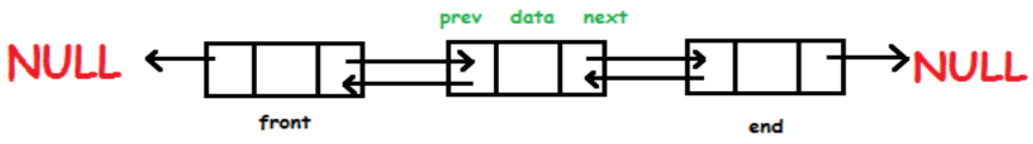
# Data Structures *For* BCA 2nd Semester Lecture 2 [Insertion and Deletion Algorithms for Doubly Linked List, Circular Linked List]	Compiled *By* Sakhi Bandyopadhyay Dept. of Computer Science & BCA, Kharagpur College, Kharagpur 721305

Doubly Linked List

Doubly linked list is a type of linked list in which each node apart from storing its data has two links. The first link points to the previous node in the list and the second link points to the next node in the list. The first node of the list has its previous link pointing to NULL similarly the last node of the list has its next node pointing to NULL.



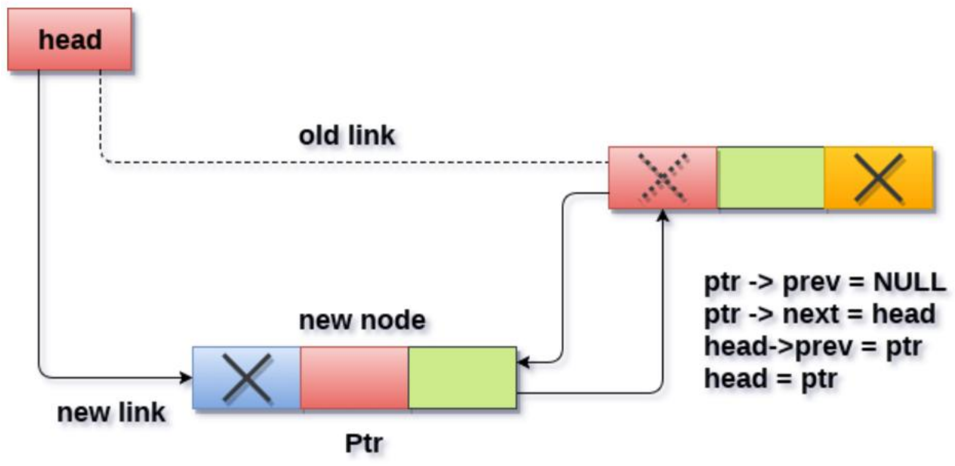
Define the node:

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

Insertion in doubly linked list at beginning:

Algorithm :

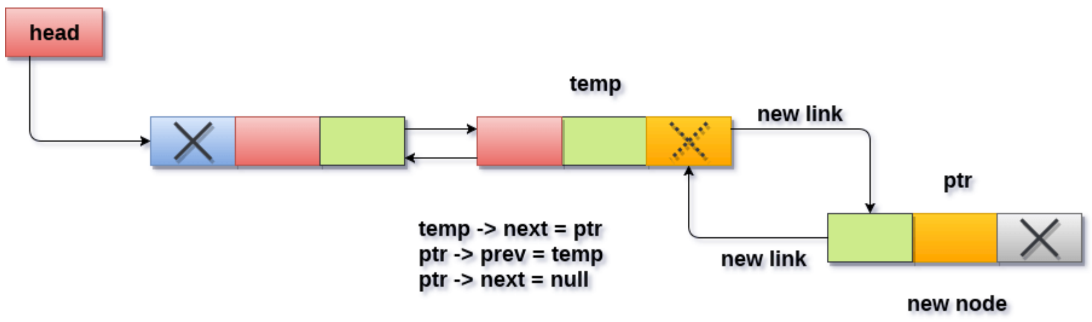
```
Step 1: IF ptr = NULL
            Write OVERFLOW
            Go to Step 9
            [END OF IF]
Step 2: SET NEW_NODE = ptr
Step 3: SET ptr = ptr -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET head -> PREV = NEW_NODE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



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

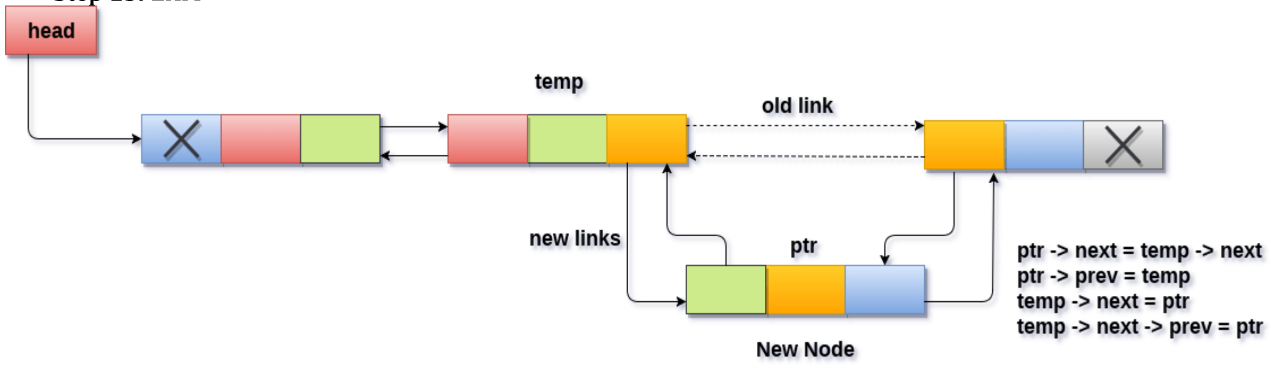


Insertion into doubly linked list at the end

Insertion in doubly linked list after Specified node:

Algorithm:

```
Step 1: IF PTR = NULL
            Write OVERFLOW
              Go to Step 15
             [END OF IF]
Step 2: SET NEW_NODE = PTR
Step 3: SET PTR = PTR -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET TEMP = START
Step 6: SET I = 0
Step 7: REPEAT 8 to 10 until I<="" li="">
Step 8: SET TEMP = TEMP -> NEXT
STEP 9: IF TEMP = NULL
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 = NEW_NODE
Step 14: SET TEMP -> NEXT -> PREV = NEW_NODE
Step 15: EXIT
```

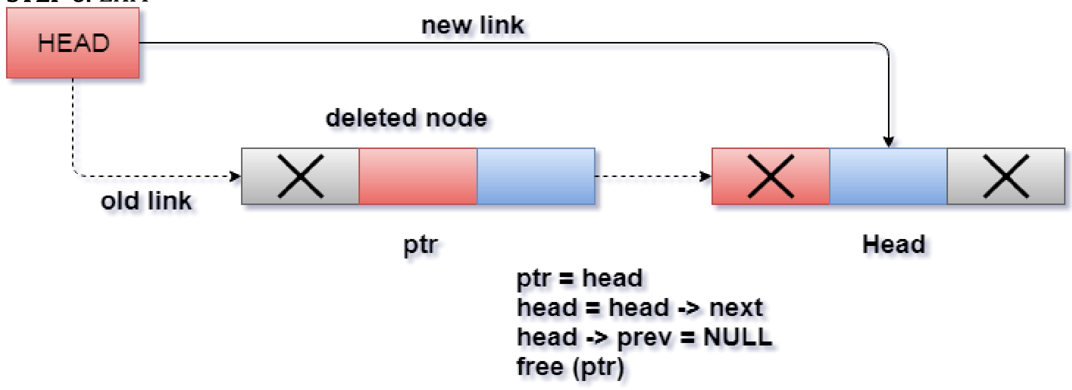


Insertion into doubly linked list after specified node

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
```

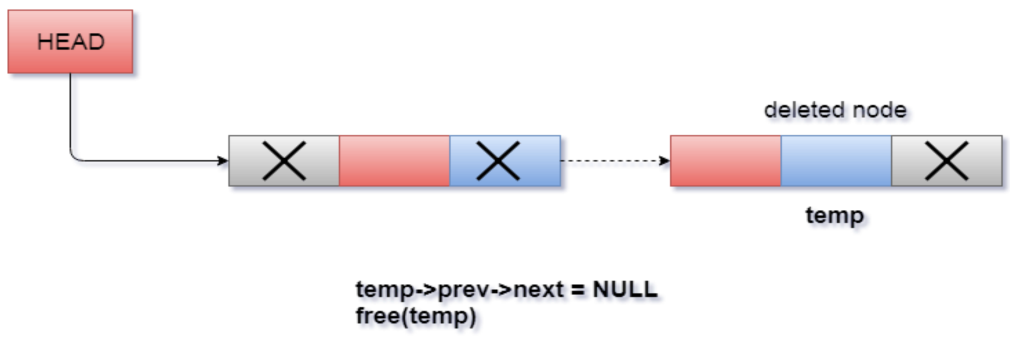


Deletion in doubly linked list from beginning

Deletion in doubly linked list at the end:

Algorithm:

```
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
```

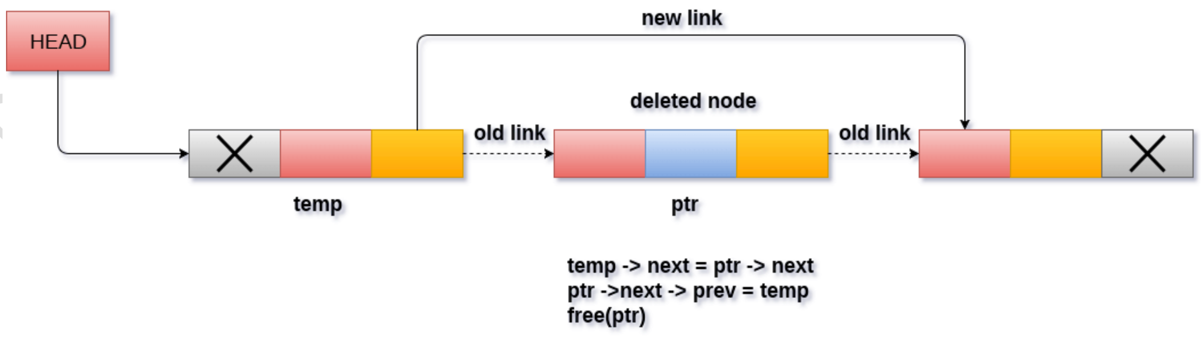


Deletion in doubly linked list at the end

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

Searching for a specific node in Doubly Linked List:

Algorithm:

```
Step 1: IF HEAD == NULL
            WRITE "UNDERFLOW"
            GOTO STEP 8
            [END OF IF]
Step 2: Set PTR = HEAD
Step 3: Set i = 0
Step 4: Repeat step 5 to 7 while PTR != NULL
Step 5: IF PTR → data = item
            return i
            [END OF IF]
Step 6: i = i + 1
Step 7: PTR = PTR → next
Step 8: Exit
```

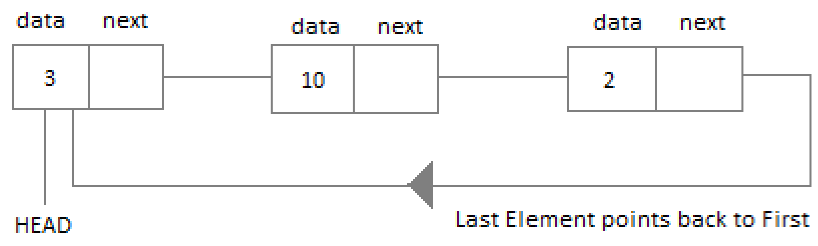
Traversing in doubly linked list:

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
```

Circular Linked List

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.



Application of Circular Linked List:

1) The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.

2) Another example can be Multiplayer games. All the Players are kept in a Circular Linked List and the pointer keeps on moving forward as a player's chance ends.

3) Circular Linked List can also be used to create Circular Queue. In a Queue we have to keep two pointers, FRONT and REAR in memory all the time, where as in Circular Linked List, only one pointer is required.

Advantages of Circular Linked Lists:

1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2) Useful for implementation of queue. We don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.

3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

4) Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.