Mock Midterm. Work alone for the first 30 minutes. (This will NOT be graded, but you will get attendance points for the lab.) **NOTE:** These problems were written by the instructor.

---

1. Write a function `is_palindrome(astr)` that returns `True` if the string `astr` is a palindrome and `False` otherwise. (A palindrome is a string that reads the same forward or backward.) The string may have both upper and lowercase letters and spaces. Your function should be case-insensitive and ignore spaces. Below are some example calls and returns:

```
print(is_palindrome("A man a plan a canal Panama"))  # Output: True
print(is_palindrome("Radar"))                        # Output: True
print(is_palindrome("hello"))                        # Output: False
```

   Requirement: Use a stack to implement the function (not recursion). Recall that the methods of the `Stack` class are `push()`, `pop()`, and `is_empty()`.

2. Write a ***recursive*** function `flatten_to_string(alist)` that takes a nested list (a list that may contain other lists) `alist` and returns a single string containing all the elements of the list concatenated together. Below is an example use of the function:

```
nested_list = [1, ['a', 'b'], 2, ['c', [3, 4]], 'd']
result = flatten_to_string(nested_list)
print(result)                                  # Output: "1ab2c34d"
```

Note: To determine if an element is a list, use this comparison: `type(alist[0]) == list`

3. (Short answer.)

a) What does FIFO stand for?

b) What does LIFO stand for?

c) In a binary tree, each node is either a leaf node or a node that has two children. (True or False)

d) An Abstract Data Type (ADT) describes a set of data values and associated operations that are allowed on the data. It also specifies how to implement the data type. (True or False)

4. For the problem below, use the following implementations of `Node` and `LinkedList`.

```
class LinkedList:
    def __init__(self):
        self._head = None

    def add(self,new):
        new._next = self._head
        self._head = new
```

```
class Node:
    def __init__(self,value):
        self._value = value
        self._next = None
```

Note: You may access the attributes directly without getter and setter methods. You may not change the implementation of `LinkedList` and `Node`.

Write a method `remove_node(self, value)` for the `LinkedList` class that removes the node whose `_value` attribute equals `value`. The method modifies the linked list and has no return value. **Note:** If `value` occurs in the linked list more than once, only the first occurrence is removed.

5. For the following problem, use the implementation of a `BinaryTree` class below:

```
class BinaryTree:
    def __init__(self, value):
        self._value = value
        self._left = None
        self._right = None
```

Also, assume that `value()`, `left()`, and `right()` are the usual getters for the attributes.

Write a recursive function `sum_odds(tree)` that takes a binary tree `tree` and returns the sum of the values in the tree that are *odd*.

6. Given the preorder and inorder traversals below, draw the resulting tree.

Preorder: 3, 6, 4, 5, 8, 9, 2
Inorder: 4, 6, 3, 5, 9, 8, 2