Traversing in doubly linked list

Algorithm

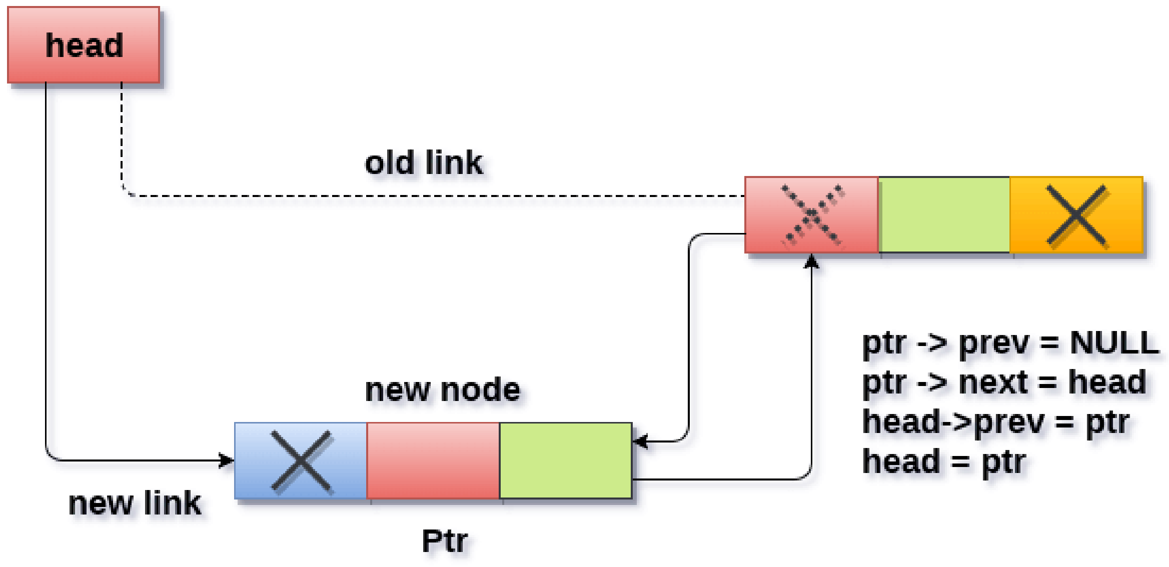
```
Step 1: IF HEAD == NULL
          WRITE "UNDERFLOW"
        GOTO STEP 6
        [END OF IF]
Step 2: Set PTR = HEAD
Step 3: Repeat step 4 and 5 while PTR != NULL
Step 4: Write PTR → data
Step 5: PTR = PTR → next
Step 6: Exit
```

Insertion in doubly linked list at beginning

As in doubly linked list, each node of the list contain double pointers therefore we have to maintain more number of pointers in doubly linked list as compare to singly linked list.

Algorithm:

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



```
ptr -> prev = NULL
ptr -> next = head
head->prev = ptr
head = ptr
```

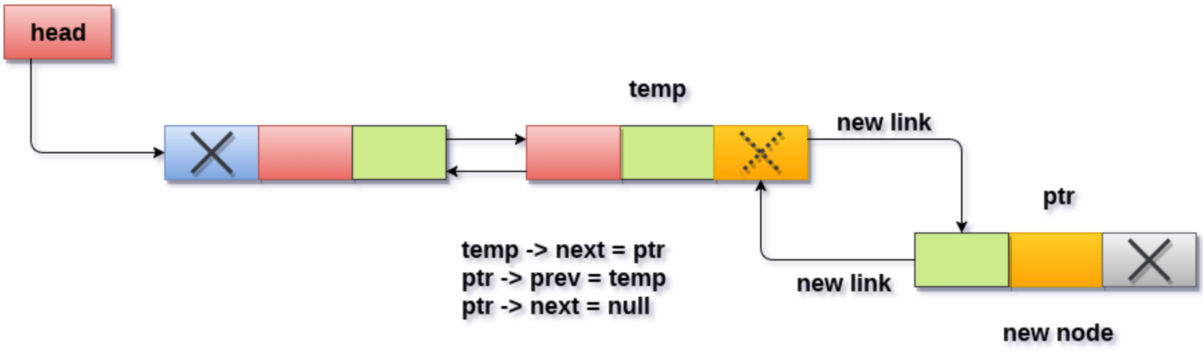
Insertion into doubly linked list at beginning

Insertion in doubly linked list at the end

Algorithm

```
○ Step 1: IF PTR = NULL
      Write OVERFLOW
       Go to Step 11
      [END OF IF]
○ Step 2: SET NEW_NODE = PTR
○ Step 3: SET PTR = PTR -> NEXT
○ Step 4: SET NEW_NODE -> DATA = VAL
○ Step 5: SET NEW_NODE -> NEXT = NULL
○ Step 6: SET TEMP = START
○ Step 7: Repeat Step 8 while TEMP -> NEXT != NULL
○ Step 8: SET TEMP = TEMP -> NEXT
     [END OF LOOP]
○ Step 9: SET TEMP -> NEXT = NEW_NODE
```

```
o   Step 10C: SET NEW_NODE -> PREV = TEMP
o   Step 11: EXIT
```



Insertion into doubly linked list at the end

Insertion in doubly linked list after Specified node

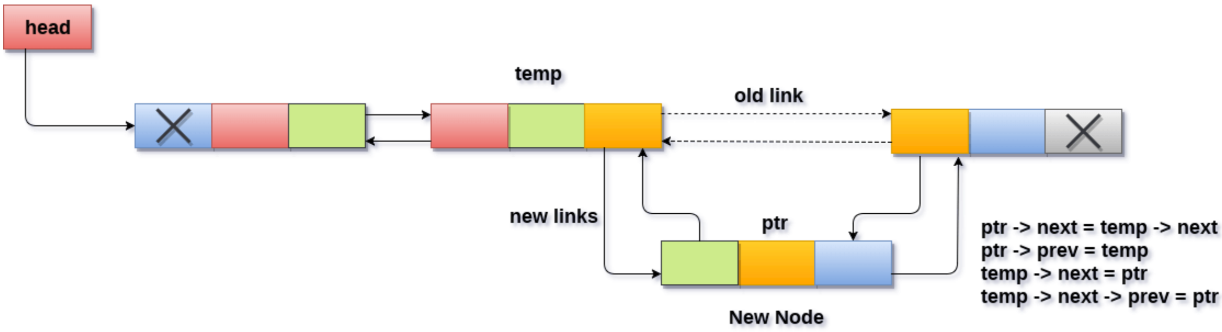
Algorithm

```
Step 1: IF PTR = NULL

          Write OVERFLOW
          Go to Step 15
        [END OF IF]
   o   Step 2: SET NEW_NODE = PTR
   o   Step 3: SET PTR = PTR -> NEXT
   o   Step 4: SET NEW_NODE -> DATA = VAL
   o   Step 5: SET TEMP = START
   o   Step 6: SET I = 0
   o   Step 7: REPEAT 8 to 10 until I
   o   Step 8: SET TEMP = TEMP -> NEXT
   o   STEP 9: IF TEMP = NULL
   o   STEP 10: WRITE "LESS THAN DESIRED NO. OF ELEMENTS"

          GOTO STEP 15
          [END OF IF]
        [END OF LOOP]
```

- **Step 11:** SET NEW_NODE -> NEXT = TEMP -> NEXT
- **Step 12:** SET NEW_NODE -> PREV = TEMP
- **Step 13:** SET TEMP -> NEXT -> PREV = NEW_NODE
- **Step 14 :** SET TEMP -> NEXT = NEW_NODE
- **Step 15:** EXIT



Insertion into doubly linked list after specified node

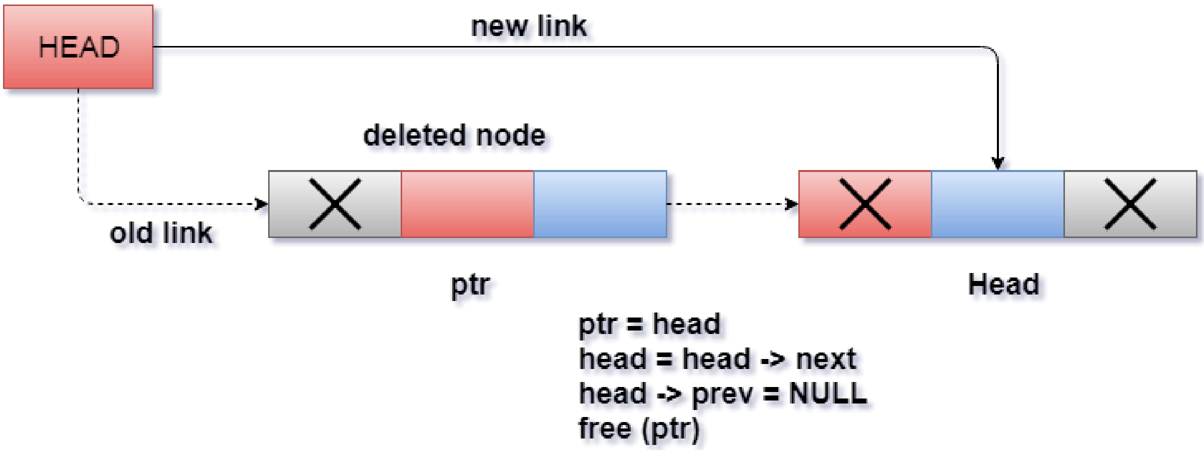
Deletion at beginning

Algorithm

- **STEP 1:** IF HEAD = NULL

  WRITE UNDERFLOW
  GOTO STEP 6

- **STEP 2:** SET PTR = HEAD
- **STEP 3:** SET HEAD = HEAD → NEXT
- **STEP 4:** SET HEAD → PREV = NULL
- **STEP 5:** FREE PTR
- **STEP 6:** EXIT



```
ptr = head
head = head -> next
head -> prev = NULL
free (ptr)
```

Deletion in doubly linked list from beginning

Deletion in doubly linked list at the end

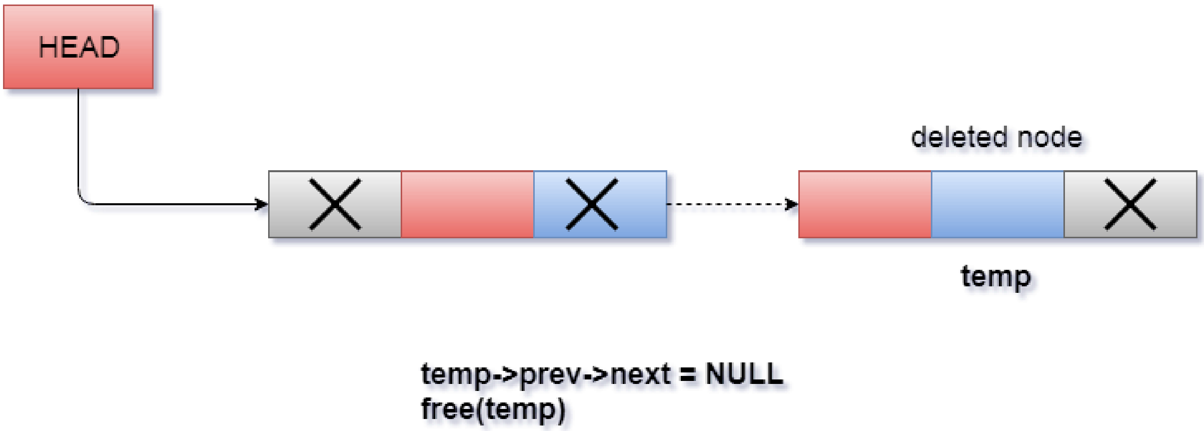
```
○ Step 1: IF HEAD = NULL

      Write UNDERFLOW
      Go to Step 7
      [END OF IF]

○ Step 2: SET TEMP = HEAD
○ Step 3: REPEAT STEP 4 WHILE TEMP->NEXT != NULL
○ Step 4: SET TEMP = TEMP->NEXT

      [END OF LOOP]

○ Step 5: SET TEMP ->PREV-> NEXT = NULL
○ Step 6: FREE TEMP
○ Step 7: EXIT
```



Deletion in doubly linked list at the end

Deletion in doubly linked list after the specified node

Algorithm

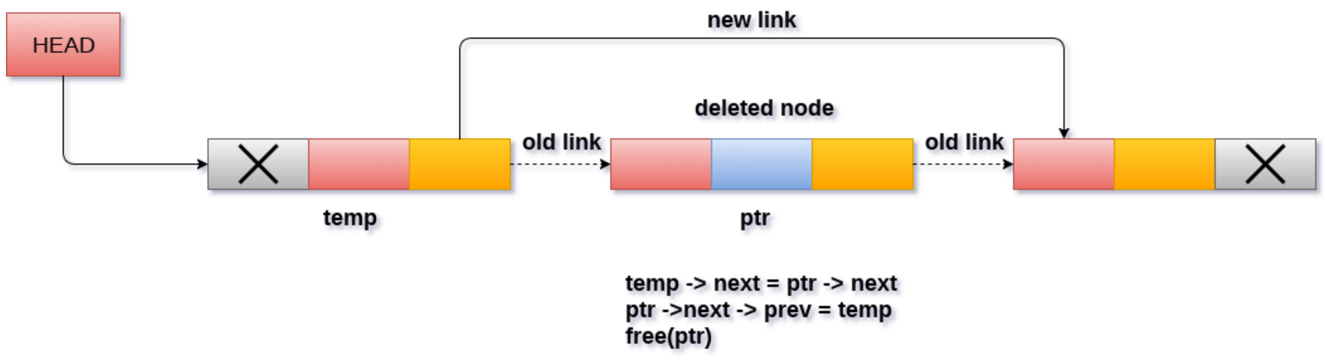
- **Step 1:** IF HEAD = NULL

  Write UNDERFLOW
  Go to Step 9
  [END OF IF]

- **Step 2:** SET TEMP = HEAD
- **Step 3:** Repeat Step 4 while TEMP -> DATA != ITEM
- **Step 4:** SET TEMP = TEMP -> NEXT

  [END OF LOOP]

- **Step 5:** SET PTR = TEMP -> NEXT
- **Step 6:** SET TEMP -> NEXT = PTR -> NEXT
- **Step 7:** SET PTR -> NEXT -> PREV = TEMP
- **Step 8:** FREE PTR
- **Step 9:** EXIT

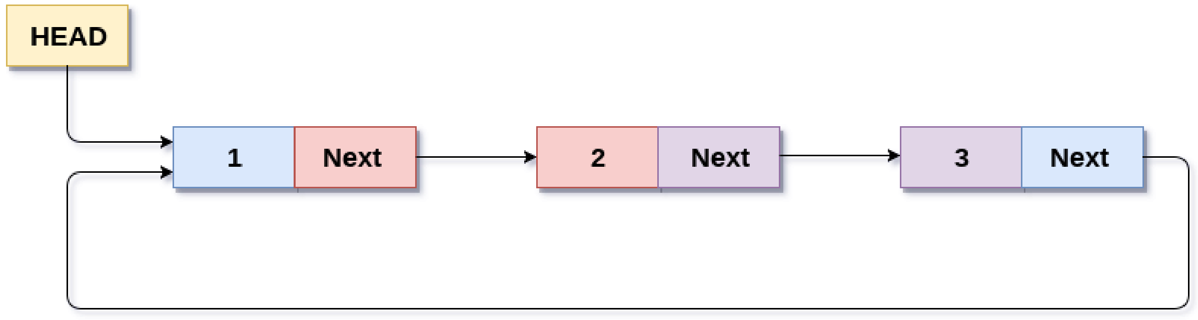


Deletion of a specified node in doubly linked list

Circular Singly Linked List

In a circular singly linked list, the last node of the list contains a pointer to the first node of the list. We can have circular singly linked list as well as circular doubly linked list.

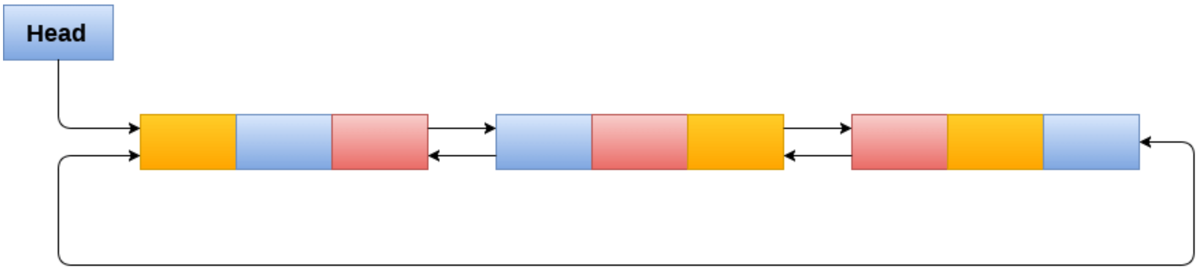
We traverse a circular singly linked list until we reach the same node where we started. The circular singly liked list has no beginning and no ending. There is no null value present in the next part of any of the nodes.



Circular Singly Linked List

Circular Doubly Linked List

Circular doubly linked list is a more complexed type of data structure in which a node contain pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the node. The last node of the list contains the address of the first node of the list. The first node of the list also contain address of the last node in its previous pointer.



Circular Doubly Linked List

Complexity(singly linked list)

Time Complexity								Space Compleity
Average				**Worst**				**Worst**
Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
θ(n)	θ(n)	θ(1)	θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)