Design a stack with operations on middle element

How to implement a stack which will support following operations in **O(1) time complexity**?

1. push() which adds an element to the top of stack.
2. pop() which removes an element from top of stack.
3. findMiddle() which will return middle element of the stack.
4. deleteMiddle() which will delete the middle element.
   Push and pop are standard stack operations.
   The important question is, whether to use a linked list or array for implementation of stack?
   Please note that, we need to find and delete middle element. Deleting an element from middle is not O(1) for array. Also, we may need to move the middle pointer up when we push an element and move down when we pop(). In singly linked list, moving middle pointer in both directions is not possible.
   The idea is to use Doubly Linked List (DLL). We can delete middle element in O(1) time by maintaining mid pointer. We can move mid pointer in both directions using previous and next pointers.
   Following is implementation of push(), pop() and findMiddle() operations. Implementation of deleteMiddle() is left as an exercise. If there are even elements in stack, findMiddle() returns the second middle element. For example, if stack contains {1, 2, 3, 4}, then findMiddle() would return 3.

```
class DLLNode:

    def __init__(self, d):
        self.prev = None
        self.data = d
        self.next = None



''' Representation of the stack
data structure that supports
findMiddle() in O(1) time. The
Stack is implemented using
Doubly Linked List. It maintains
pointer to head node, pointer
to middle node and count of
nodes '''
```

```python
class myStack:

    def __init__(self):
        self.head = None
        self.mid = None
        self.count = 0


''' Function to create the stack data structure '''


def createMyStack():
    ms = myStack()
    ms.count = 0
    return ms


''' Function to push an element to the stack '''


def push(ms, new_data):
    ''' allocate DLLNode and put in data '''
    new_DLLNode = DLLNode(new_data)

    ''' Since we are adding at the beginning,
    prev is always NULL '''
    new_DLLNode.prev = None

    ''' link the old list off the new DLLNode '''
    new_DLLNode.next = ms.head

    ''' Increment count of items in stack '''
    ms.count += 1

    ''' Change mid pointer in two cases
    1) Linked List is empty
    2) Number of nodes in linked list is odd '''
    if(ms.count == 1):
        ms.mid = new_DLLNode

    else:
        ms.head.prev = new_DLLNode
        # Update mid if ms->count is odd
        if((ms.count % 2) != 0):
            ms.mid = ms.mid.prev

    ''' move head to point to the new DLLNode '''
    ms.head = new_DLLNode


''' Function to pop an element from stack '''


def pop(ms):
    ''' Stack underflow '''
    if(ms.count == 0):

        print("Stack is empty")
        return -1

    head = ms.head
    item = head.data
    ms.head = head.next

    # If linked list doesn't become empty,
    # update prev of new head as NULL
    if(ms.head != None):
        ms.head.prev = None
    ms.count -= 1

    # update the mid pointer when
    # we have even number of elements
    # in the stack, i,e move down
    # the mid pointer.
    if(ms.count % 2 == 0):
        ms.mid = ms.mid.next
    return item

# Function for finding middle of the stack


def findMiddle(ms):
    if(ms.count == 0):
        print("Stack is empty now")
        return -1
    return ms.mid.data


# Driver code
if __name__ == '__main__':

    ms = createMyStack()
    push(ms, 11)
    push(ms, 22)
    push(ms, 33)
    push(ms, 44)
    push(ms, 55)
    push(ms, 66)
    push(ms, 77)

    print("Item popped is " +
          str(pop(ms)))
    print("Item popped is " +
          str(pop(ms)))
    print("Middle Element is " +
          str(findMiddle(ms)))
```

My code:

```
class DLL:
    def __init__(self,val):
        self.val = val
        self.prev = None
        self.next = None



class DesignStack:
    def __init__(self):
        self.count = 0
        self.head = None
        self.mid = None

    def push(self, val):
        node = DLL(val)
        if self.head is None:
            self.head = node
        else:
            node.next = self.head
            self.head.prev = node
            self.head = node
        self.count = self.count+1
        if self.count==1:
            self.mid = self.head
        else:
            if self.count%2==0:
                self.mid = self.mid.prev


    def pop(self):
        if self.count==0:
            return "Underflow of Stack"

        self.count = self.count - 1
        if self.count==0:
            curr = self.head
            self.head = None
            self.mid = None
        else:
            curr = self.head
            self.head = self.head.next
            self.head.prev = None
            curr.next = None
        if self.count==0:
            self.mid = None
        else:
            if self.count==1:
                self.mid = self.head
            elif self.count%2!=0:
                self.mid = self.mid.next
        return curr.val


    def midEle(self):
        if self.mid is not None:
            return self.mid.val
        else:
            return None
```

```python
ms = DesignStack()
ms.push(5)
ms.push(10)
ms.push(15)
ms.push(20)
ms.push(25)
ms.push(30)

print(ms.pop())
print(ms.pop())
print(ms.pop())
print(ms.pop())
print(ms.pop())
# print(ms.pop())
# print(ms.head.val)
print(ms.midEle())
```