COS2021 - Homework 2
Stacks and Queues: What are They Good For?

Due: Sept 30th: before midnight

Version: 00

Task Overview and Learning Goals

In this assignment, you will create an implementation for a Stack class and a Queue class based on the IContainer interface provided by me. Your Stack and Queue classes should make use of the linked node design, using the Node class. Once you have created and tested these two classes, you will use them to decipher a slightly encoded bit of text.

- Practice advanced C++ concepts, including inheritance, templates and classes.
- Develop an implementation that conforms to a pre-defined interface.
- Implement a stack and queue.
- Utilize the basic nature of two data structures to solve a problem.

Submission and Requirements

Submit your solution (all files) as zipped. I only want the code files (.hpp and .cpp) related to your project. See additional requirements below for

Your Stack and Queue implementations should be in a file called *implementations.hpp*. The main program used to solve the problem should in a file called *name_hw2.cpp*, where *name* is replaced with your full name.

Your program should not contain any kind of menu, interface or additional output. The only thing we should see is the decrypted string printed to the console.

Your code should contain a header comment with your name and any issues or bugs I should know about. Remember: most programs contain some type of bug and it is a sign of a mature coder to acknowledge issues (not hide them).

You do not need to submit the code.txt file. I will use a different one for grading to insure your submission can decode any text encrypted using this scheme.

Details

I have provided a C++ header file called *interfaces.hpp*. It contains two classes: IContainer class and the Node class. Look them over before starting. Read the instructions below very carefully. If anything is unclear or confusing, ask.

As mentioned above, your job is to design two classes, *Queue* and *Stack*, that inherit from the IContainer abstract class. Each class is explained below.

Queue Class

The Queue class should inherit from the abstract IContainer class and implement the required methods. Its *add* and *remove* methods should follow the FIFO (or first-in, first-out) behavior of a queue. The queue class has several additional requirements:

- Your implementation must use a *class template* design. You cannot hard code it to use strings or ints.
- The underlying data structure must be a linked list consisting of objects of the *Node* class found in the *interfaces.hpp* file. You may **not** use an array or vector to implement the queue.
- You should only implement the functions required by the IContainer class. No other functionality is required.

Stack Class

The stack class should inherit from the abstract IContainer class and implement the required methods. Its *add* and *remove* methods should follow the LIFO (or last-in, first-out) behavior of a stack. Like the queue, the stack has several requirements.

- Like the queue, your stack class should use a *class template* design. Again, you cannot hard code the stack class to store a particular type of data.
- Like the Queue class, you may not use an array or vector to represent the stack.
- Again, only implement the methods required by the abstract base class.

Testing

Before moving on to the actual problem, test both data structures. Write small driver programs that instantiate each class, add and remove elements. Make sure they follow their prescripted behaviors. Make sure data isn't lost. You do not need to submit these test files; however, do keep them in case I have questions about how you tested your work.

The Problem

With this assignment, I have provided a text file called *code.txt*. It contains a single sentence encrypted using a lazy but non-deterministic encryption method. However, decryption is deterministic. Here are the steps your solution must follow:

1. Read in the sentence from the file, character-by-character. Store each character (as a string) into an instance of your Queue class. This is the *character queue*.
2. Create another queue of strings, called the *sentence queue*. This will store the words of the decrypted sentences.
3. Create a stack of queues of strings called the *decode stack*.
4. LOOP:
5. Remove one character from the chararacter queue. We will refer to this character as C. Based on what C is, the algorithm will do one of three things.
   (a) If C is a (, then your program should create a new, empty queue of strings, called a *word queue* (to distinguish it from the character and sentence queues). If the current word queue is not empty, it should be first pushed onto the stack of queues.
   (b) If C is a), then your program should combine all the characters in the current word queue into a single string and add it to the sentence queue. If the decode stack is not empty, pop off the top queue and it becomes the current word queue.
   (c) If C is anything else, then your program should add it to the current word queue.
6. If the character queue is not empty, goto LOOP. Else, print out all the words in the sentence queue, with a space between each word, and quit.

Example

Assume that *code.txt* contains the following text:

```
(wo(sa(hello)d)rld)
```

Your program should print: *hello sad world*

Grading

It is not enough that the code solves the final problem. The solution must adhere to the requirements. It is possible for a submission to solve the program and receive a failing grade.

I will not try to crash your programs. This means, I will not give your programs malformed input. However, I do suggest you take every opportunity to develop code that fails gracefully.

I will test your code using a different encrypted sentence from the one provided with the homework.

If your program crashes, it will be hard to make above a 7 (or a C). Please test your code extensively.

- 3 pts Correct Queue Implementation
- 3 pts Correct Stack Implementation

- 3 pts Solves the problem
- 1 pt Miscellaneous (messy code, too much commented out code, poorly named variables, no header comment, very inefficient implementations, etc.).

The above breakdown does not mean you can skip part of the assignment. For example, only implementing the stack and queue and not solving the problem. Submissions that do not *attempt* to solve all parts of the assignment will be counted as incomplete and receive no higher than a 5. What constitutes an attempt is up to my discretion.