CP2410 Practical 04

Sihan Chen, jcu ID: 14187662

Question 1

Stack follows the rule of LIFO, method `push` add an element to the end of stack, does not return anything; method `pop` removes and returns the last element of stack. So, here should be the result of the aforementioned series of operations.

```
stack = ArrayStack()
stack.push(5)   # return None; stack = [5]
stack.push(3)   # return None; stack = [5, 3]
stack.pop()     # return 3,  ; stack = [5]
stack.push(2)   # return None; stack = [5, 2]
stack.push(8)   # return None; stack = [5, 2, 8]
stack.pop()     # return 8   ; stack = [5, 2]
stack.pop()     # return 2   ; stack = [5]
stack.push(9)   # return None; stack = [5, 9]
stack.push(1)   # return None; stack = [5, 9, 1]
stack.pop()     # return 1   ; stack = [5, 9]
stack.push(7)   # return None; stack = [5, 9, 7]
stack.push(6)   # return None; stack = [5, 9, 7, 6]
stack.pop()     # return 1   ; stack = [5, 9, 7]
stack.pop()     # return 9   ; stack = [5, 9]
stack.push(4)   # return None; stack = [5, 9, 4]
stack.pop()     # return 1   ; stack = [5, 9]
stack.pop()     # return 9   ; stack = [5]
```

Question 2

The size of stack S increases by 1 upon `push`, decreases by 1 upon **successful** `pop`, and unchanged upon `top`.

Since `pop` and `top` are both capable of rasing Empty error, but only `pop` removes element from stack, there are four possible senarios:

1. all three raised by `pop`

   in this case, only 10 - 3 = 7 `pop` were succesfully executed, so the size of stack S is 25(pushes) - 7(pops) = 18

2. two out of three raised by `pop`

   in this case, only 10 - 2 = 8 `pop` were succesfully executed, so the size of stack S is 25(pushes) - 8(pops) = 17

3. one out of three raised by `pop`

in this case, only 10 - 1 = 9 `pop` were succesfully executed, so the size of stack S is 25(pushes) - 9(pops) = 16

4. all three raised by `top`

   in this case, all 10 `pop` were succesfully executed, so the size of stack S is 25(pushes) - 10(pops) = 15

Question 3

Here is the implementated and testing code:

```
def transfer(S, T):
  while not S.is_empty():
    ele = S.pop()
    T.push(ele)

if __name__ == '__main__':
  S = ArrayStack()
  T = ArrayStack()
  for i in range(10):
    S.push(random.randint(0, 10))
  print(f"Now stack S has ten random elements: {S._data}")
  print(f"Stack T is empty: {T._data}")
  transfer(S, T)
  print(f"After transfer, stack S: {S._data}")
  print(f"After transfer, stack T: {T._data}")

NORMAL array_stack.py                         dos | utf-8 | pytho
"array_stack.py" 79L, 2615C written
```

And here is the result:

```
pwsh ~\Dev\cp2410\Week04  main +1                         98
> python .\array_stack.py
Now stack S has ten random elements: [2, 4, 3, 4, 6, 8, 1, 6, 4, 5]
Stack T is empty: []
After transfer, stack S: []
After transfer, stack T: [5, 4, 6, 1, 8, 6, 4, 3, 4, 2]
```

Question 4

Queue follows the rule of LILO, method `enqueue` add an element to the end of stack, does not return anything; method `dequeue` removes and returns the first element of stack. So, here should be the result of the aforementioned series of operations.

```
queue = ArrayQueue()
queue.enqueue(5)   # return None; queue = [5]
queue.enqueue(3)   # return None; queue = [5, 3]
queue.dequeue()    # return 3,  ; queue = [3]
queue.enqueue(2)   # return None; queue = [3, 2]
queue.enqueue(8)   # return None; queue = [3, 2, 8]
queue.dequeue()    # return 8   ; queue = [2, 8]
queue.dequeue()    # return 2   ; queue = [8]
queue.enqueue(9)   # return None; queue = [8, 9]
queue.enqueue(1)   # return None; queue = [8, 9, 1]
queue.dequeue()    # return 1   ; queue = [9, 1]
```

```
queue.enqueue(7)    # return None; queue = [9, 1, 7]
queue.enqueue(6)    # return None; queue = [9, 1, 7, 6]
queue.dequeue()     # return 1   ; queue = [1, 7, 6]
queue.dequeue()     # return 9   ; queue = [7, 6]
queue.enqueue(4)    # return None; queue = [7, 6, 4]
queue.dequeue()     # return 1   ; queue = [6, 4]
queue.dequeue()     # return 9   ; queue = [4]
```

Question 5

The size of queue Q increases by 1 upon enqueue, decreases by 1 upon **successful** dequeue. So, the current size of queue Q should be 32(enqueues) - 15(dequeues) = 17

Question 6

To get elements from D to Q, here are the required operations:

operation	D(deque)	Q(queue)
	(1, 2, 3, 4, 5, 6, 7, 8)	()
Q.enqueue(D.delete_first())	(2, 3, 4, 5, 6, 7, 8)	(1)
Q.enqueue(D.delete_first())	(3, 4, 5, 6, 7, 8)	(1, 2)
Q.enqueue(D.delete_first())	(4, 5, 6, 7, 8)	(1, 2, 3)
Q.enqueue(D.delete_first())	(5, 6, 7, 8)	(1, 2, 3, 4)
Q.enqueue(D.delete_first())	(6, 7, 8)	(1, 2, 3, 4, 5)
Q.enqueue(D.delete_first())	(7, 8)	(1, 2, 3, 4, 5, 6)
Q.enqueue(D.delete_first())	(8)	(1, 2, 3, 4, 5, 6, 7)
Q.enqueue(D.delete_first())	()	(1, 2, 3, 4, 5, 6, 7, 8)

Question 7

To get elements from D to S, here are the required operations:

operation	D(deque)	S(stack)
	(1, 2, 3, 4, 5, 6, 7, 8)	()
S.push(D.delete_first())	(2, 3, 4, 5, 6, 7, 8)	(1)
S.push(D.delete_first())	(3, 4, 5, 6, 7, 8)	(1, 2)
S.push(D.delete_first())	(4, 5, 6, 7, 8)	(1, 2, 3)
S.push(D.delete_first())	(5, 6, 7, 8)	(1, 2, 3, 4)
S.push(D.delete_first())	(6, 7, 8)	(1, 2, 3, 4, 5)

operation	D(deque)	S(stack)
S.push(D.delete_first())	(7, 8)	(1, 2, 3, 4, 5, 6)
S.push(D.delete_first())	(8)	(1, 2, 3, 4, 5, 6, 7)
S.push(D.delete_first())	()	(1, 2, 3, 4, 5, 6, 7, 8)