Lab 4: Abstract Data Type Solution

1) Stack review

1. Write a function that takes a stack of integers and removes all of the items which are greater than 5. The other items in the stack, and their relative order, should remain unchanged.

```
    ...
    def remove_big(s: Stack) -> None:
        """Remove the items in <stack> that are greater than 5.

        Do not change the relative order of the other items.

        >>> s = Stack()
        >>> s.push(1)
        >>> s.push(29)
        >>> s.push(8)
        >>> s.push(4)
        >>> remove_big(s)
        >>> s.pop()
        4
        >>> s.pop()
        1
        >>> s.is_empty()
        True
        """
        side_stack = Stack()

        # 1. Move all elements in stack 1 to stack 2
        while not s.is_empty():
            # 1.1 While moving elements, if element value is greater
    than 5, then pass
            popped_element = s.pop()

            if type(popped_element) == int and popped_element > 5:
                continue

            side_stack.push(popped_element)

        # 2. Move back all elements from stack 2 back to stack 1.
        while not side_stack.is_empty():
            s.push(side_stack.pop())
```

```
    ...

```

Listing 1: task_1_q1_solution.py

2. Write a function that takes a stack and returns a new stack that contains each item in the old stack twice in a row. We'll leave it up to you to decide what order to put the copies into in the new stack.

   Note that because the docstring doesn't say that the old stack will be mutated, the old stack should remain unchanged when the function returns.

```
    ...
    def double_stack(s: Stack) -> Stack:
        """Return a new stack that contains two copies of every item in
    <stack>.

        We'll leave it up to you to decide what order to put the copies
    into in
        the new stack.

        >>> s = Stack()
        >>> s.push(1)
        >>> s.push(29)
        >>> new_stack = double_stack(s)
        >>> s.pop()  # s should be unchanged.
        29
        >>> s.pop()
        1
        >>> s.is_empty()
        True
        >>> new_items = []
        >>> new_items.append(new_stack.pop())
        >>> new_items.append(new_stack.pop())
        >>> new_items.append(new_stack.pop())
        >>> new_items.append(new_stack.pop())
        >>> sorted(new_items)
        [1, 1, 29, 29]
        """
        side_stack = Stack()
        output_stack = Stack()

        # 1. Move elements from original_stack to side_stack
        while not s.is_empty():
            side_stack.push(s.pop())

        # 2. Move back elements from side stack to original stack
        while not side_stack.is_empty():

            popped_element = side_stack.pop()

            # 2.1 When moving back element, copy element twice and push
    to output_stack
```

```
            copied_element = copy.deepcopy(popped_element)
            for i in range(2):
                output_stack.push(copied_element)

            s.push(popped_element)
    ...

```

Listing 2: task_1_q2_solution.py