CS256 Lab 6 - Build vs buy, part 1

Value: 30 points (plus 10 points extra credit)
Due: Friday 11 October @ 23:50 (design document due for class on Wednesday 9 October)
Submission Format: Design document (image) and three Python scripts, myDataStructures.py, lab_06_stack.py, and lab_06_queue.py

The first part of this lab is about designing and building implementations of the basic data structures we have been working with recently: stacks, queues, and optionally deques; using the dynamicArray class from Lab 5 as the underlying storage for each of them. The second part of this lab, Part 2, will be about comparing the performance and ease of use of hand-built data structures and the built-in equivalents available in Python.

Setup

As part of Lab 5 you improved the dynamicArray class and its associated methods (dynamic_array.py). For this lab you will use that improved class definition as the basis of two (or three) abstract data types, stacks, queues, and possibly deques. Using those new objects you will process a bunch of numbers and display some results. If you haven't yet produced a correct version of Lab 5 now would be a good time to take advantage of the resubmission policy and finish that up. As before we will provide data files for testing.

Tasks

1. Design, code, and test an integer stack class. It should support push(), pop(), and top() methods. This class should be based on your dynamicArray class.
2. Design, code and test an integer queue. It should support enqueue(), dequeue(), and front() methods (front() is to a queue as top() is to a stack). This class should be based on your dynamicArray class.
3. Package your class definitions up in a file named myDataStructures.py This source file should have a main() section that tests the basic functionality of all the classes and methods it contains.
4. Design, code and test a script (lab_06_stack.py) that imports myDataStructures.py and contains:
    a. A method that given a stack object (of your design) and an input file populates that stack by successively calling push() on the input file of integers.
    b. A method that given a populated stack of integers determines if they are a palindrome, returns True if they are, False if not.
    c. A method that given a populated stack of integers returns a new stack whose contents are the reverse of the original stack, that is the former top is on the bottom, etc.
    d. A method that given a populated stack calculates and returns the count and sum of the values stored in it. The stack can be changed (even completely pop()'d out if you will…) by this method.
    e. Your main() will roughly need to process the command line argument for the input file checking that it exists, instantiate one of your stacks, call your a, make a deep copy of your stack, call your b passing it the deep copy of the stack, print the results to stdout, make another deep copy of your stack, call your c passing it the second deep copy of the stack, with the returned stack call your b again and print the result, print the top() of that stack to stdout, and finally call your d passing it your stack and printing the result to stdout. Your output should be a simple list of three items (these are just an example):
        False
        False
        9
        2567, 5667
5. Design, code and test a script (lab_06_queue.py) that imports myDataStructures.py and contains:

a. A method that given a queue object (of your design) and an input file populates that queue by successively calling enque() on the input file of integers.
b. A method that given a populated queue of integers determines if they are a palindrome, returns True if they are, False if not.
c. A method that given a populated queue of integers returns a new queue whose contents are the reverse of the original queue, that is the former front is at the back, etc.
d. A method that given a populated queue calculates and returns the count and sum of the values stored in it. The queue can be changed (even completely dequeue()'d out if you will…) by this method.
e. Your main() will roughly need to process the command line argument for the input file checking that it exists, instantiate one of your queues, call your a, make a deep copy of your queue, call your b passing it the deep copy of the queue, print the results to stdout, make another deep copy of your queue, call your c passing it the second deep copy of the queue, with the returned queue call your b again and print the result, print the front() of that queue to stdout, and finally call your d passing it your queue and printing the result to stdout. Your output should be a simple list of three items (these are just an example):

```
False
False
9
2567, 5667
```

Implementation Notes

1. Design your solution on paper first. Most people would start by reviewing the whole enchilada, 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.
2. The input file name should be a required command line argument (-i).
3. Check to make sure the file exists before trying to open it.
4. We will provide data files of integers for you to use in testing. 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.

Extra Credit

- Design, code and test an integer deque. It should support enqueueFront(), dequeueFront(), enqueueBack(), dequeueBack(), front(), and back() methods (front() and back() are to a deque as top() is to a stack). This class should be based on your dynamicArray class. Create a reasonable test scenario for it in a file named lab_06_deque.py and submit it to Moodle with your other files.