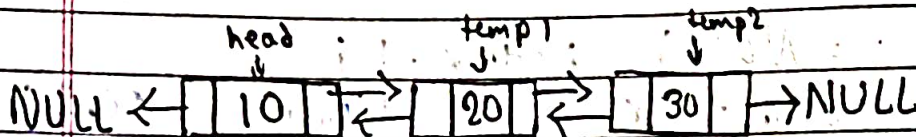


(*)

DOUBLY LINKED LIST



In Doubly Linked list we have an extra pointer with every node name as previous

```
struct Node {
```

```
    int data;
```

```
    Node *prev;
```

```
    Node *next;
```

```
    Node (int d)
```

```
    { data = d;
```

```
      prev = NULL;
```

```
      next = NULL; }
```

```
};
```

```
int main () {
```

```
    Node *head = new Node (10);
```

```
    Node *temp1 = new Node (20); // head->next
```

```
    Node *temp2 = new Node (30); // head->next->next
```

```
    head->next = temp1;
```

```
    temp1->prev = head;
```

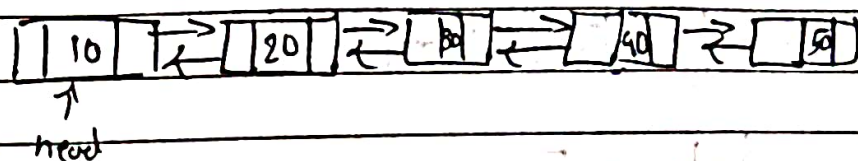
```
    temp1->next = temp2;
```

```
    temp2->prev = temp1;
```

```
    // temp2->next = NULL; if already NULL
```

```
}
```

* Singly vs Doubly linked list:-



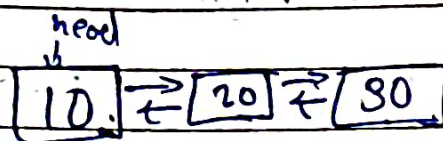
Advantages:-

- 1) Can be traversed in both direction.
- 2) A given delete a node in $O(1)$ time with given reference/pointer to it.
- 3) Insert/Delete before a given node.
- 4) Insert/Delete from both end in $O(1)$ time by maintaining tail.

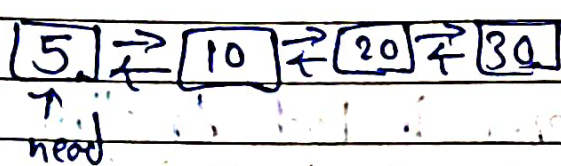
Disadvantages:-

- 1) Extra space for prev.
- 2) Code becomes more complex.

* Insert at Beginning of DLL

1) ILP:- 

data = 5

O/P:- 

```

=> struct Node {
    int data;
    Node *next;
    Node *prev;
    Node (int x) { data = x;
                  prev = NULL;
                  next = NULL; }
};

```

```

Node *insertBegin (Node *head, int data)
{
    Node *temp = new Node (data);
    temp->next = head;
    if (head != NULL) { head->prev = temp; }
    return temp;
}

```

* Insert at End of D.L.L.

1) I/P:- $\boxed{10} \rightarrow \boxed{20} \rightarrow \boxed{30}$, data = 40
 \uparrow
 head

O/P:- $\boxed{10} \rightarrow \boxed{20} \rightarrow \boxed{30} \rightarrow \boxed{40}$,

2) I/P:- NULL. data = 40

O/P:- $\boxed{40}$
 \uparrow
 head.

only case where
head gets change

⇒

```
Node *insertEnd (Node *head, int data) {
```

```
    Node *temp = new Node(data);
```

```
    if (head == NULL)
```

```
    { return temp; }
```

```
    Node *curr = head;
```

```
    while (curr->next != NULL)
```

```
    { curr = curr->next; }
```

```
    curr->next = temp;
```

```
    temp->prev = curr;
```

```
    return head;
```

```
}
```


* Reverse a Doubly linked list:-

1) I/P:- $\boxed{10} \rightleftarrows \boxed{20} \rightleftarrows \boxed{30} \rightleftarrows \boxed{40}$

O/P:- $\boxed{40} \rightleftarrows \boxed{30} \rightleftarrows \boxed{20} \rightleftarrows \boxed{10}$

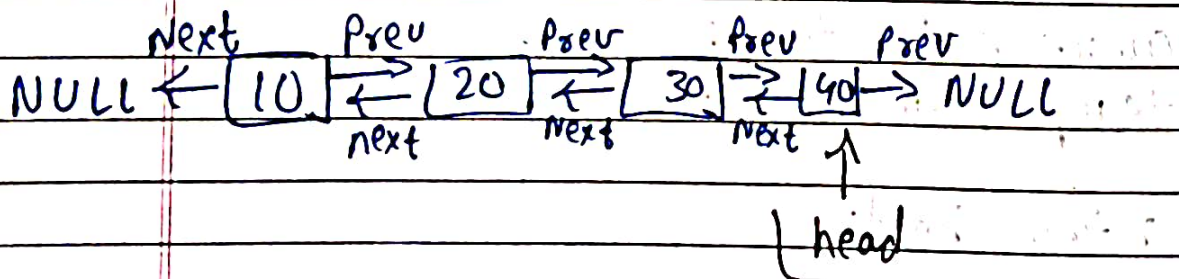
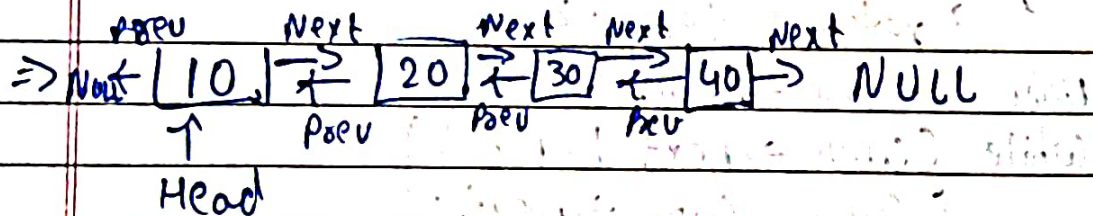
2) I/P:- $\boxed{10}$

O/P:- $\boxed{10}$

3) I/P:- NULL

O/P:- NULL

We just have to swap the next & prev of all the element



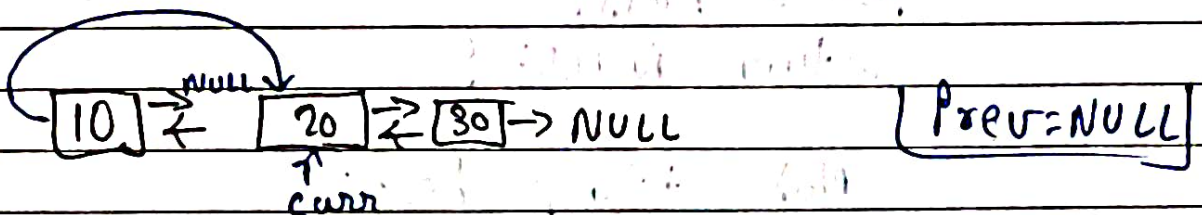
⇒

```
Node *reverseDLL (Node *head) {
    if (head == NULL || head->next == NULL)
        return head;
    Node *prev = NULL; curr Node *curr = head;
    while (curr != NULL)
    {
        prev = curr->prev;
        curr->prev = curr->next;
        curr->next = prev;
        curr = curr->prev;
    }
    return prev->prev;
}
```

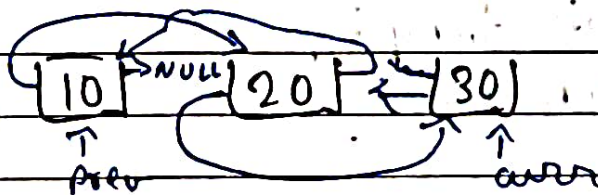
Swapping

NULL ← [10] → [20] → [30] → NULL

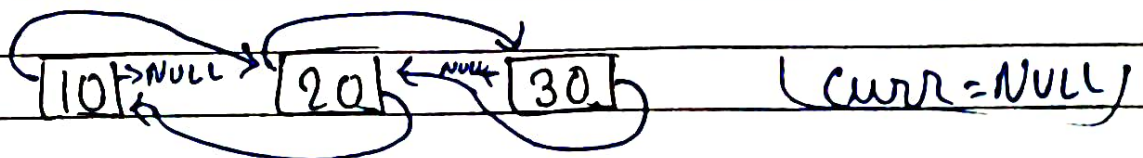
- After 1st Iteration:-



- After 2nd Iteration:-



- After 3rd Iteration:-



* Delete Head of a Doubly Linked List:-

1) I/P:- [10] \leftrightarrow [20] \leftrightarrow [30]

O/P:- [20] \leftrightarrow [30]

2) I/P:- [10]

O/P:- NULL

3) I/P:- NULL

O/P:- NULL

Time:- $O(1)$

\Rightarrow

```
Node *delHead(Node *head) {
    if (head == NULL) {return NULL;}
```

```
    if (head->Next == NULL) {
        delete head;
        return NULL; }
    else {
```

```
        Node *temp = head;
        head = head->next;
        head->prev = NULL;
        delete temp;
        return head; }
```

```
    }
```

* Delete Last Node of Doubly Linked List :-

1) I/P:- $10 \leftrightarrow 20 \leftrightarrow 30$: :

O/P:- $10 \leftrightarrow 20$ ~~$\leftrightarrow 30$~~

2) I/P:- 10 :

O/P:- NULL

Time:- $O(n)$

3) I/P:- NULL

O/P:- NULL

⇒

```
Node *delLast(Node *head) {
    if (head == NULL) { return NULL; }
    if (head->next == NULL) {
        delete head;
```

```
        return NULL; }
    Node *curr = head;
```

```
    while (curr->next != NULL)
        { curr = curr->next; }
```

```
    curr->prev->next = NULL;
```

```
    delete curr;
```

```
    return head;
```

```
}
```

If we maintain a tail pointer then the loop is not required.