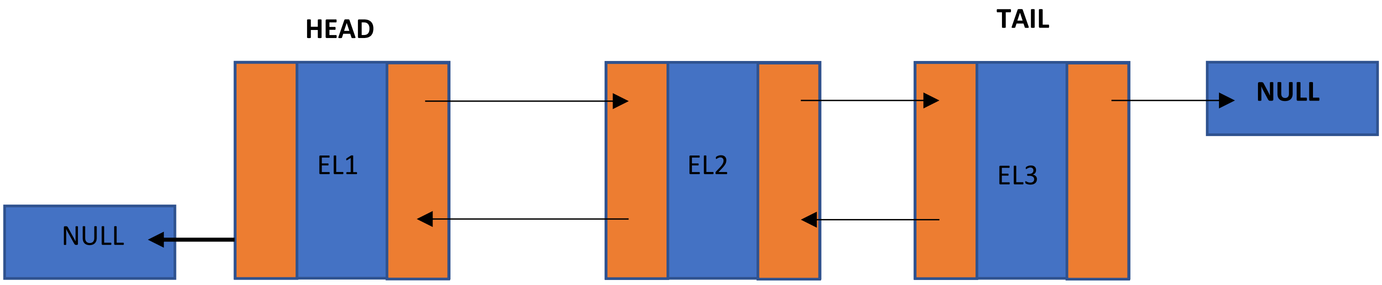
Double Linked List

- each item in the list has next and previous fields

- the list has a head and a tail

- in double linked list we can add new element to end or start with the constant time complexity

- to find specified element we will have to traverse all the list from head to tail or reverse to find wanted item



Insert at head

1) Create new node “newElement”
2) Assing “El1” to “newElement” next field
3) Assign whatever “El1” is pointing to as previous to “newElement” previous field
4) Assign “newElement” to “El1” previous field
5) Assign head to “newElement”
6) O(1) time complexity

Insert at tail

1) Create new node “newElement”
2) Assing tail’s next field to “newElement” next field
3) Assign tail to “newElement” previous field
4) Assign tail’s next field to “newElement”
5) Assign tail to “newElement:
6) O(1) time complexity

Delete from head

1) Assign “El1” to “removedNode”
2) Assign “El2” previous field to “El1” previous field
3) Assign head to “El1” next field
4) Return “removedNode” from the method
5) O(1) time complexity

Delete from tail

1) Assign “El3” to “removedNode”
2) Assign “El2” next field to “El3” next field
3) Assign tail to “El3” previous field
4) Return “removedNode” from the method
5) O(1) time complexity

Insert a node A between nodes B and C

1) Assign A’s next field to B’s next field
2) Assign A’s previous field to C’s previous field
3) Assign B’s next field to A
4) Assign C’s previous field to A
5) O(n) time complexity – because we need to find the insertion position

Remove a node A from between nodes B and C

1) Assign A to “removedNode”
2) Assign C’s previous field to A’s previous field
3) Assign B’s next field to A’s next field
4) Return A from the method
5) O(n) time complexity – because we need to find the insertion position