`class arStack, llStack, arQueue, llQueue`

1. Implement and test four classes `arStack{}, llStack{}, arQueue{}`, and `llQueue{}` such that the following operations are constant time `O(1)`. For `xxStack{}`, implement `xxStack::push(), xxStack::pop(), xxStack::top(), xxStack::isEmpty(), xxStack::isFull()`. For `xxQueue{}`, implement `xxQueue::enq(), xxQueue::deq(), xxQueue::front(), xxQueue::isEmpty(), xxQueue::isFull()`. Test each of your stack/queue implementations by filling them with the words from a file whose name is given as the first argument to your program (you will test your program on `words.txt`) then emptying them printing the words, as you remove them and write them to an output file whose name is given as the second program argument. Use an appropriately named file, such as `arstackoutput.txt` for `arStack{}`. For example, to write two separate programs which test both implementations of stack and queue:

```
$ test_stack words.txt arstackoutput.txt  llstackoutput.txt
$ test_queue words.txt arqueueOutput.txt  llqueueoutput.txt
```

2. Write two abstract base classes; one for Stack and another for Queue, so you can reuse the same testing functions to test each of your stack and queue implementations respectively. You will need to make the public methods (and destructor) "virtual" for this to work correctly. You should write functions (not methods), `fillAll()` and `emptyAll()` that take an appropriate container (either stack or queue), passed by reference, and enter the words from the input file into that specific container, then remove the words, one at a time, printing them to the output file as you remove them.

3. Implement a function `isBalanced()`, that takes in a string of brackets and returns true only if the brackets match properly. Handle three types of brackets: (), {}, and []. Assume N is the length of your paren string for doing complexity of this function. Use your LinkedStack implementation to handle the nesting of parentheses. E.g.

```
// returns true because all are correctly matched and balanced:
isBalanced("({(())})((([({})])))((((((()([{()}])(()))))))()")

// returns false because missing a closing paren:
isBalanced("({(())})((([({})])))((((((()([{()}])((((((")

// returns false because it has too many closing square brackets:
isBalanced("({(())})((([({})])))((((((()([{()}])(()))))))()])")
```

spoiler: select below to see it, but only after trying to solve yourself:

Always write the correct time complexity of each member function (e.g., `push(), pop(), enq(), deq()`) and each regular function (e.g., `fillAll(), emptyAll()`) in a comment next to each function. Points will be deducted for each error in time complexity and for each incorrect method or function up to the maximum. For this homework, N is the number of words from the [provided] file `words.txt`.