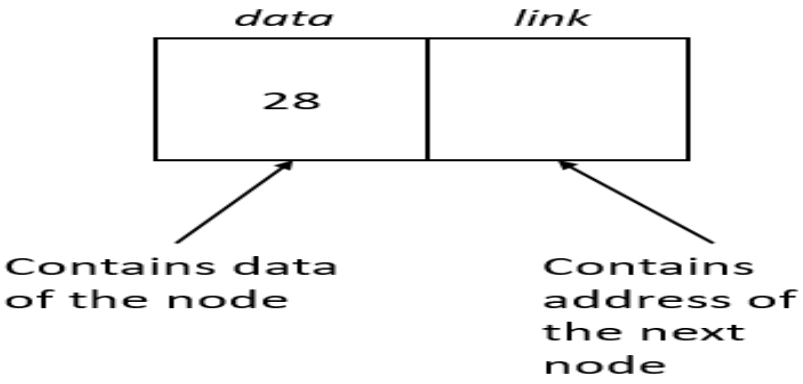
Linear / Linked List

- **Types of linear/linked lists:**
  1) Single Linked List
  2) Doubly Linked List
  3) Circular Linked List
- **Single Linked List :**
  - A linked list is a linear collection of data elements. These data elements are called nodes.
  - In single linked list every node contains two fields, data field and link field -a pointer to the next node/address of next node.

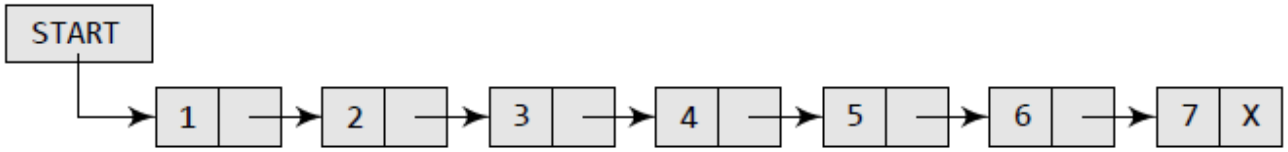


Single Linked List....

- The node in a single linked list is declared as

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

- The last node address field in the single linked list contains NULL.



- **Operations performed on a single linked list:**

I. Insertion

II. Deletion

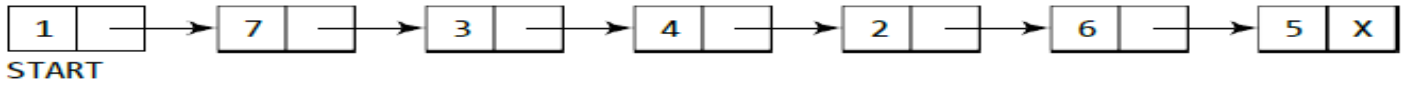
III. Searching

IV. Traversing

Single Linked List....

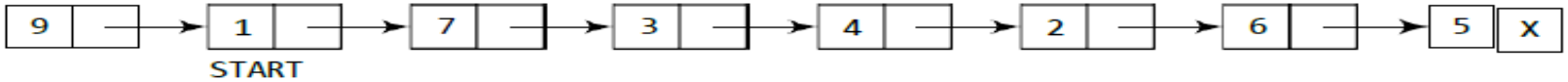
- **Insertion:**

Case 1: The new node is inserted at the beginning

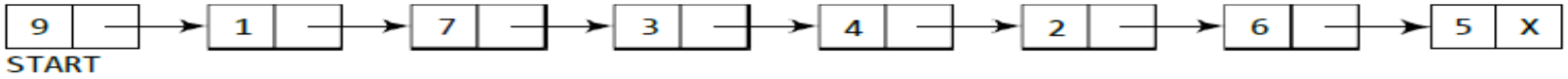


Allocate memory for the new node and initialize its DATA part to 9.

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.



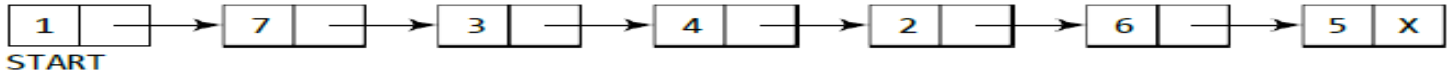
Now make START to point to the first node of the list.




```
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

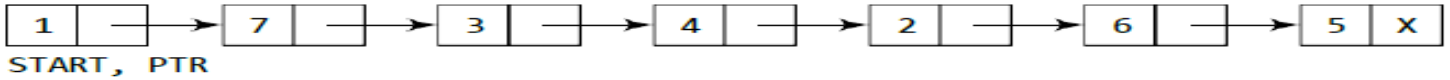
Single Linked List....

Case 2: The new node is inserted at the end

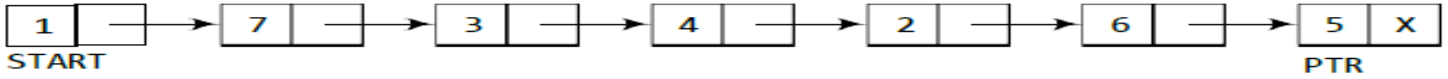


Allocate memory for the new node and initialize its DATA part to 9 and NEXT part to NULL.

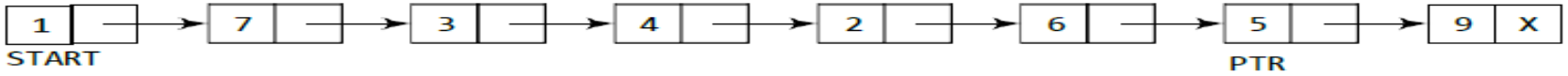
Take a pointer variable PTR which points to START.



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



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.

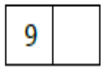



```
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

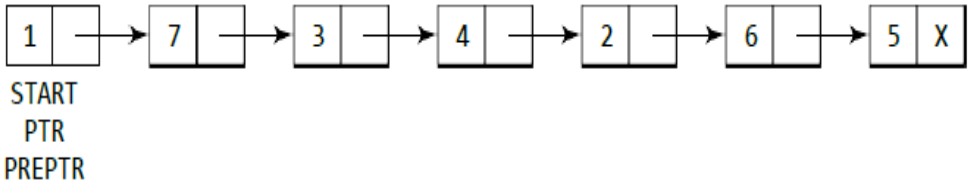
Single Linked List....

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

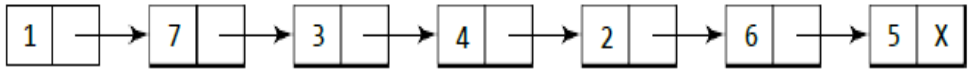
Allocate memory for the new node and initialize its DATA part to 9.

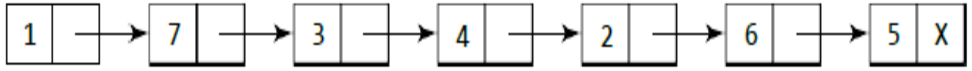


Take two pointer variables PTR and PREPTR and initialize them with START so that START, PTR, and PREPTR point to the first node of the list.

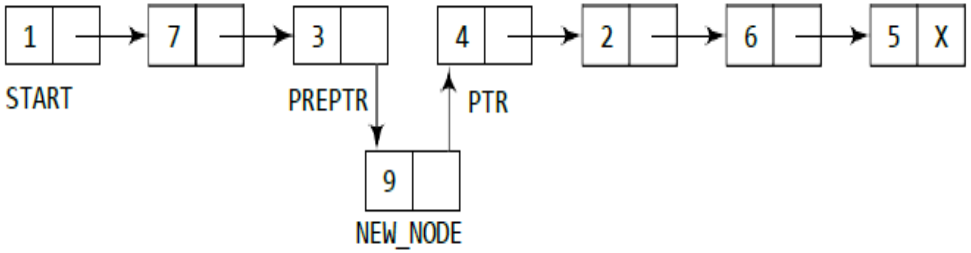


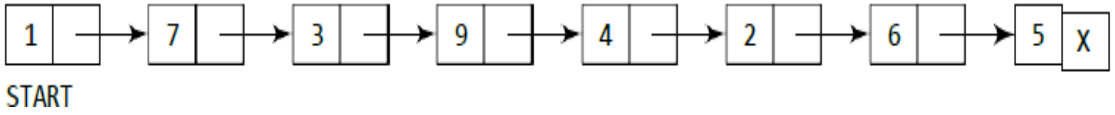
Move PTR and PREPTR until the DATA part of PREPTR = value of the node after which insertion has to be done. PREPTR will always point to the node just before PTR.





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



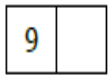



```
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
        != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT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

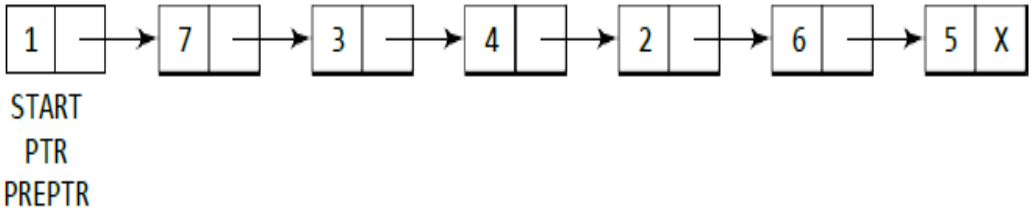
Single Linked List....

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

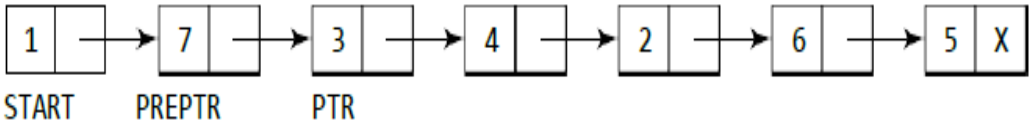
Allocate memory for the new node and initialize its DATA part to 9.



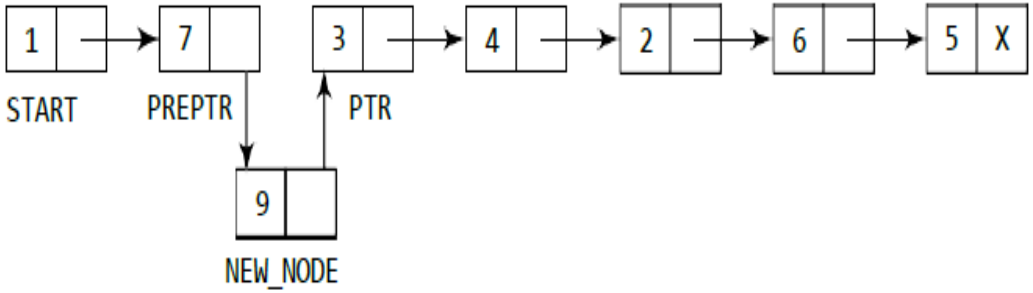
Initialize PREPTR and PTR to the START node.

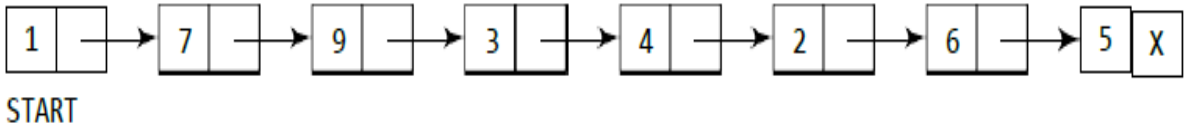


Move PTR and PREPTR until the DATA part of PTR = value of the node before which insertion has to be done. PREPTR will always point to the node just before PTR.



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

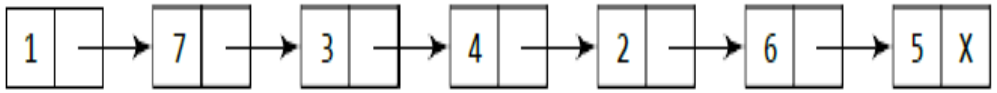





```
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR -> DATA != NUM
Step 8:      SET PREPTR = PTR
Step 9:      SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step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

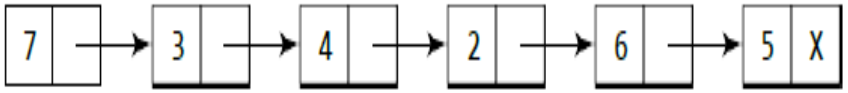
Single Linked List....

- **Deletion : Case 1: The first node is deleted**



START

Make START to point to the next node in sequence.

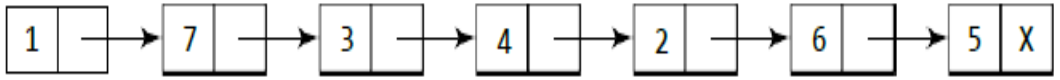


START

```
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
```

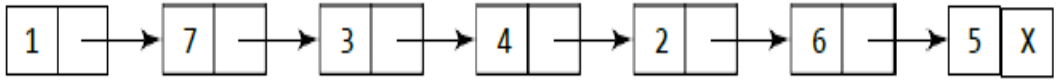
Case 2: The last node is deleted

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



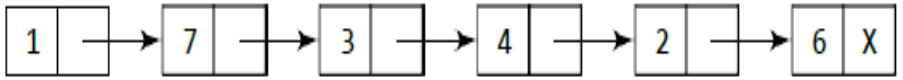
START
PREPTR
PTR

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.



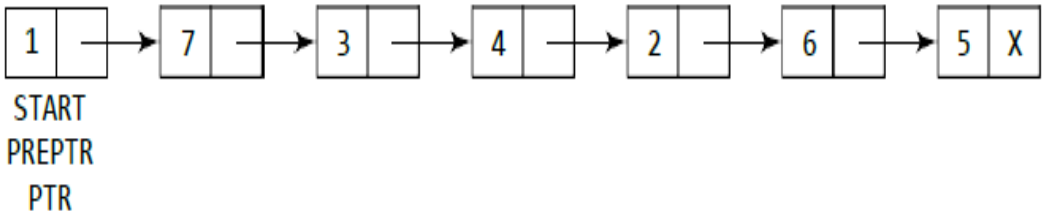
START

```
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
```

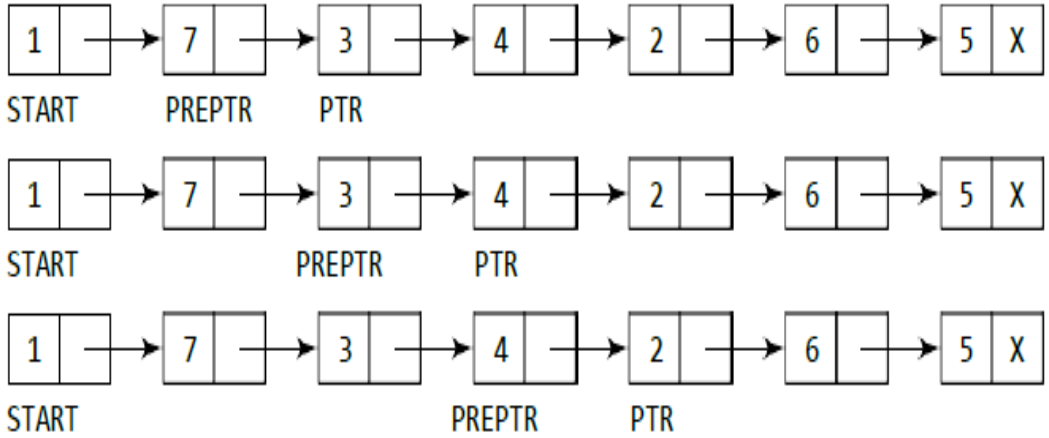
Single Linked List....

Case 3: The node after a given node is deleted

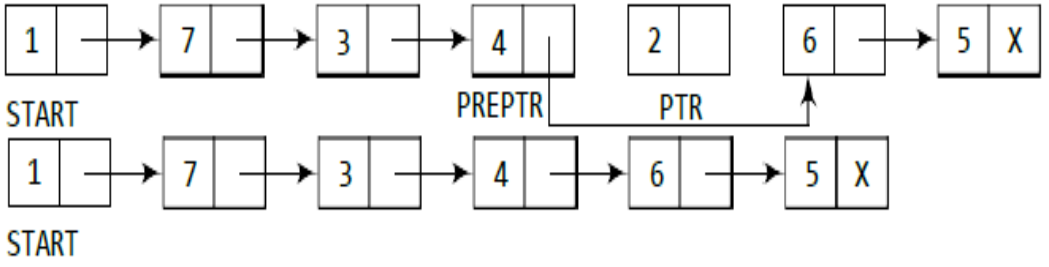
Take pointer variables PTR and PREPTR which initially point to START.



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



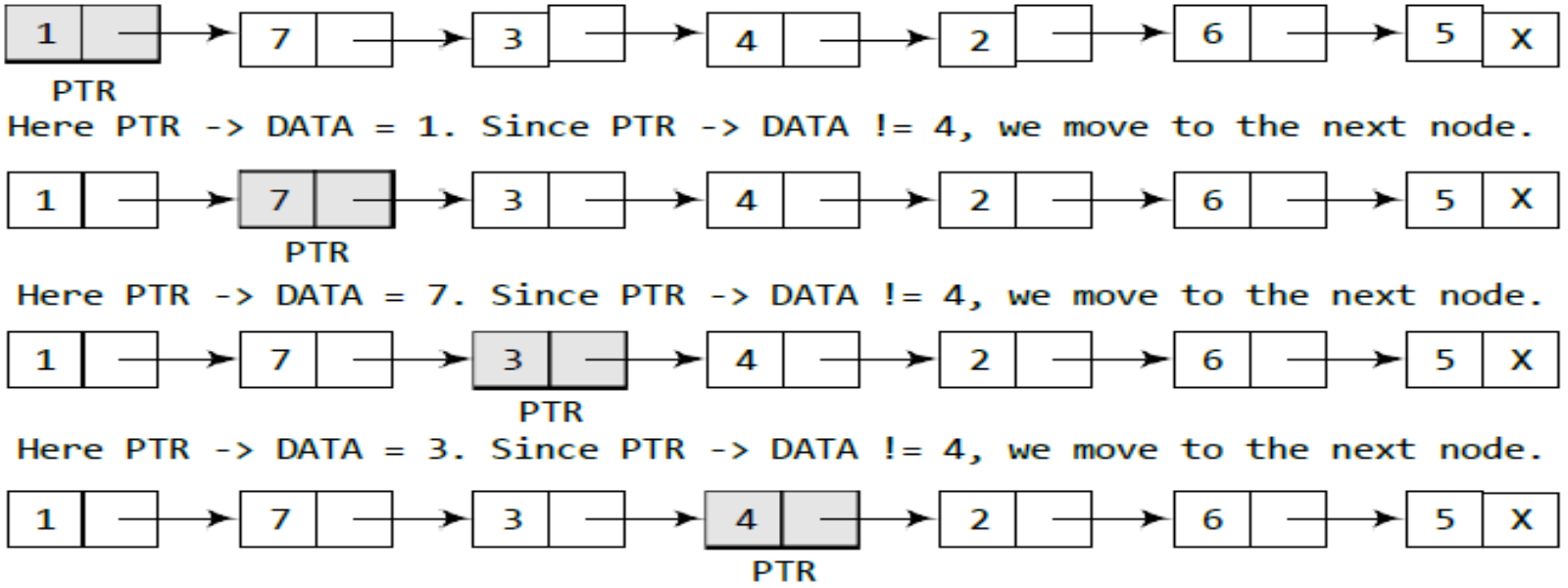
Set the NEXT part of PREPTR to the NEXT part of PTR.




```
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
```

Single Linked List....

- **Searching:**




```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:      IF VAL = PTR -> DATA
                   SET POS = PTR
                   Go To Step 5
             ELSE
                   SET PTR = PTR -> NEXT
             [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

Single Linked List....

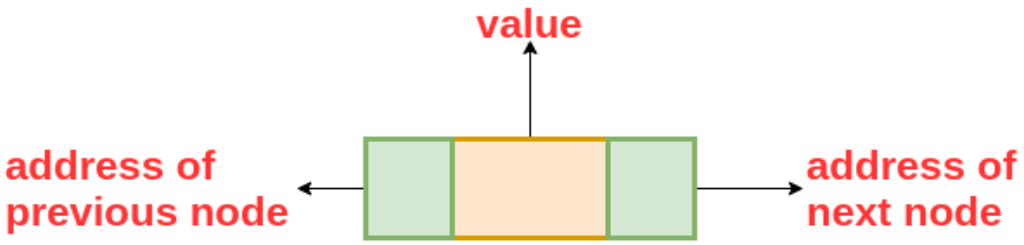
- **Traversing:**

Traversing a linked list means accessing the nodes of the list in order to perform some processing on them.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:           Apply Process to PTR->DATA
Step 4:SET PTR = PTR->NEXT [END OF
       LOOP]
Step 5: EXIT
```

Doubly Linked List

- 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.
- It is a collection of node , in which each node will contain three fields- a pointer to the previous node ,data, a pointer to the next node.



Node in DLL

- The declaration of node in double linked list is given as

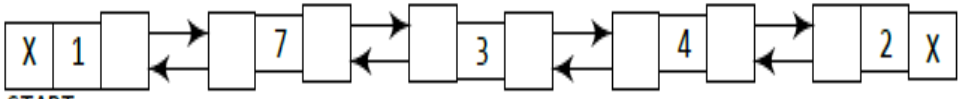
```
struct node
{
   struct node *prev;
   int data;
   struct node *next;
 };
```

- The PREV field of the first node and the NEXT field of the last node will contain NULL.
- The NEXT field is used to traverse the list in forward direction and PREV field is used to traverse the list in backward direction.

Doubly Linked List....

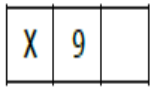
- **Operation on Doubly linked list are**
  - I. Insertion
  - II. Deletion
  - III. Searching
  - IV. Traversing
- **Insertion:**

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

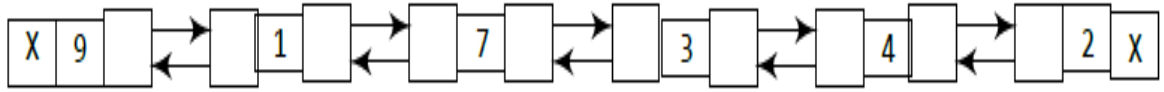


START

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

```
SET NEW_NODE -> DATA = VAL
SET NEW_NODE -> PREV = NULL
SET NEW_NODE -> NEXT = START
SET START -> PREV = NEW_NODE
SET START = NEW_NODE
EXIT
```

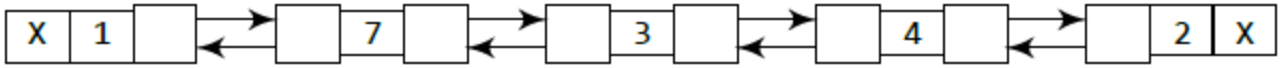
Doubly Linked List....

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

START

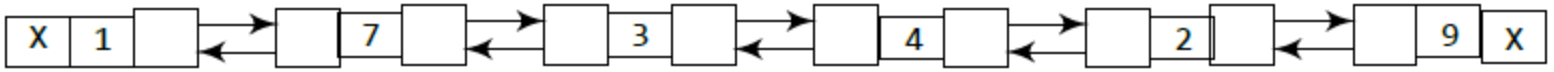
Allocate memory for the new node and initialize its DATA part to 9 and its NEXT field to NULL.

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.

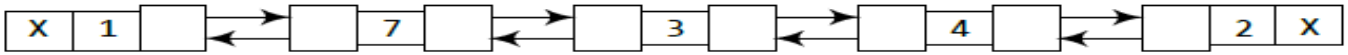


START PTR

```
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
```

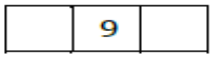
Doubly Linked List....

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

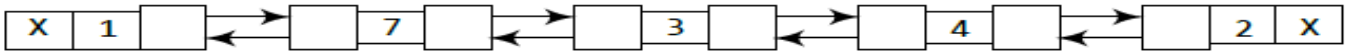


START

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

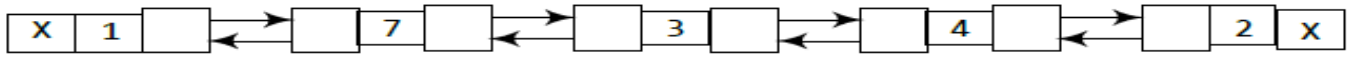


Take a pointer variable PTR and make it point to the first node of the list.



START,PTR

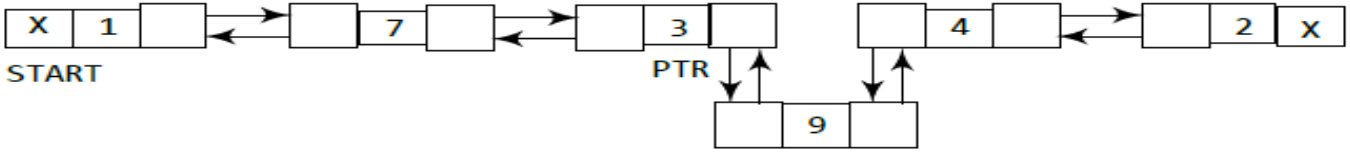
Move PTR further until the data part of PTR = value after which the node has to be inserted.

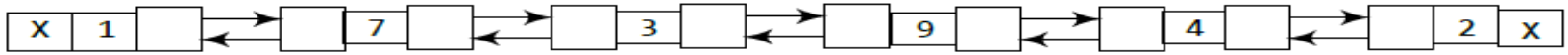


START

PTR

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



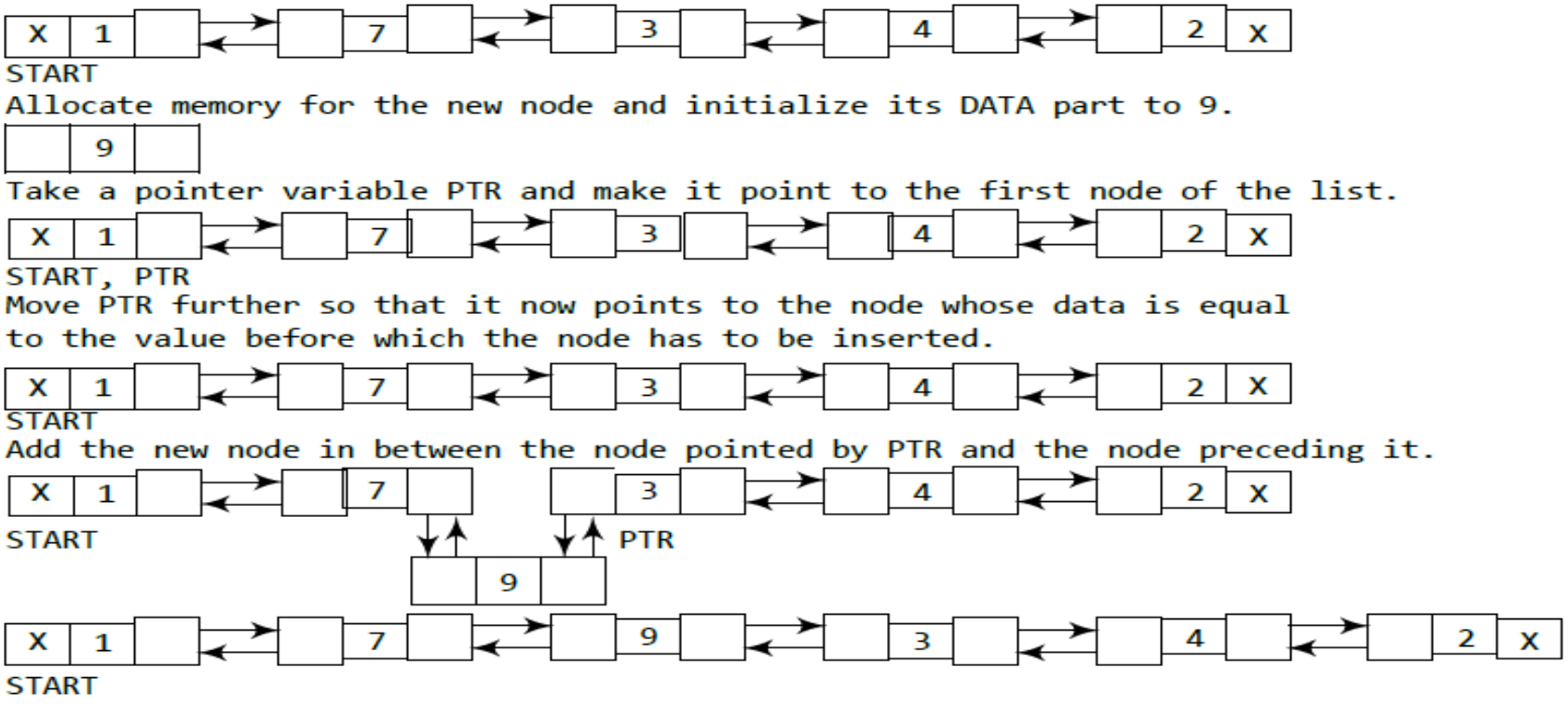


START

```
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
```

Doubly Linked List....

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

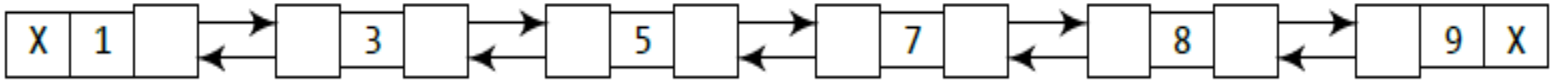



```
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10: SET PTR -> PREV = NEW_NODE
Step 11: SET PTR  -> PREV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

Doubly Linked List....

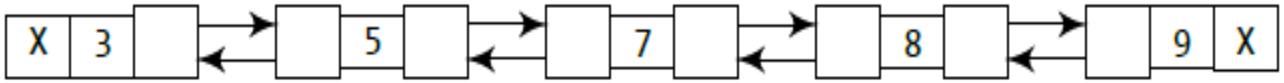
- **Deletion:**

Case 1: The first node is deleted.



START

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



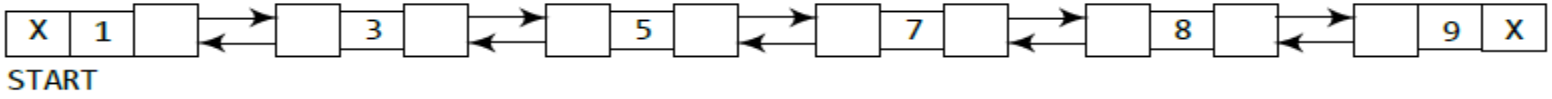
START

```
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
```

Doubly Linked List....

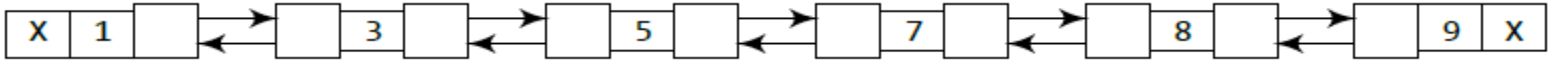
- **Deletion:**

Case 2: The last node is deleted.



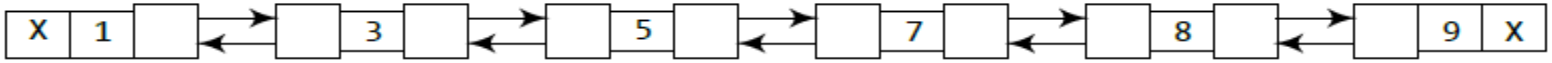
START

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.

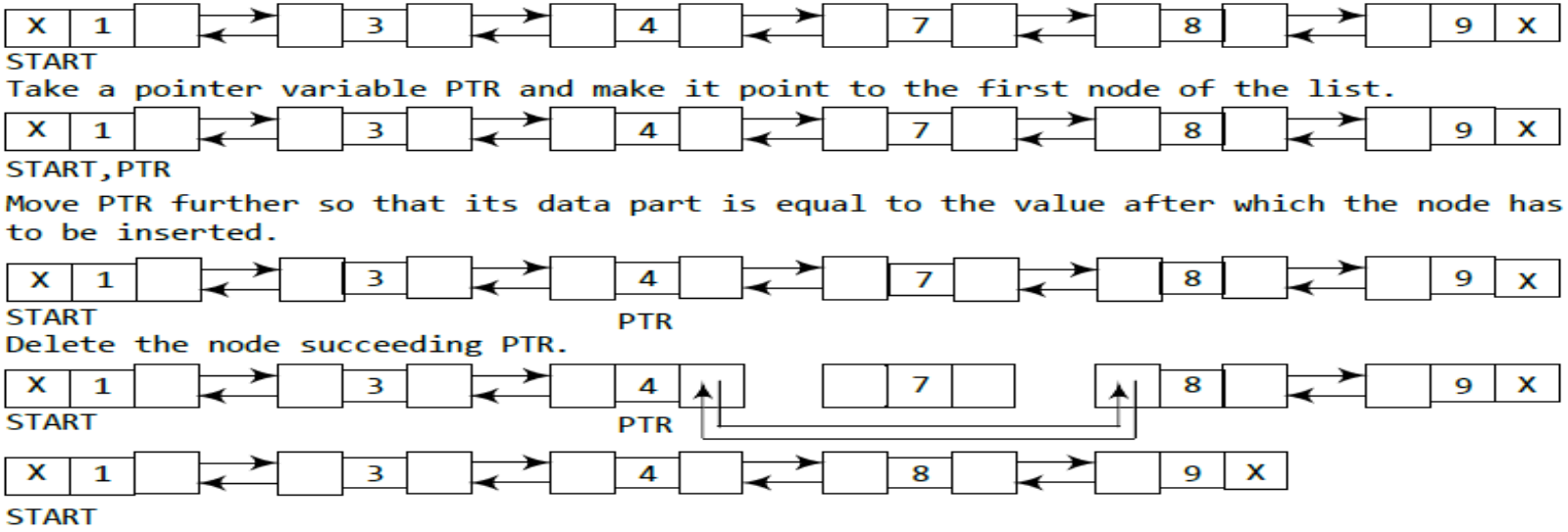
START

```
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
```

Doubly Linked List....

- **Deletion:**

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

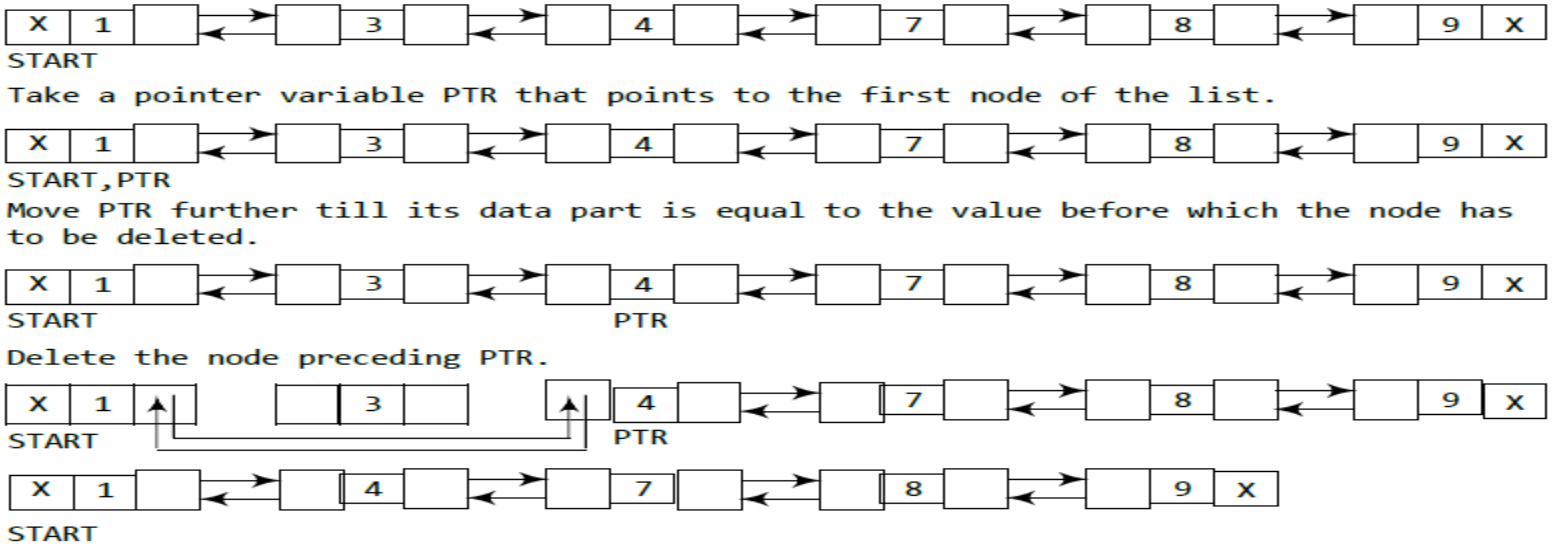



```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step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

Doubly Linked List....

- **Deletion:**

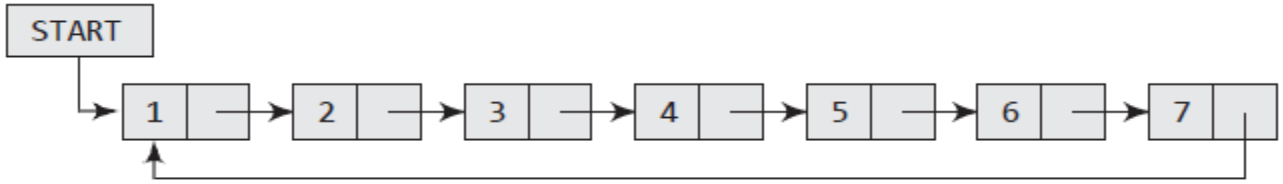
Case 4: The node before a given node is deleted.




```
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 Linked List

- **Circular Linked List Types:**
  I. Circular Single Linked List
  II. Circular Doubly Linked List
- **Circular Single Linked List:**
  - In a circular single linked list, the last node contains a pointer to the first node of the list i,e the last node address field contains the address of the first node.

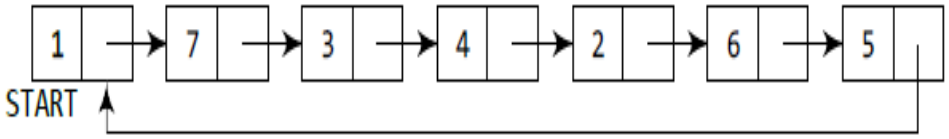


  - Operation on circular single linked list are:
    I. Insertion
    II. Deletion
    III. Searching
    IV. Traversing

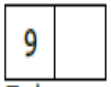
Circular Single Linked List.....

- **Insertion:**

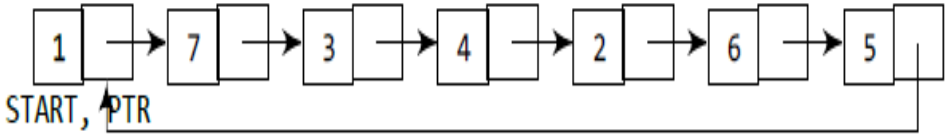
Case 1: The new node is inserted at the beginning of the circular linked list.



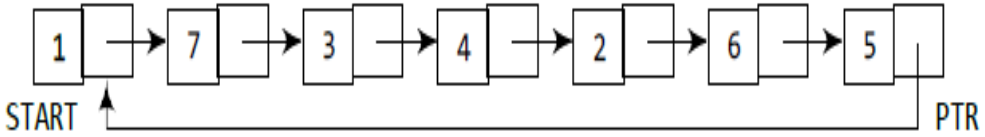
Allocate memory for the new node and initialize its DATA part to 9.



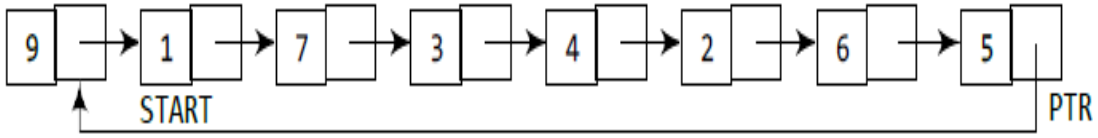
Take a pointer variable PTR that points to the START node of the list.



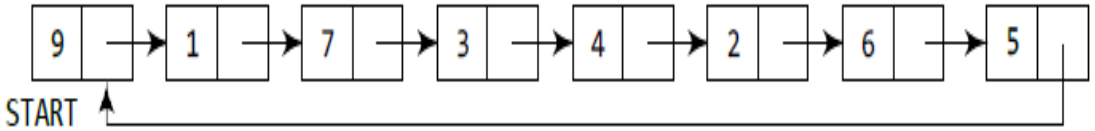
Move PTR so that it now points to the last node of the list.



Add the new node in between PTR and START.



Make START point to the new node.

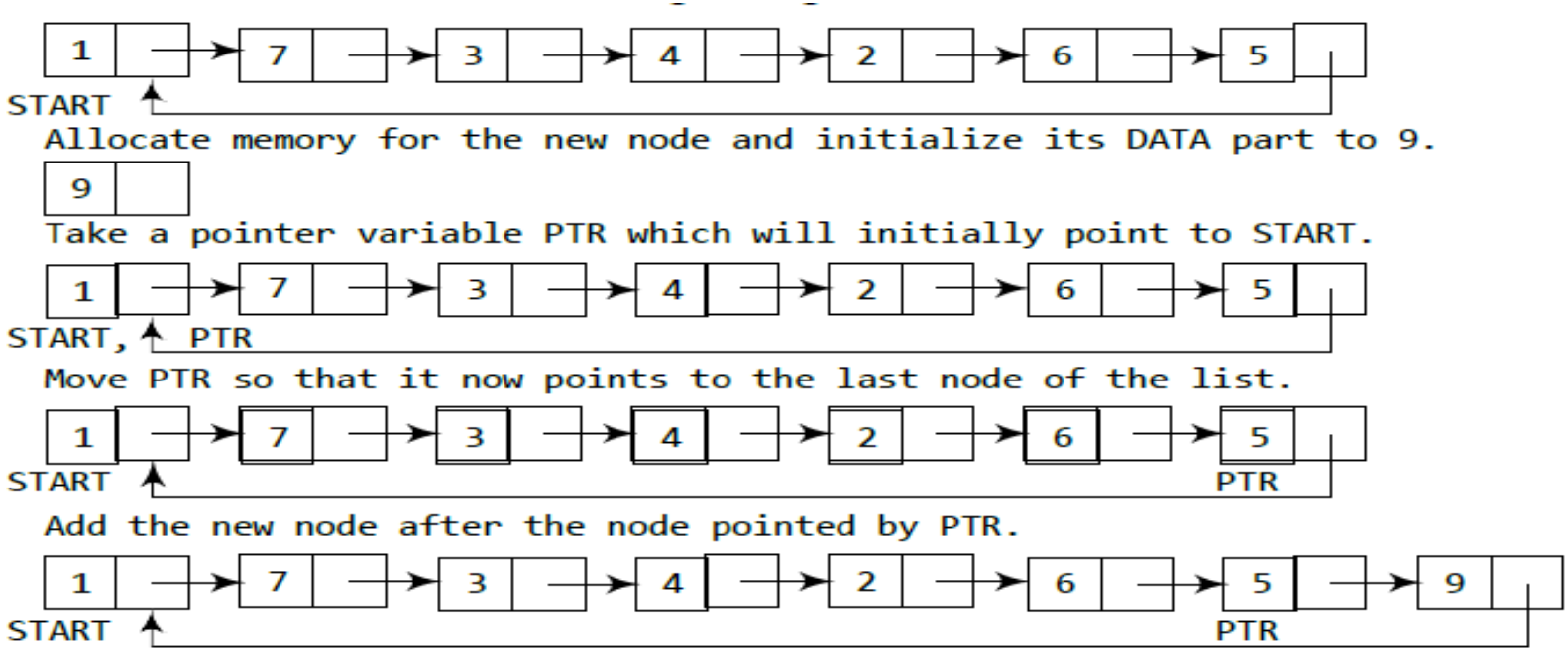



```
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:      PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

Circular Single Linked List.....

- **Insertion:**

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

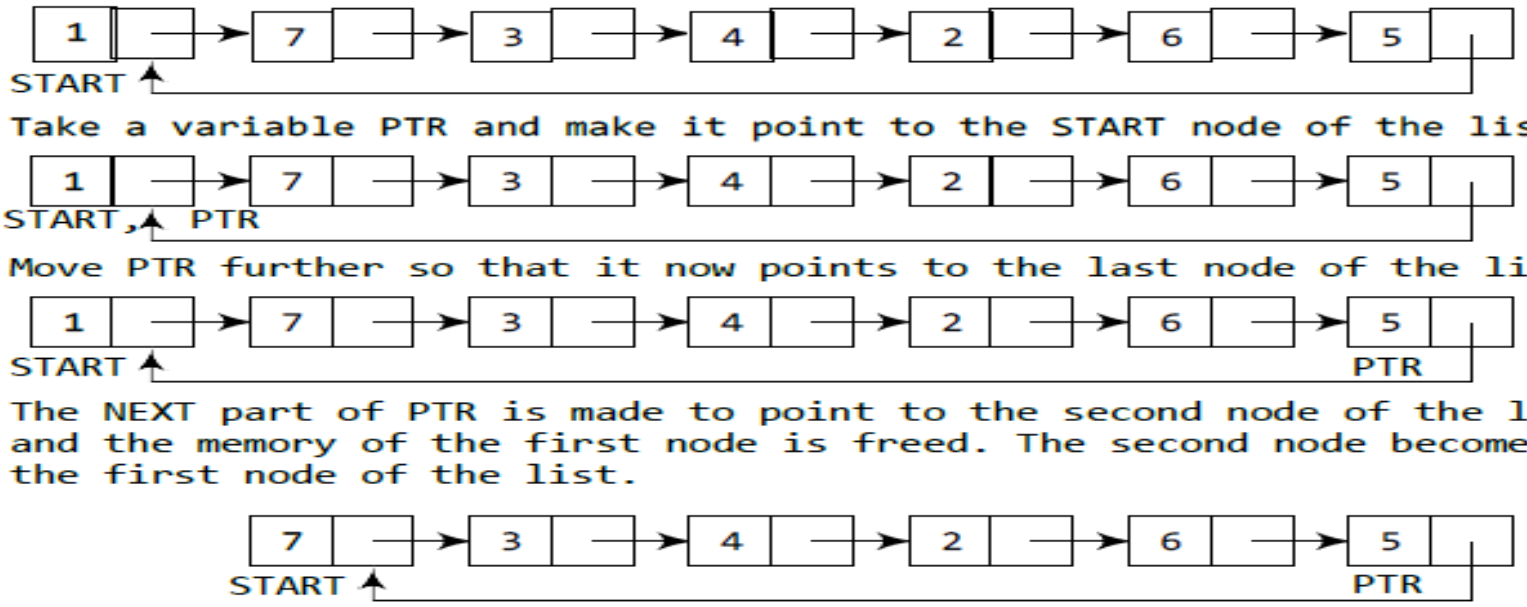



```
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

Circular Single Linked List.....

- **Deletion:**

Case 1: The first node is deleted.

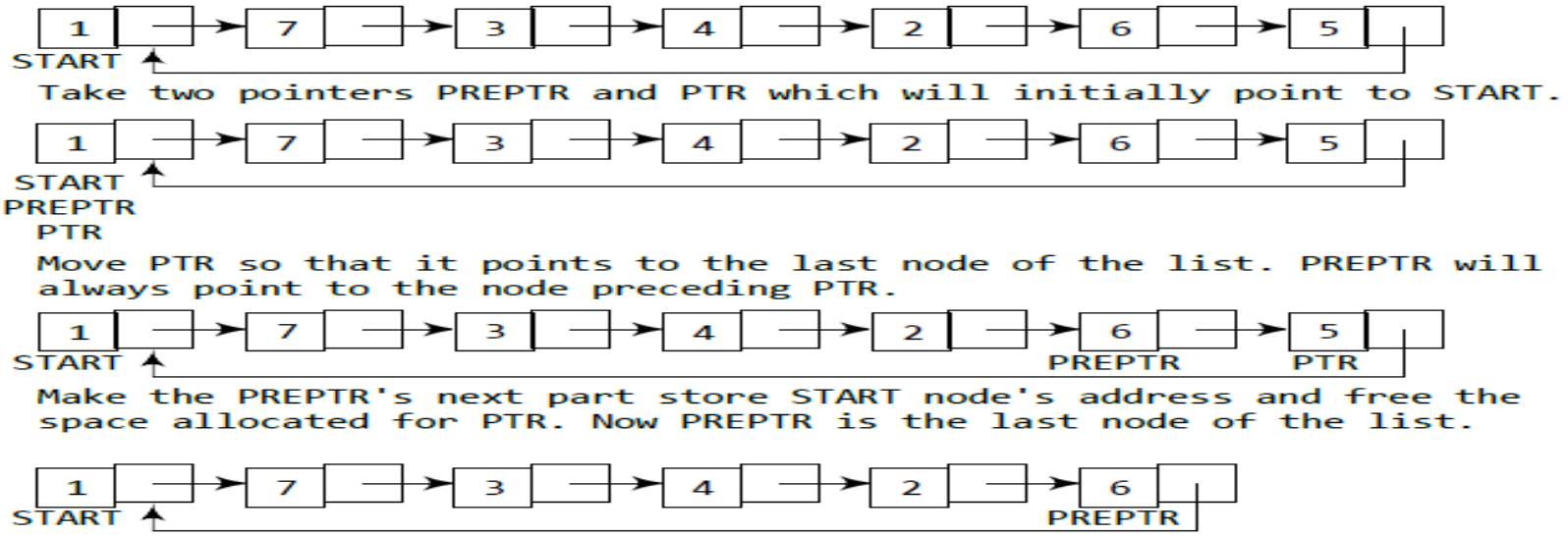



```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
         [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: FREE START
Step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

Circular Single Linked List.....

- **Deletion:**

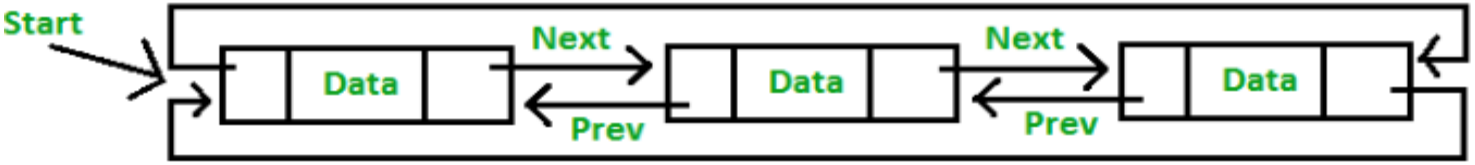
Case 2: The last node is deleted.




```
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
```

Circular Doubly Linked List

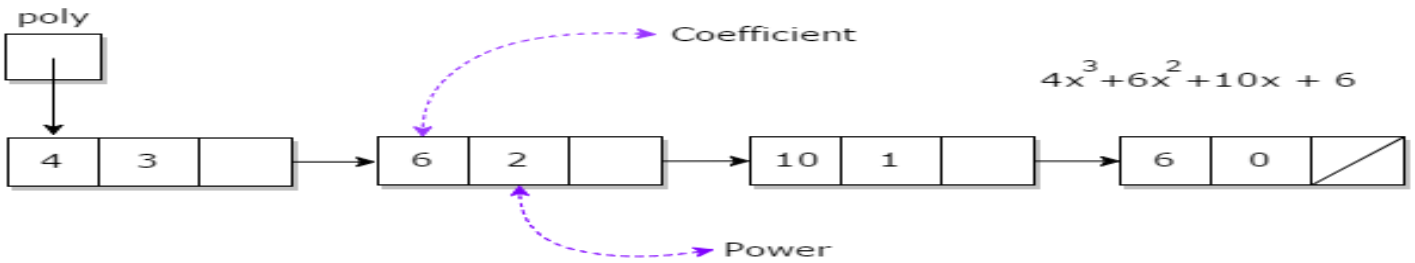
- **Circular Doubly Linked List:**
  - Circular doubly linked list doesn't contain NULL in any of the node.
  - The last node NEXT field of the list contains the address of the first node of the list.
  - The first node PREV field of the list contain address of the last node of the list.



  - **Operations:**
    I. Insertion
    II. Deletion
    III. Searching
    IV. Traversing
  - Implementation is more complex than other linked lists.

Applications of Linked Lists

- Polynomial Representation



- Implementation of different data structures like stack ,queues etc.
- Dynamic Memory allocation.
- Used in Image viewer-previous and next images are linked , hence we can access by using next and previous buttons.
- In Operating System, all the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running.
- Used in web browser to access the next and previous web pages while browsing.

****