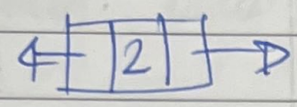
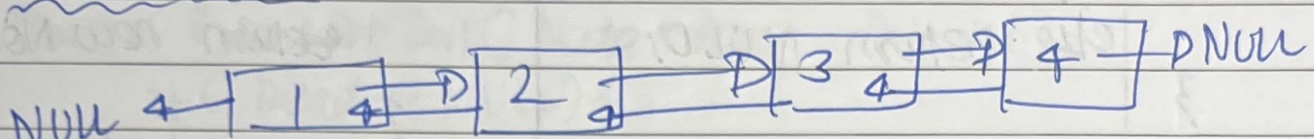
21 | December | 23
__/__/____

{ Doubly Linked List }

- contain both pointer back as well as front



- Representation



- { Structure for doubly Linked List }

```
class Node{
    Public:
    int data;
    Node* back;
    Node* next;
    Public:
    Node(int data1, Node* back1, Node* next1){
        data = data1;
        back = back1;
        next = next1;
    }
    Public:
    Node(int data1){
        data = data1;
        back = NULL;
        next = NULL;
    }
};
```

__/__/____

{ conversion Array to doubly linked list }

```
pair<Node*, Node*> convert Array 2 DLL (vector<int>vec)
{
    Node* head = new Node(vec[0]);
    Node* tail = NULL;
    Node* mover = head;
    for (int i=1; i< vec.size(); i++) {
        Node* temp = new Node(vec[i], mover, NULL);
        mover->next = temp;
        mover = temp;
        tail = mover;
    }
    return { head, tail }; // returning head as
                           well as tail
}
```

{ deletion of the Head }

```
Node* Deletion of Head ( Node* head){
    if (head == NULL && head->next == NULL){
        return NULL;
    }
    Node* temp = head;
    head = head->next;
    head->back = NULL;
    temp->next = NULL;
    delete temp;
    return Head;
}
```

__/__/____

c { Deletion of Tail }

```
Node* DeletionOfTail(Node* head){
    if(head==NULL or head->next==NULL){
        return NULL;
    }
    Node* temp = head;
    while( temp->next->next != NULL){
        temp = temp->next;
    }
    Node* tail = temp->next;
    temp->next = NULL;
    tail->back = NULL;
    delete tail;
    return head;
}
```

d { Delete the kth element of DLL }

```
Node* DeletionAtKth(Node* head, int k){
    if(head == NULL){ return NULL; }
    Node* temp = head;
    int cnt=0;
    while(temp != NULL){
        cnt++;
        if(cnt==k) break;
        temp = temp->next;
    }
```

__/__/____

```
Node* prev = temp -> back;      // only one node
Node* front = temp -> next;      ↑
if( prev == front and prev == NULL){
        delete temp;
        return NULL;
    }
// Head Node
else if( prev == NULL){
        head = head => next;
        head -> back = NULL;
        temp -> next = NULL;
        delete temp;
        return head;
}          ← // call Node
else if( front == NULL){
        prev -> next = NULL;
        temp -> back = NULL;
        delete temp;
        return head;
}
else {
        // middle element
          prev -> next = front;
          front -> back = prev;
          temp -> back = NULL;
          temp -> next = NULL;
           delete temp;
           return head;
}
        return head; }
```

__/__/____

ϕ { Delete a Particular Node }
edgecase → { temp must not be head node }

```
void deleteNode (Node* temp){
    Node* prev = temp->back;
    Node* front = temp->next;
        // Tail Node
    if (front == NULL){
            prev->next = NULL;
            temp->back = NULL;
            free(temp);
            return;
        }
        prev->next = front
        front->back = prev
        temp->next = prev->back = NULL;
        free(temp); temp
}
```

ϕ { Insert before Head }

```
Node* insertBeforeHead (Node* head, int data){
        Node* newNode = new Node(data, NULL, head);
        head->back = newNode;
        return newNode;
    }
```

__/__/____

{ Insert Before Tail }

```
Node* insertBeforeTail(Node* head, int data){
    if(head->next==NULL){
        Node* newNode = new Node(data);
        head->next = newNode;
        newNode->back = head;
        return head;
    }
    Node* temp = head;
    while(temp->next != NULL){
        temp = temp->next;
    }
    Node* newNode = newNode(data, temp->back,
                            temp);
    temp->back->next = newNode;
    temp->back = newNode;
    return head;
}
```

__/__/____

① {Reverse DLL}

Input → {1 → 2 → 3 → 4 → 5}

Output → {5 → 4 → 3 → 2 → 1}

❌ Brute :- TC → O(2N)

SC → O(N).

```
Node* CallBrute(Node* head){
    stack<int> st;
    Node* temp = head;
    while(temp != NULL){
        st.push(temp->data);
        temp = temp->next;
    }
    temp = head;
    while(temp != NULL){
        temp->data = st.top();
        st.pop();
        temp = temp->next;
    }
return head;
}
```

✅ Optimal :-

```
Node* callOptimal(Node *head){
    Node* temp = head;
    while(temp != NULL){
        Node* last = temp->back;
        temp->back = temp->next;
        temp->next = last;
        head = temp;
        temp = temp->back;
    }
return head;
}
```