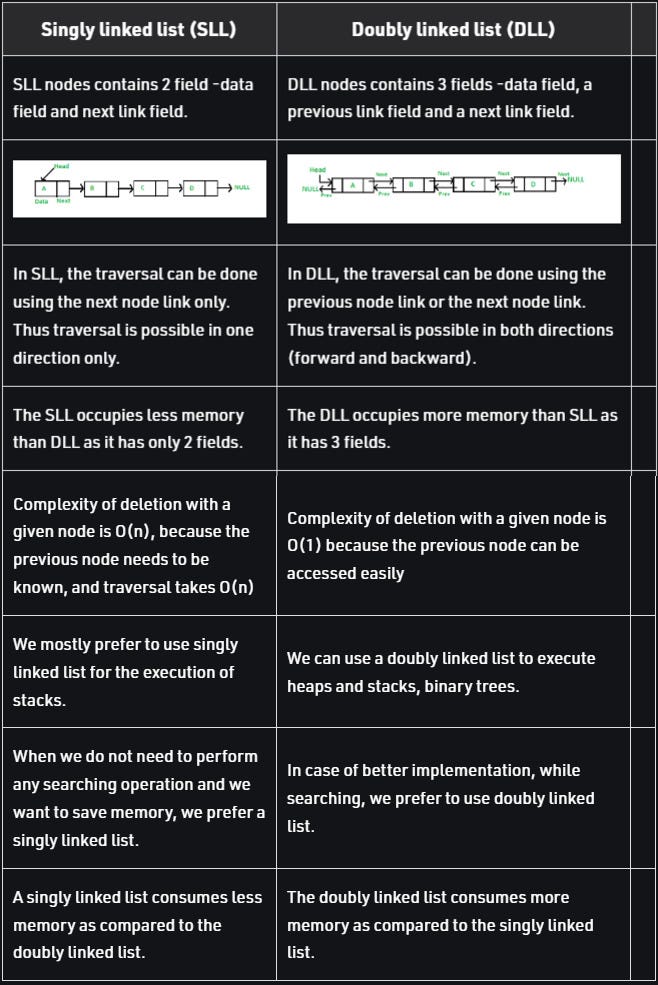
**Doubly Linked List**

The main characteristics of doubly linked list is, in here we need to store previous node reference as well as the current value and next node reference.

# ****Difference between singly linked list and doubly linked list****



**Implementation**

Similar to implementation of Singly Linked List, we will also follow the object-oriented way in here. So that first we need a node class and a linked list class.

**Class Node**

In ***Node class,*** with ***value*** property and ***next*** node reference we need to save ***prev***(previous) node reference.

class Node {  
 constructor(value, next = null, prev = null) {  
 this.value = value;   
 this.next = next;   
 this.prev = prev;   
 }  
}

**Class DoublyLinkedList**

class DoublyLinkedList {  
 constructor(value) {   
 let newNode = new Node(value);   
 this.head = newNode;  
 this.tail = newNode   
 this.length = 1;   
 }   
}

**Node append method**

append(value) {  
 let newNode = new Node(value);   
 this.tail.next = newNode;   
 newNode.prev = this.tail;   
 this.tail = newNode;   
  
 this.length++;   
}

**Node prepend method**

prepend(value) {  
 let newNode = new Node(value);  
 newNode.next = this.head;  
 this.head.prev = newNode;  
 this.head = newNode;  
 this.length++;  
}

**Insert a node at specific position**

insertAtPosition(value, n) {   
 if(n === 1) {  
 this.prepend(value);  
 return;  
 }   
   
 if(n > this.length) {  
 this.append(value)  
 return  
 }  
   
 let newNode = new Node(value);   
 let prevNode = this.find(n-1)  
 let nextNode = prevNode.next  
 if(!prevNode) return;  
   
 prevNode.next = newNode  
 newNode.prev = prevNode  
 newNode.next = nextNode  
 nextNode.prev = newNode  
 this.length++;  
}  
  
find(n) {  
 let data = this.head  
 let position = 1  
  
 while(data) {  
 if (n === position) {  
 return data  
 }  
  
 position++  
 data = data.next  
 }  
}

**Head delete method**

deleteHead() {  
 if (this.length === 1) {  
 this.head = null  
 this.tail = null  
 this.prev = null  
 this.length--  
 return  
 }  
  
 let newHead = this.head.next  
 newHead.prev = null  
 this.head = newHead  
 this.length--  
}

**Tail delete method**

deleteTail() {  
 if (this.length === 1) {  
 this.head = null  
 this.tail = null  
 this.prev = null  
 this.length--  
 return  
 }  
  
 let newTail = this.tail.prev  
 newTail.next = null  
 this.tail = newTail  
 this.length--  
}

**Delete at specific position method**

delete(n) {  
 if (this.length === 1) {  
 this.head = null  
 this.tail = null  
 this.prev = null  
 this.length--  
 return  
 }  
  
 if(n === 1) {  
 this.deleteHead();  
 return;  
 }   
   
 if(n > this.length) {  
 this.deleteTail()  
 return  
 }  
   
 let prevNode = this.find(n-1)  
 let nextNode = prevNode.next.next  
   
 prevNode.next = nextNode  
 nextNode.prev = prevNode  
   
 this.length--  
}

**Reverse a doubly linked list**

reverse() {  
 let currNode = this.head  
 let prevNode = null  
 let nextNode =null  
   
 while(currNode) {  
 prevNode = currNode.prev  
 nextNode = currNode.next  
   
 currNode.next = prevNode  
 currNode.prev = nextNode  
   
 currNode = nextNode  
 }  
   
 this.tail = this.head  
 this.head = prevNode  
}