```
Script started on 2021-03-04 19:19:42-0600
l_ladios@ares:~$ cat StacksAndQueues.info
 /***************************************************************************
     *                                                                      *
     *  NAME:  Leia Ladios                        CLASS:  CSC121-W01        *
     *                                                                      *
     *  Assignment: Stacks and Queues HW          Level:   4.5              *
     *                                                                      *
     *  Description:                                                        *
     *                                                                      *
        R-6.1:  Suppose an initially empty stack S has performed:
                        25 push operations,
                        12 top operations,
                        and 10 pop operations(3 of which returned null
                        to indifcate an empty stack).
                What is the current size of the stack?

         R-6.2: Given Code Fragment 6.2, what would be the final value
                of the instance variable t for the previous problem?

         R-6.5: Give a recursive method for removing all the elements from
                the stack.

         C-6.24: Describe how you can use a queue to scan S to see if it
                contains an element x, with an additional constraint
                that you return all the elements back into the stack in
                their original order. You may only use the stack, queue,
                and a constant number of other primitive variables.

         C-6.30: Give pseudocode description of an array-based implementation
                of the double ended queue ADT. What is the running time for
                each operation.

         C-6.31: Describe how to implement the deque ADT using two stacks
                as the only instance variables. What are the running times
                of the methods?

     *                                                                      *
    ***************************************************************************/



l_ladios@ares:~$ cat StacksAndQueuesHW.txt
cat: StacksAndQueuesHW.txt: No such file or directory
l_ladios@ares:~$ cat StacksAndQueuesHW
R-6.1
Pushing 25 operations means 25 elements get added to the
stack. Top operations do not affect the size of the stack
as it merely returns the element at the top of the stack
without removing it. Having 10 pop operations mean that 10
elements are removed from the stack. However, 3 of those
pop operations returned null. This means that they happened
when the stack was empty. This means that only 7 pop
operations really affected the size of the stack.
With that, the size of the stack is 25-7 = 18.

R-6.2
The instance variable is t = -1 when a stack is instantiated.
A t = -1 indicates and empty stack. The size of the stack is
determined by the equation t + 1 = size. When the stack is empty,
the equation is t + 1 = 0. When the size of the stack is 15,
the equation is t + 1 = 18. This allows us to derive the value
of t in that scenario. We would get t = 18 -1 => t = 17.

R-6.5
Recursive method to remove all elements from a stack:
remove(){
        if(size() == 0){ //the size is updated in pop.
                return;
        }
        else{
                self.pop(); //returns removed element.
                                        //pop also reduces the size of the stack
                remove();
        }
}

C-6.24
We would have to pop from the stack. To make sure we don't loose
any of our elements, we can enqueue it into a queue Q. To determine
whether or not the stack S contains an element x, we can pop from
the stack, compare its value with the given element x. If it is not
a match, enqueue it into Q. Continue to pop off each element until a
match for x is found or you run out of elements to pop which would
indicate that the stack does not contain an element equivalent to x.

To satisfy the additional constraint that the algorithm must return
the elements back to S in their original order, we would have to
dequeque each element and push them back into the Stack. At the end
of this process, we will have the elements in a flipped order.
To maintain its original order, we will have to go through the
whole process all over again. We will have to pop each element from
the queue. When all the elements are back in the queue, we will
then push each element back into the stack. Going through the process
twice will allow us to maintain the original order of the Stack S.

C-6.30
An array-based implementation of a double ended queue ADT would utilize
a circular array. We must be able to add and remove from the start as
well as the end of the array.

To add to the front of the array:
//check if deque is full bc you can't add anymore
        if it is
//if the front of the array is at 0, make the front equal size - 1
                - this makes the front the last index of the array.
//else, do a regular front-- and enqueue the element given to the method
```

```
        by assigning it to the front index.

To add to the end of the array:
//check if deque is full bc you can't add anymore
        if it is
//if the rear is the end of the array(size-1),
        make rear = 0 which would be the next index
        to add to since the deque is not full.
//else just increment rear to keep adding after
        the current rear which has not reached the end of the deque.
        enqueue the given element passed int the method by assigning
        it to the rear index.

To remove from the front:
//check if the deque is empty
//if there is only one element in the array(front == rear),
        dequeue the element by storing it in a helper variable,
        setting that index to null/0, then setting front and
        rear index to be = -1.
        -1 rear and front mean the deque is empty.
//else if there are not any more elements in the circular array
        front should be made the to be index 0 of the array
//else
        front++;

To remove from the rear:
//check id deque is empty
/if there is only one element in the array(front == rear),
        dequeue the element by storing it in a helper variable,
        setting that index to null/0, then setting front and
        rear index to be = -1.
        -1 rear and front mean the deque is empty.
//else if there are not any more elements in the circular array,
        rear will have rear equal to be the index the size of the
        array less 1.
        rear = size - 1;
//else
        rear--;

Each operation has a constant runtime.

C-6.31
A deque allows us to add and remove from two places in a queue.
We are able to access elements from the top and bottom.
To implement a deque using two stacks as the only instance
variables, we will assign one stack to handle elements added
and removed from the top of the deque. The second stack will
handle elements added and removed from the bottom of the deque.
Since poping an element into a stack adds it to the top of the
stack, we have access to the most recent addition to either the
end or the top of the deque. The following methods are needed in
the implementation:

Stack<T> A = new Stack<T>();
Stack<T> B = new Stack<T>();
addTopA(t){
        A.push(t);
}

removeTopA(){
        A.pop(); //removes the top of stack A or top of Deque
}

addLastB(t){
        B.push(t);
}

removeLastB(){
        B.pop();
}

Each operation has a constant runtime.




l_ladios@ares:~$ exit
exit

Script done on 2021-03-04 19:20:17-0600
```