ICS 46
Homework Assignment 4

Programming Assignment:

1. (4 X 15 points each = 60 points) Implement and test **four classes** ArrayStack, LinkedStack, ArrayQueue, and LinkedQueue such that the following operations are constant time O(1). For Stack, implement push, pop, top, isEmpty, isFull. For Queue, implement enq, deq, front, isEmpty, 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 random.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 LinkedStackOutput.txt for the LinkedStack). e.g., to write two separate programs - one that tests both implementations of stack and queue
   test_stack random.txt ArrayStackOutput.txt LinkedStackOutput.txt
   test_queue random.txt ArrayQueueOutput.txt LinkedQueueOutput.txt
2. (10 points) Write two abstract base classes - one for Stack and another for Queue - so you can re-use 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. (10 points) (Link to my lecture notes on C++ exceptions) Define exception classes ContainerOverflow and ContainerUnderflow and have your stack and queue containers throw the appropriate exception when a remove (pop or deq) is done on an empty container or when an enter (push or enq) is done on a full container. These exception classes should have one string data member to hold a message about what caused the exception to be thrown. The constructor should take a c-string parameter to initialize this string data member.
4. (20 points) Implement a function isBalanced, that takes in a string of brackets and returns true only if the brackets match properly. Handle four 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.

isBalanced("({(())})((([({})]))))((((((()([{()}])(()))))))()") return true because all are correctly matched and balanced
isBalanced("({(())})((([({})]))))((((((()([{()}])(()))))))") returns false because it is missing a closing paren
isBalanced("({(())})((([({})]))))((((((()([{()}])(()))))))()]") returns false because it has too many closing square brackets

   SPOILER (select the text to see it, but don't look until after you have tried to solve it by yourself):

5. (up to 20 points deducted for missing time complexity) Always write the correct time complexity of each member function (e.g., push, pop, enq, deq) and each regular function (e.g., fillAll and emptyAll) in a comment next to each function. Five 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 file random.txt.

Homeworks that are not formatted nicely will not be accepted. Pay careful attention to memory management. Obvious code copying will be down graded as well. Feel free to add additional methods or data members as needed, for example, it may be a good idea to write a print method so you can see what your data structures look like.

1. Submit two things: Your report to gradescope and your zip file of a directory named by your UCI Net ID containing all your programs and a Makefile to build them.
    - a single report in .pdf format that shows your program works correctly when run under valgrind. Your report must follow the specified format, then submitted to GradeScope.
    - You must also submit a zip of your program which includes Makefile that builds your program called testMystring by compiling testMystring.cpp with the required warning flags.
2. Copy/Paste your code for the functions (with their time complexity assuming N is the number of words in the input file) related to fillAll and emptyAll for each of the two different linked list implementations of LinkedStack and LinkedQueue, (should be about 6 functions), e.g.,

```
void fillAll(Queue & q) // O(N!)   just kidding ;-)  but you should put the correct time complexity
{
    // your code here
}
void enque(string s) { }
void isFull() { }
void emptyAll(Queue & q) { }
string deque() { }
bool isEmpty() { }
```

Homeworks that are not formatted nicely will not be accepted. Pay careful attention to memory management. Do not share memory across objects. Obvious code copying, from a textbook, from another student, or within your own program, will be down-graded (write everything once (perhaps by writing auxillary functions), then call the function or method each time that operation is required). Feel free to add additional functions or methods, but please use the data members outlined in the homework.