CS256 Lab 6-2 - Build vs buy, part 2

Value: 30 points (plus 10 points extra credit)
Due: Friday 18 October @ 23:50 (design document due for class on Wednesday 16 October),
10% bonus for submitting it before Friday @ 18:00!
Submission Format: Design document (image) and two or three Python scripts, lab_06-2_stack.py, and lab_06-2_queue.py, optionally lab_06-2_deque.py

In the first part of this lab you designed and built the basic data structures we have been working with recently: stacks, queues, and optionally deques; using your now modified dynamicArray class from Lab 5 as the underlying storage for each of them. Part 2 is about comparing the performance and ease of use of hand-built data structures and the built-in equivalents available in Python.

Setup

Using Python lists, which support both stack (LIFO) and queue (FIFO) semantics, you will process a bunch of numbers and display some results. In essence this is an abstraction of Lab 6, part 1; do the exact same high-level tasks only this time use the native tools rather than your hand-built ones. For testing you can use the same data files as Lab 6, part 1.

Tasks

1. Design, code and test a script (lab_06-2_stack.py) that contains:
    a. A method that given an input file populates a list using stack (LIFO) semantics by successively calling push() on the input file of integers, returning that stack to the caller.
    b. A method that given a populated stack of integers determines if they are a palindrome using only stack (LIFO) semantics, returns True if they are, False if not. The passed stack does not need to be preserved.
    c. A method that given a populated stack of integers returns a new stack whose contents are the reverse of the original stack using only stack (LIFO) semantics, that is the former top is on the bottom, etc. The passed stack does not need to be preserved.
    d. A method that given a populated stack calculates and returns the count and sum of the values stored in it using only stack (LIFO) semantics. The stack can be changed (even completely pop()'d out if you will…) by this method.
    e. Your main() will roughly need to:
        1. process the command line argument for the input file checking that it exists
        2. pass filename to your a,
        3. make a deep copy of your returned stack, call your b passing it the deep copy of the stack, print the results to stdout,
        4. make another deep copy of your stack, call your c passing it the second deep copy of the stack,
        5. make a deep copy of the returned stack (henceforth known as current), and with the copy call your b again and print the result to stdout,
        6. print the top() of the current stack to stdout,
        7. and finally call your d passing it the current stack and printing the result to stdout. Your output should be a simple list of four items (these are just an example):

            False
            False
            9
            2567, 5667

2. Design, code and test a script (lab_06-2_queue.py) that contains:
    a. A method that given an input file populates a list using queue (FIFO) semantics by successively calling enque() on the input file of integers, returning that stack to the caller.
    b. A method that given a populated queue of integers determines if they are a palindrome using only queue (FIFO) semantics, returns True if they are, False if not. The passed queue does not need to be preserved.
    c. A method that given a populated queue of integers returns a new queue whose contents are the reverse of the original queue using only queue (FIFO) semantics, that is the former front is at the end, etc. The passed queue does not need to be preserved.
    d. A method that given a populated queue calculates and returns the count and sum of the values stored in it using only queue (FIFO) semantics. The queue does not need to be preserved.
    e. Your main() will roughly need to:
        1. process the command line argument for the input file checking that it exists
        2. pass filename to your a,
        3. make a deep copy of your returned queue, call your b passing it the deep copy of the queue, print the results to stdout,
        4. make another deep copy of your queue, call your c passing it the second deep copy of the queue,
        5. make a deep copy of the returned queue (henceforth known as current), and with the copy call your b again and print the result to stdout,
        6. print the front value of the current queue to stdout,
        7. and finally call your d passing it the current queue and printing the result to stdout.

        Your output should be a simple list of four items (these are just an example):

        False
        False
        9
        2567, 5667

Implementation Notes

- Design your solution on paper first. Most people would start by reviewing the whole , and then designing the low-level elements (the class/method definitions) and then move to the higher level tasks. Update it as you develop and refine your thinking. Make sure your design covers all of the assignment, you will be turning-in an image of it on Wednesday and Friday.
- Your script should use the main() technique, it helps organize the code and it makes reuse much easier.
- Note the requirement to only use the native semantics of the data structure in-use for each of the methods.
- The input file name should be a required command line argument (-i).
- Check to make sure the file exists before trying to open it.
- Use the same data files as Lab 6, part 1. One is small, one is large. Use the small one for testing (and/or make your own), use the large one when running the assignment to test if it is correct before submitting it.
- Code neatness and organization count, a lot. Less is more. Pay close attention to your style. Use pylint.
- The copy.deepcopy() works on Python's native lists, "import copy" gives you access to this.
- Make sure you follow the file naming conventions specified in blue above.

Extra Credit

- Design, code and test an integer deque using the same technique. Create a reasonable test scenario for it following the pattern above. Wrap all of this in a file named lab_06-02_deque.py and submit it to Moodle with your other files.