Assignment 2

A. Implement in JavaScript the function findMiddle(L) than we did in class. The DLinkedList class is provided in the DLinkedList.js file.

```
function findMiddle(list) {
    let p = list.first();
    let q = list.last();

    if (list.isEmpty()) {
    return "No element in the list."
    } else {
        do {
            p = list.after(p);
            q = list.before(q)
        } while (p != q && list.after(p) != q)
    }
    return q.element()
}
let objList = new DLinkedList();
objList.insertFirst(1);
objList.insertAfter(objList.first(), 2);
objList.insertLast(5);
objList.insertBefore(objList.last(), 4);
objList.insertBefore(objList.before(objList.last()),3)
objList.print();

findMiddle(objList)

console.log(findMiddle(objList))
```

B. Describe, in pseudo-code, how to implement the stack ADT using a DLinkedList. What is the running time of the push() and pop() methods in this case? Implement a new Stack class in JavaScript based on (using) the DLinkedList class like done in A above.

```
class Stack {
    constructor() {
        this._stk = new DLinkedList();
    }
    isEmpty() {
        return this._stk.isEmpty();
    }
    push(element) {
        this._stk.insertFirst(element);
    }
    pop() {
        return this._stk.remove(this._stk.first());
    }
    top() {
        return this._stk.first().element();
    }
    size() {
        return this._stk.size();
    }
    remove() {
        this._stk.remove(first());
    }
    print() {
        this._stk.print();
    }
}
```

```
let objStack = new Stack()
objStack.push(1); // the running time is O(1)
objStack.push(3); // the running time is O(1)
objStack.push(2); // the running time is O(1)
objStack.push(5); // the running time is O(1)
objStack.push(4); // the running time is O(1)
objStack.print();
console.log(objStack.size());

objStack.pop(); // the running time is O(1)
objStack.pop(); // the running time is O(1)

objStack.print();
console.log(objStack.size());
```

Algorithm **stack**

O(1)-------stk← new DlinkedList( )

O(1)---------push(element)

O(1)---------pop(element)

The running time is **O(1)**

C. Describe, in pseudo-code, how to implement the queue ADT using a DLinkedList. What is the running time of the enqueue() and dequeue() methods in this case? Implement a new Queue class in JavaScript based on the DLinkedList class.

```
class Queue {
    constructor() {
        this._que = new DLinkedList();
    }
    isEmpty() {
        return this._que.isEmpty();
    }
    enqueue(element) {
        this._que.insertLast(element);
    }
    dequeue() {
        return this._que.remove(this._que.first());
    }
    front() {
        return this._que.first().element();
    }
    size() {
        return this._que.size();
    }
    print() {
        this._que.print();
    }
}
let objQueue = new Queue();
objQueue.enqueue(1);// the running time is O(1)
objQueue.enqueue(2); // the running time is O(1)
objQueue.enqueue(3); // the running time is O(1)
objQueue.enqueue(4); // the running time is O(1)
objQueue.enqueue(5); // the running time is O(1)
objQueue.print();
objQueue.dequeue(); // the running time is O(1)
objQueue.dequeue(); // the running time is O(1)
objQueue.print();
```

Algorithm **Queue**

O(1)-----objQueue← new Queue

O(1)--objQueue.enqueue(element)

O(1)----- objQueue.dequeue()

The running time is **O(1)**

C-2.2 Describe, in pseudo-code, how to implement the queue ADT using two stacks. What is the running time of the enqueue() and dequeue() methods in this case?

```
class DoubleQueue {
    constructor() {
        this._queIn=new Stack();
        this._queOut=new Stack();
    }
    enqueue(e){
        this._queIn.push(e);
    }
    dequeue(){
        while(this._queIn.size()!=0){
            this._queOut.push(this._queIn.pop());
        }
        return this._queOut.pop();
    }
    front(){
        return this._queOut.top()
    }
    print(){
        this._queIn.print();
    }
}

let objQue= new DoubleQueue();
objQue.enqueue(1);
objQue.enqueue(2);
objQue.enqueue(3);
objQue.enqueue(4);
objQue.enqueue(5);
objQue.print();

objQue.dequeue();
objQue.print();
```

Algorithm DoubleQueue()

O(1)------objQueue← new DoubleQueue()

O(1)------objQueue.enqueue(element)

O(1)----- objQueue.dequeue()

The running time is **O(1)**