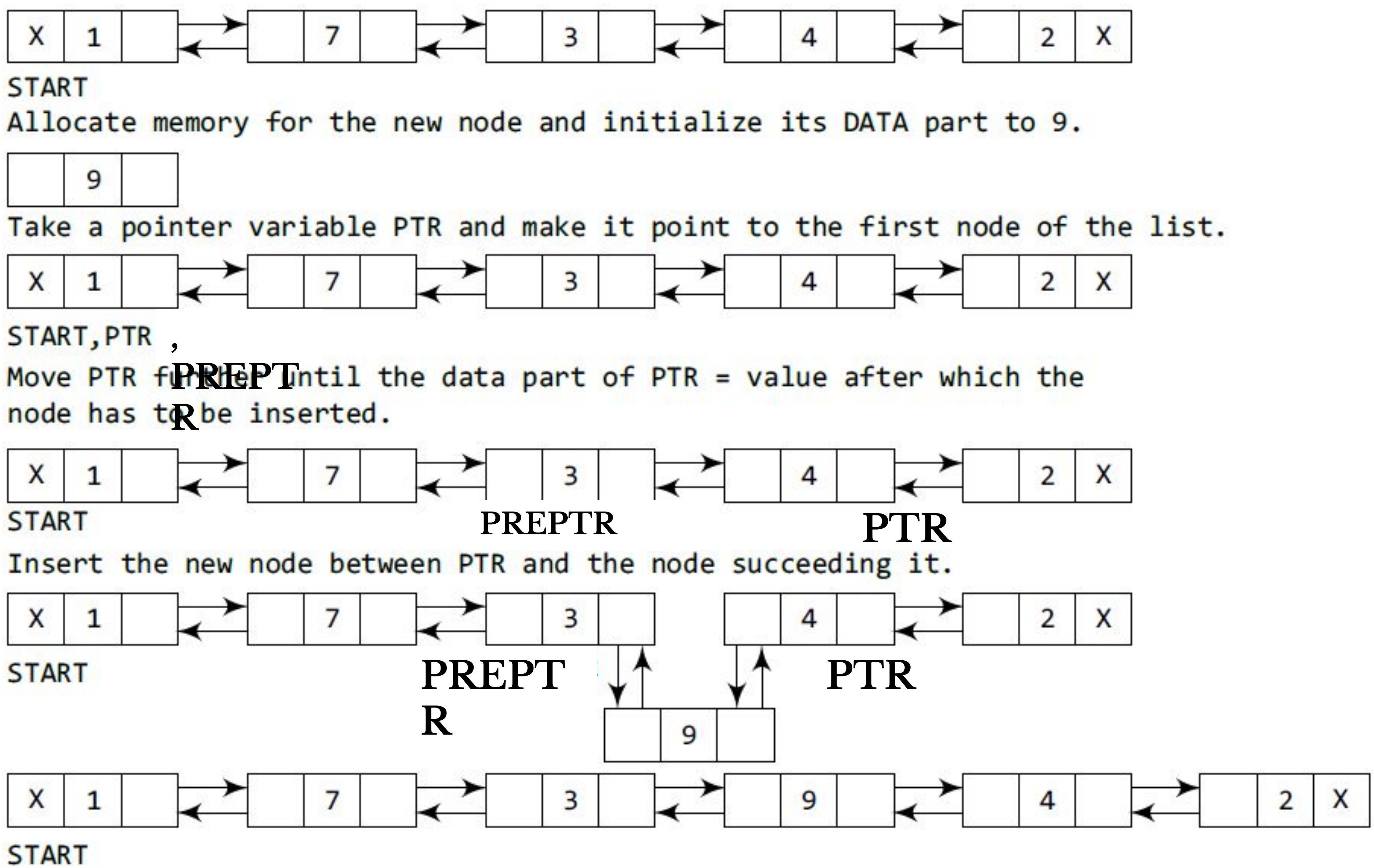


Figure 6.44 Inserting a new node after a given node in a doubly linked list

Algorithm

```
Step 1: IF AVAIL = NULL, then
                Write OVERFLOW
                Go To Step 14
        [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, New_Node -> PREV= NULL
Step 6: SET PTR = START
Step 7: SET PREPTR = PTR
Step 8: Repeat Step 9 while PREPTR-> DATA != NUM
Step 9:         SET PREPTR = PTR
                SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 10: SET New_Node -> NEXT = PTR
Step 11: SET New_Node -> PREV = PREPTR
Step 12: SET PREPTR -> NEXT = New_Node
Step 13: SET PTR-> PREV = New_Node
Step 14: Exit
```

Insertion Case 4:

New node is inserted before a given node

Example

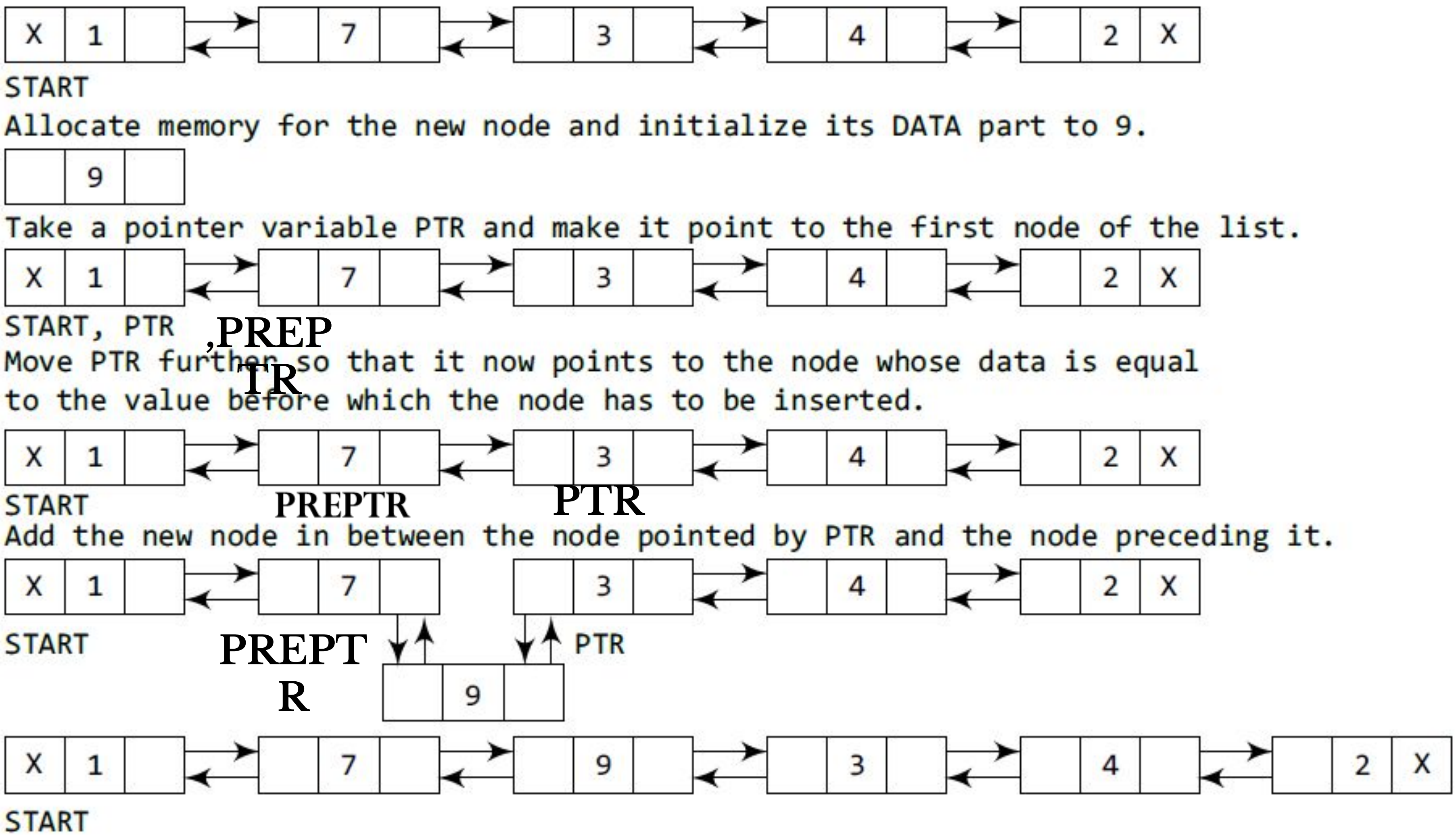


Figure 6.46 Inserting a new node before a given node in a doubly linked list

Algorithm

Step 1: IF AVAIL = NULL, then
    Write OVERFLOW
    Go To Step 14
  [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, New_Node -> PREV= NULL
Step 6: SET PTR = START
Step 7: SET PREPTR = PTR
Step 8: Repeat Step 9 while PTR-> DATA != NUM
Step 9: SET PREPTR = PTR
    SET PTR = PTR -> NEXT
  [END OF LOOP]
Step 10: SET New_Node -> NEXT = PTR
Step 11: SET New_Node -> PREV = PREPTR
Step 12: SET PREPTR -> NEXT = New_Node
Step 13: SET PTR -> PREV = New_Node
Step 14: Exit

DELETIONS IN DOUBLY LINKED LIST

Deleting a node from DLL

Case 1: The first node is deleted

Case 2: The last node is deleted

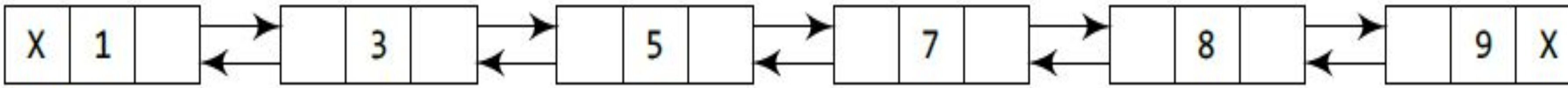
Case 3: The node after a given node is deleted

Case 4: The node before a given node is deleted

Deletion Case 1:

The first node is deleted

Example



START ,PTR

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

START

Figure 6.47 Deleting the first node from a doubly linked list

Algorithm

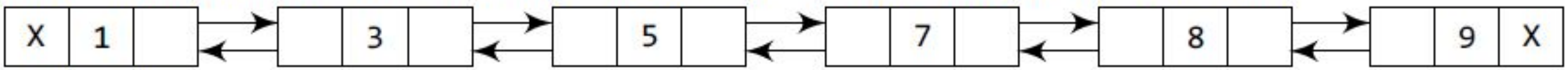
Step 1: IF START = NULL, then
            Write UNDERFLOW
            Go to Step 9
            [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: SET PTR -> PREV = NULL
Step 7: SET PTR -> NEXT = NULL
Step 8: PTR = NULL
Step 9: Exit

Deletion Case 2:

The last node is deleted

Example

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.

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

Figure 6.49 Deleting the last node from a doubly linked list

Algorithm

```
Step 1: IF START = NULL, then
            Write UNDERFLOW
            Go to Step 11
            [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 while PTR->NEXT != NULL
Step 5:           SET PREPTR = PTR
              SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: SET PTR -> PREV = NULL
Step 9: SET PTR -> NEXT = NULL
Step 10: PTR = NULL
Step 11: Exit
```

Deletion Case 3:

The node after a given node is deleted

Example

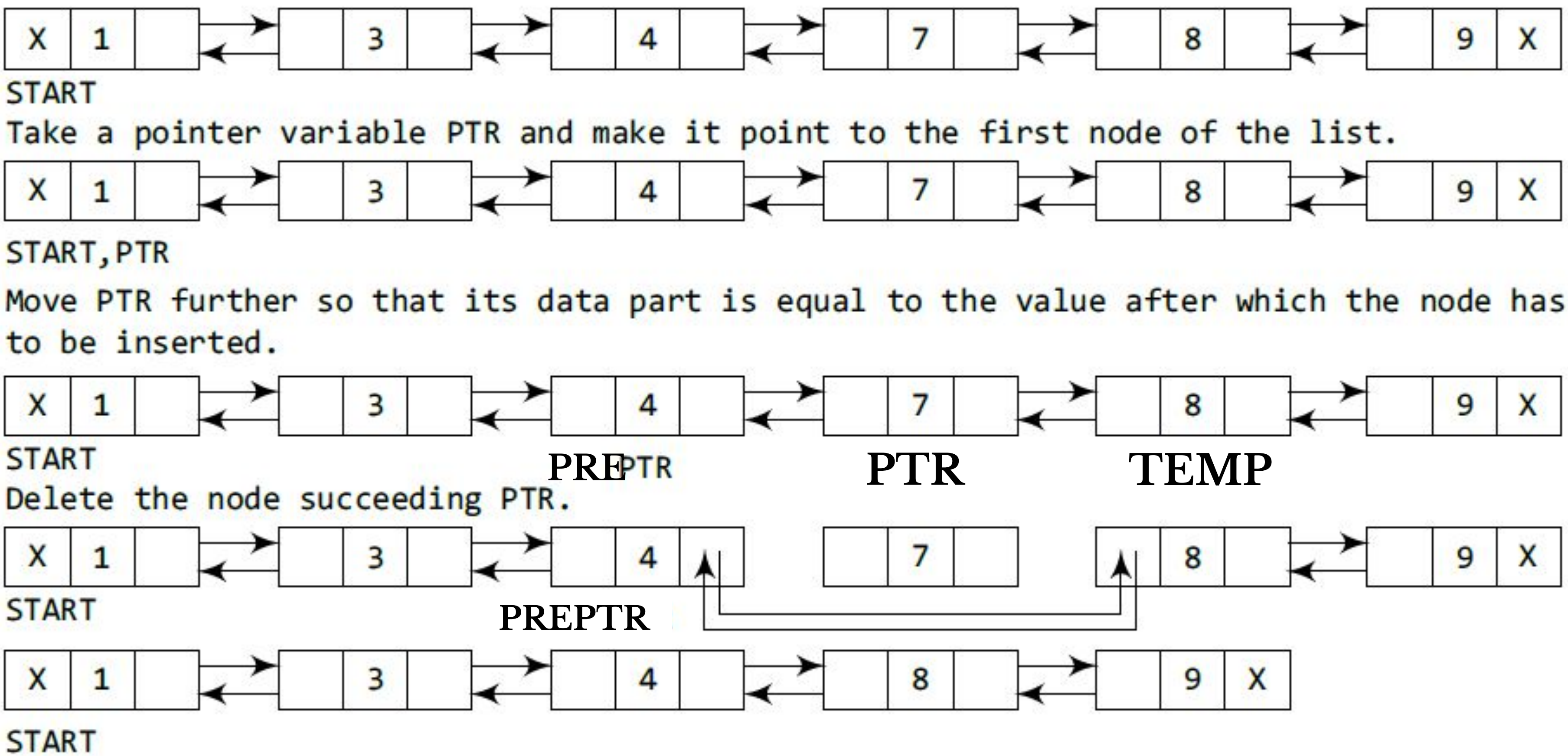


Figure 6.51 Deleting the node after a given node in a doubly linked list

Algorithm

```
Step 1: IF START = NULL, then
            Write UNDERFLOW
            Go to Step 13
            [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 while PREPTR->DATA != NUM
Step 5:         SET PREPTR = PTR
                SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 6: SET TEMP = PTR -> NEXT
Step 7: SET PREPTR -> NEXT = TEMP
Step 8: SET TEMP -> PREV = PREPTR
Step 9: FREE PTR
Step 10: SET PTR -> PREV = NULL
Step 11: SET PTR -> NEXT = NULL
Step 12: PTR = NULL
Step 13: Exit
```

Deletion Case 4:

The node before a given node is deleted

Example

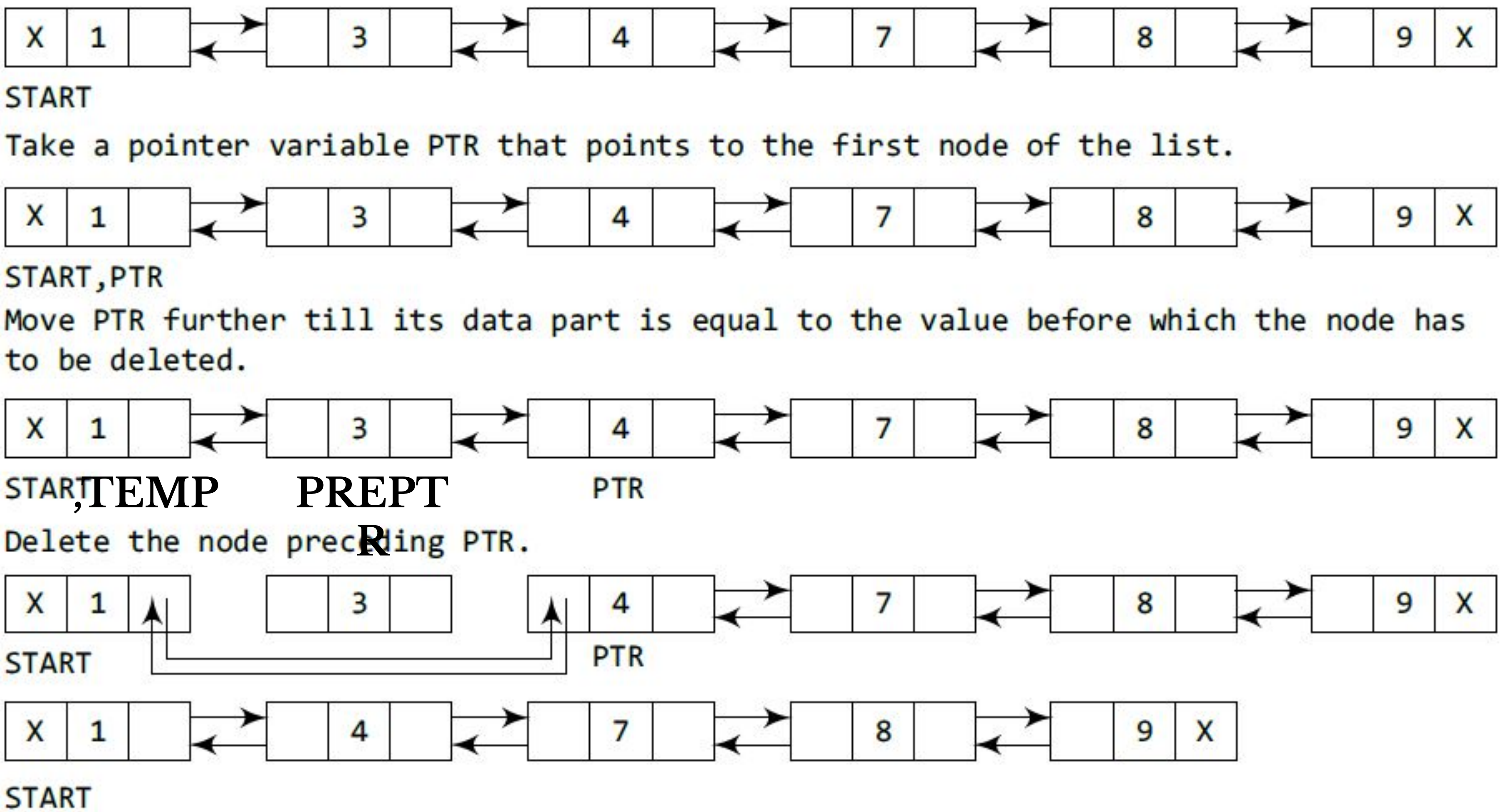


Figure 6.53 Deleting a node before a given node in a doubly linked list

Algorithm

```
Step 1: IF START = NULL, then
            Write UNDERFLOW
            Go to Step 13
            [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Step 5 while PTR->DATA != NUM
Step 5:         SET PREPTR = PTR
                SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: SET TEMP = PREPTR -> PREV
Step 7: SET TEMP -> NEXT = PTR
Step 8: SET PTR -> PREV = TEMP
Step 9: FREE PREPTR
Step 10: SET PREPTR -> PREV = NULL
Step 11: SET PREPTR -> NEXT = NULL
Step 12: PREPTR = NULL
Step 13: Exit
```

Thank You