Programming assignment 2: Queues, stacks and singly-linked lists

*It is very important when running programs that names and references are **exactly** correct. If a programming assignment has tests that import a file with a specific name, and use classes and operations with specific names, a student's submission must include implementations of classes and operations with **exactly** those names in files with **exactly** those names.*
*The names of operations and classes given in assignment descriptions are **exactly** the names that must be used.*

20% Implement the class ***ArrayDeque,*** including the following operations:

- push_back
  - Takes a parameter and adds its value to the back of the deque
- push_front
  - Takes a parameter and adds its value to the front of the deque
- pop_back
  - Removes the item from the back of the deque and ***returns*** *its value*
    - *If the deque is empty, return None*
- pop_front
  - Removes the item from the front of the deque and ***returns*** *its value*
    - *If the deque is empty, return None*
- get_size
  - Returns the number of items currently in the deque
- __str__
  - Returns a string with all the items in the deque, separated by a single space

There must not be a limit on how many items can be added to the list. Limitations on the use of python lists apply to this assignment, as they have in previous assignments on arrays.
*For a bonus 5%, implement all operations (*apart from __str__*) O(1) (amortized)*

20% Implement the *singly-linked list* class ***LinkedList*** including the following operations:

- push_back
  - Takes a parameter and adds its value to the back of the list
- push_front
  - Takes a parameter and adds its value to the front of the list
- pop_front
  - Removes the item from the front of the list and ***returns*** *its value*
    - *If the list is empty, return None*
- get_size
  - Returns the number of items currently in the list
- __str__
  - Returns a string with all the items in the list, separated by a single space

*For full marks, implement all these operations (*apart from __str__*) with time complexity O(1)*

20% Implement the class ***Stack,*** including the following operations:

- push
    - Takes a parameter and adds its value onto the stack
- pop
    - Removes the item off the top of the stack and ***returns*** *its value*
        - *If the stack is empty, return None*
- get_size
    - Returns the number of items currently on the stack

The class should own (as an instance variable) an instance of ***ArrayDeque*** *or* ***LinkedList*** *and implement its own operations* ***only*** *with forwarding calls to the operations of the encapsulated container.*

The class ***Stack*** should take ***type*** as a parameter in its constructor. ***type*** is a string, and if it 1equals "array" it uses the ***ArrayDeque*** as a container, if type equals "linked" it should use ***LinkedList***.

20% Implement the class ***Queue***, including the following operations:

- add
    - Takes a parameter and adds its value to the back of the queue
- remove
    - Removes the item off the front of the queue and ***returns*** *its value*
        - *If the queue is empty, return None*
- get_size
    - Returns the number of items currently in the queue

The class should own (as an instance variable) an instance of ***ArrayDeque*** *or* ***LinkedList*** *and implement its own operations* ***only*** *with forwarding calls to the operations of the encapsulated container.*

The class ***Queue*** should take ***type*** as a parameter in its constructor. ***type*** is a string, and if it equals "array" it uses the ***ArrayDeque*** as a container, if type equals "linked" it should use ***LinkedList***.

20% Implement the function ***palindrome(head)*** using recursive programming.

- The function checks if a singly-linked list is the same forwards and backwards.
    - *abba and calelac are palindromes, adba is not a palindrome*
- The function takes a node (***head***) as a parameter and returns ***True*** if the list is a palindrome, otherwise ***False***.

This can be done with more than one separate recursive calls that may initialize new instances of Node, or move data around. As long as all runs through the list/lists are ***recursive***, and the ***original list*** sent in is not broken in any way, full marks will be given.

Points can be deducted for unnecessarily complex code or memory allocation.

Solutions that put the data into a different type of data structure to solve it do not count!

Bonus 5% will be given for solutions that only go once through the list recursively and do not change the list in any way or allocate memory for new nodes or their data.