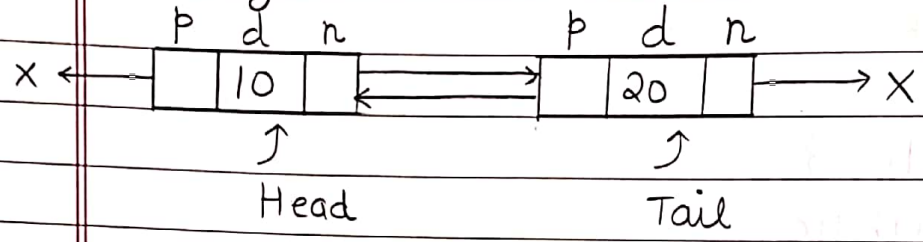
9/04/2023

Doubly linked list




```
class Node { public :
        int data ;
        Node * prev ;
        Node * next ;
};
```

In doubly linked list, reverse traversal is possible which is not the case in singly linked list.

Printing the doubly linked list
// same as that of singly linked list

```
void print (Node* & head) {

    Node * temp = head ;
    while (temp != NULL) {
        cout << temp -> data << " " ;
        temp = temp -> next ;
    }
}
```

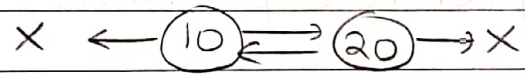
Length of doubly linked list
// Same as that of singly linked list

```
int length DLL (Node * & head) {
        Node * temp = head ;
```

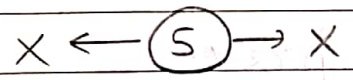
```
    int len = 1;
    while (temp -> next != NULL) {
        len++;
        temp = temp -> next;
    }
    return len;
}
```

Time complexity = O(n) } For both print and
Space complexity = O(1) } lengthDLL function

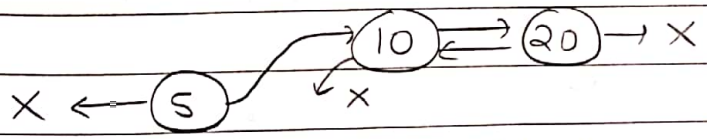
Insertion at head



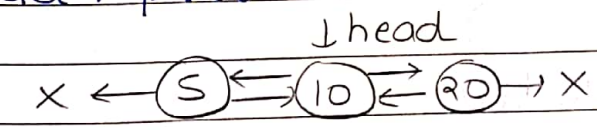
1) Create a newNode with data say = 5



2) newNode -> next = head



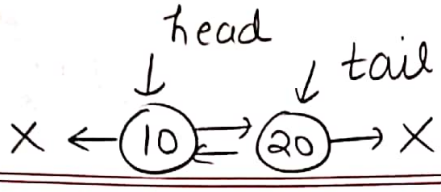
3) head -> prev = newNode;



4) Update head as newNode.

Just we need to handle the empty linked list as we did in the singly linked list case.

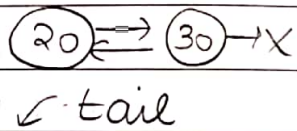
Insertion at Tail



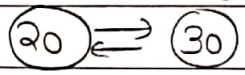
1) Create a node as newNode

2) tail → next = newNode;

3) newNode → prev = tail;

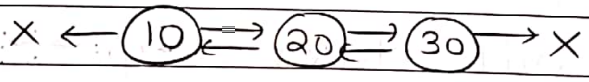


4) tail = newNode;

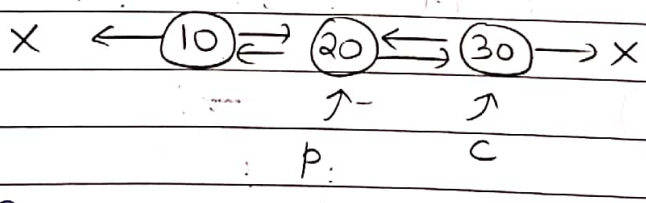


Time complexity = O(1) → Tail pointer maintained
Space complexity = O(1) else O(n).

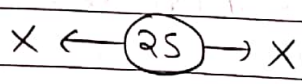
Insertion at specified position



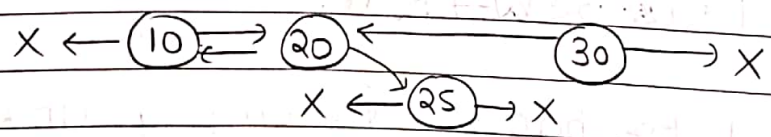
1) Find current and prev.



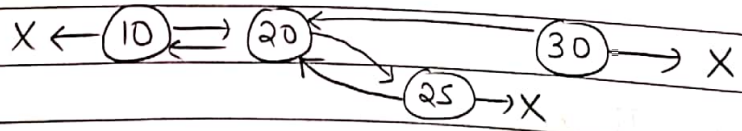
2) Create a node



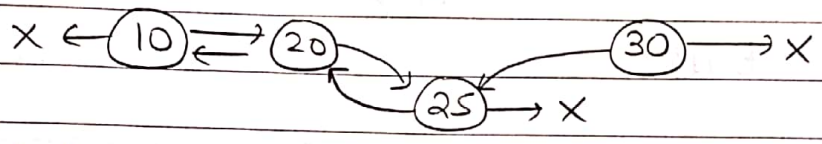
3) p → next = newNode;



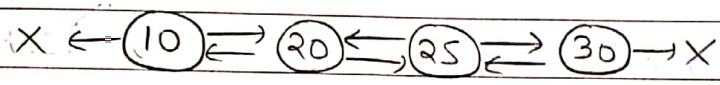
4) newNode → prev = p;



5) c → prev = newNode;



6) newNode → next = c;

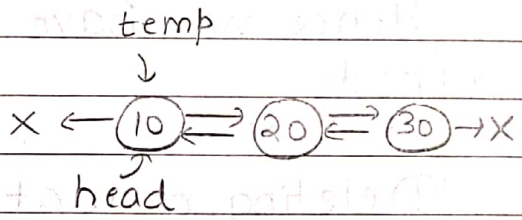


Hence we have inserted the node.

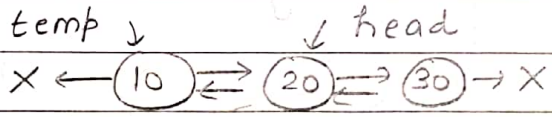
Deletion of head

head

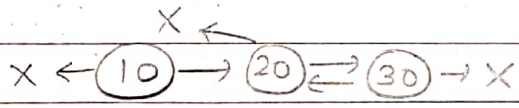
1) temp = head



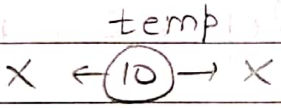
2) head = head → next



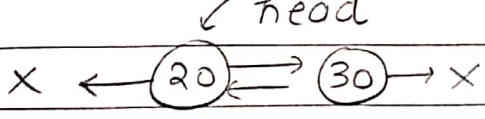
3) head → prev = NULL;



4) temp → next = NULL



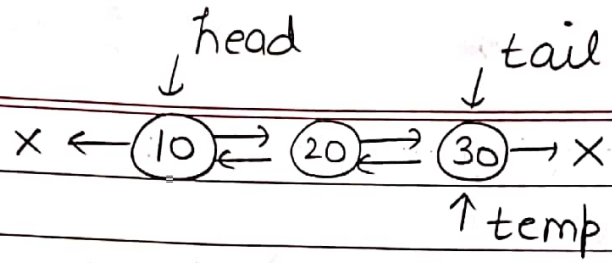
5) delete temp;



Hence deleted the head & also updated head.

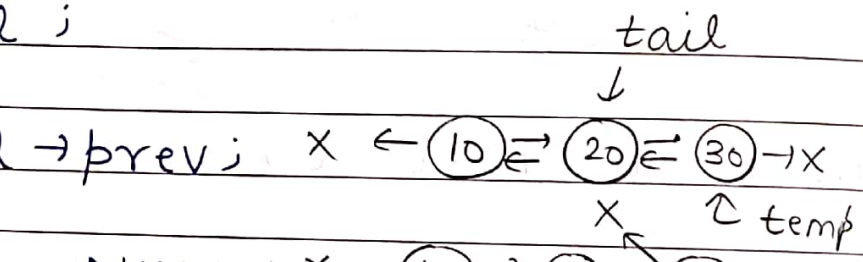
Handle empty linked list case & single node case.

Deletion of tail

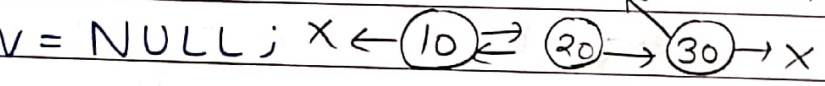


1) temp = tail ;

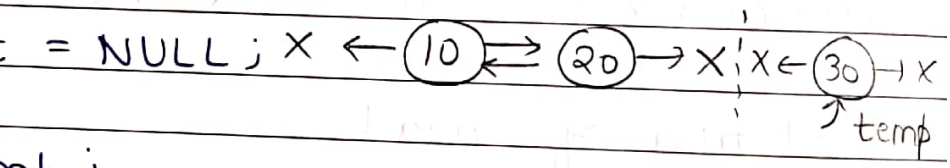
2) tail = tail → prev ;



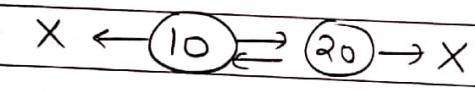
3) temp → prev = NULL ;



4) tail → next = NULL ;

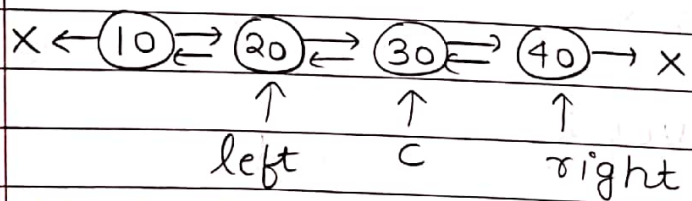


5) delete temp ;



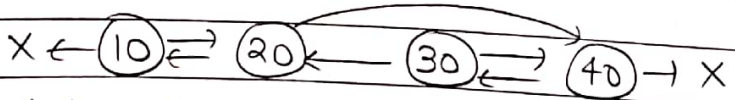
Hence we have successfully deleted the node.

Deleting node at specific position



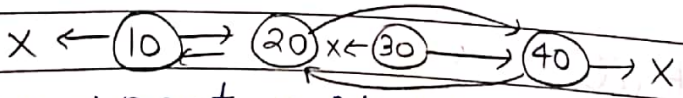
1) Find left, c, and right.

2) left → next = right ;

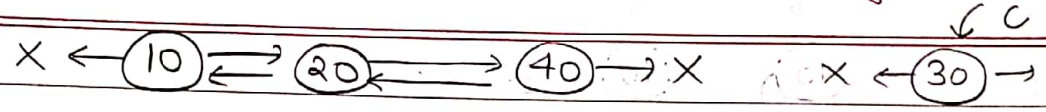


3) right → prev = left ;

4) curr → prev = NULL ;



5) curr → next = NULL ;



6) delete curr;
↳(c)

Hence we have successfully deleted the node at specified position.

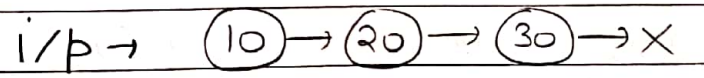
Circular linked list

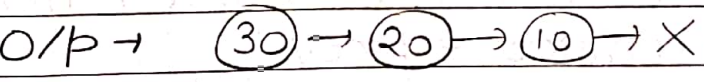
(i) Singly ⇒ tail → next = head;
(ii) Doubly ⇒ head → prev = tail;
tail → next = head

However there is no head & tail in the circular linked list.

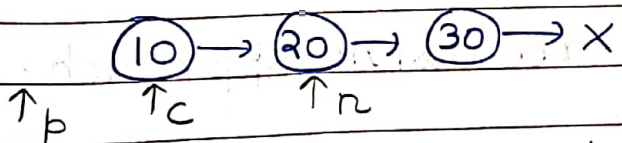
Questions

Q1 Reverse a linked list

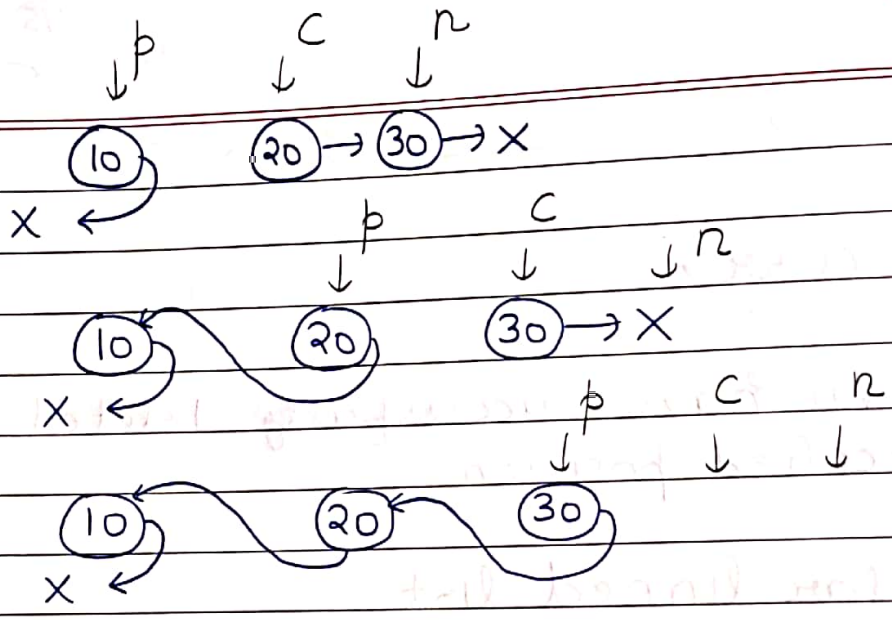




We need to make sure every node points to the backward node.



Do c → next = p and then recursion will handle, but before breaking the link we need to maintain next pointer.



Here c becomes NULL & hence we found the base case also.

Code

* Recursion

```
Node * reverse (Node* & p , Node* &c){

    // Base case
    if ( c == NULL) {
        return p; // new Head
    }
    // I case solve                    → (c)
    Node * forward = curr → next;
(c) ← curr → next = prev;
    // recursive call                  ↳ (p)
    return reverse (curr, forward);
}                                       ↳ (c)
```

The above is the recursive approach.

* Iterative

```
Node* reverse (Node * & prev, Node * &curr
```

```
{ while (curr != NULL) {
        //Same steps as in recursion
        Node * forward = curr -> next ;
        curr -> next = prev ;
        prev = curr ;
        curr = forward ;
}
return prev ; // New Head
}
```