```
function Node(data) {
  this.data = data;
  this.previous = null;
  this.next = null;
}

class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
    this.numberOfValues = 0;
  }

  add(data) {
    const node = new Node(data);
    if(!this.head) {
      this.head = node;
      this.tail = node;
    } else {
      node.previous = this.tail;
      this.tail.next = node;
      this.tail = node;
    }
    this.numberOfValues++;
  }

  remove(data) {
    let current = this.head;
    while(current) {
      if(current.data === data) {
        if(current === this.head && current === this.tail) {
          this.head = null;
          this.tail = null;
        } else if(current === this.head) {
          this.head = this.head.next;
          this.head.previous = null;
        } else if(current === this.tail) {
          this.tail = this.tail.previous;
          this.tail.next = null;
        } else {
          current.previous.next = current.next;
          current.next.previous = current.previous;
        }
        this.numberOfValues--;
      }
      current = current.next;
    }
  }

  insertAfter(data, toNodeData) {
    let current = this.head;
    while(current) {
      if(current.data === toNodeData) {
        const node = new Node(data);
        if(current === this.tail) {
          this.add(data);
        } else {
          current.next.previous = node;
          node.previous = current;
          node.next = current.next;
```

```
          current.next = node;
          this.numberOfValues++;
        }
      }
      current = current.next;
    }
  }

  traverse(fn) {
    let current = this.head;
    while(current) {
      if(fn) {
        fn(current);
      }
      current = current.next;
    }
  }

  traverseReverse(fn) {
    let current = this.tail;
    while(current) {
      if(fn) {
        fn(current);
      }
      current = current.previous;
    }
  }

  length() {
    return this.numberOfValues;
  }

  print() {
    let string = '';
    let current = this.head;
    while(current) {
      string += `${current.data} `;
      current = current.next;
    }
    console.log(string.trim());
  }
}

const doublyLinkedList = new DoublyLinkedList();
doublyLinkedList.print(); // => ''
doublyLinkedList.add(1);
doublyLinkedList.add(2);
doublyLinkedList.add(3);
doublyLinkedList.add(4);
doublyLinkedList.print(); // => 1 2 3 4
console.log('length is 4:', doublyLinkedList.length()); // => 4
doublyLinkedList.remove(3); // remove value
doublyLinkedList.print(); // => 1 2 4
doublyLinkedList.remove(9); // remove non existing value
doublyLinkedList.print(); // => 1 2 4
doublyLinkedList.remove(1); // remove head
doublyLinkedList.print(); // => 2 4
doublyLinkedList.remove(4); // remove tail
doublyLinkedList.print(); // => 2
console.log('length is 1:', doublyLinkedList.length()); // => 1
```

```
121 doublyLinkedList.remove(2); // remove tail, the list should be empty
122 doublyLinkedList.print(); // => ''
123 console.log('length is 0:', doublyLinkedList.length()); // => 0
124 doublyLinkedList.add(2);
125 doublyLinkedList.add(6);
126 doublyLinkedList.print(); // => 2 6
127 doublyLinkedList.insertAfter(3, 2);
128 doublyLinkedList.print(); // => 2 3 6
129 doublyLinkedList.traverseReverse(node => { console.log(node.data); });
130 doublyLinkedList.insertAfter(4, 3);
131 doublyLinkedList.print(); // => 2 3 4 6
132 doublyLinkedList.insertAfter(5, 9); // insertAfter a non existing node
133 doublyLinkedList.print(); // => 2 3 4 6
134 doublyLinkedList.insertAfter(5, 4);
135 doublyLinkedList.insertAfter(7, 6); // insertAfter the tail
136 doublyLinkedList.print(); // => 2 3 4 5 6 7
137 doublyLinkedList.add(8); // add node with normal method
138 doublyLinkedList.print(); // => 2 3 4 5 6 7 8
139 console.log('length is 7:', doublyLinkedList.length()); // => 7
140 doublyLinkedList.traverse(node => { node.data = node.data + 10; });
141 doublyLinkedList.print(); // => 12 13 14 15 16 17 18
142 doublyLinkedList.traverse(node => { console.log(node.data); }); // => 12 13 14 15 16
    17 18
143 console.log('length is 7:', doublyLinkedList.length()); // => 7
144 doublyLinkedList.traverseReverse(node => { console.log(node.data); }); // => 18 17 16
    15 14 13 12
145 doublyLinkedList.print(); // => 12 13 14 15 16 17 18
146 console.log('length is 7:', doublyLinkedList.length()); // => 7
147
```