CISC 160
Lab 03

Background

We have discussed, in detail, the function of Stacks and Queues and how they are specifically implemented in Python. To get a better understanding of the utility of these data structures, we will be creating functions which use the data structures and comparing the functionality of each.

Other Resources

String object documentation in Python 3.

Palindromes

A palindrome is a string of characters that is the same both forwards and backwards. A full and robust definition of what a palindrome is can be found on Wikipedia. Colloquially, however, a palindrome is considered to be any string of data that is the same both forwards and backwards **after all spaces and punctuation has been removed and capitalization is normalized**. Consider, for example, the title of the 1971 book by David McCullough about the construction of the Panama Canal:

A Man, A Plan, A Canal: Panama!

This title is (hopefully obviously) not a palindrome if punctuation is left in as there is no exclamation point at the beginning of the sentence. However, with the punctuation and spaces removed and capitalization normalized, the title becomes:

*AMANAPLANA**C**ANALPANAMA*

The central C has been bolded and turned red to give you a point to focus on.

1. Download the files Stack_Interface.py, any of the stack classes mentioned in class and included with the lab, exceptions.py, and palindrome.py. from the lab page. Use (and do not alter) any one of the stack classes which extend the `Stack_Interface` class to implement the `is_palindrome_stack` function declared in palindrome.py. (**BE AWARE**: If you are planning to use the `Linked_Stack` object, remember that you will also need a copy of Node.py.) As mentioned in the associated comment, this function should take a string and use only stack objects in a *non-trivial way* in order to determine if the string, without spaces, without punctuation, and with normalized capitalization, is a palindrome. It should return `True` if it is a palindrome and `False` if it is not.

2. Download the file Queue_Interface.py and any of the queue classes mentioned in class and included with the lab from the lab page. Use (and do not alter) any one of the queue classes which extend the `Queue_Interface` class to implement the `is_palindrome_queue` function declared in palindrome.py. (**BE AWARE**: If you are planning to use the `Linked_Queue` object, remember that you will also need a copy of Node.py.) As mentioned in the associated comment, this function should take a string and use only queue objects in a *non-trivial way* in order to determine if the string, without spaces, without punctuation, and with normalized capitalization, is a palindrome. It should return `True` if it is a palindrome and `False` if it is not.

3. Examine both function implementations. Which of these functions is more efficient and why? (Note: You may use Big-O notation as part of your evidence but, even if you determine that they have the same Big-O functionality, you should be able to determine which of the two functions is nominally more efficient.)

Postfix Notation

Postfix notation, also sometimes called Reverse Polish Notation or RPN, is a type of mathematical notation that, by its very structure, is unambiguous without the need for parentheses for grouping. This notation was made famous by Hewlett Packard during the 1970's and 1980's when all handheld and desktop calculators (yes, such a thing used to exist) implemented postfix notation as their exclusive input method.

The algorithm for implementing postfix notation is fairly simple and famous:

```
//I) For each token in the expression
    //A) If the token is a number
        //1) Push the token on the stack
    //B) Else, if the token is an operator
        //1) Pop 2 tokens off the stack
        //2) Perform the operation on those 2 tokens (bottom token operator top token)
        //3) Push the resulting token back onto the stack
//II) Return the 1 token remaining on the stack
```

The above algorithm does not account for any errors that you could encounter in the processing of your string. These include unidentifiable tokens (such as alphabetical or non-mathematical operator characters), encountering an operator too soon (without two tokens on the stack) or having too many tokens.

Below are some examples of postfix notation and the resulting value:

3 4 +	7
5 6 + 3 *	33
5 + 6 * 3	ERROR!
2 1 + 4 – 8 *	-8
Pass	ERROR!
3 4 + 5	ERROR!

4. Download the file postfix.py from the lab page. Use (and do not alter) any one of the stack classes which extend the `Stack_Interface` class to implement the `postfix_calculator` function declared in postfix.py. (**BE AWARE**: If you are planning to use the `Linked_Stack` object, remember that you will also need a copy of Node.py.) As mentioned in the associated comment, this function should take a string and use a stack to implement the postfix notation algorithm implemented above. If your implementation encounters any problems during execution, it should throw the `RPNError` declared and defined in exceptions.py.

   NOTE: You may assume that all numbers inputted to the calculator are integers but you cannot assume that the output will be an integer. Make sure that your algorithm works for +, -, *, and /.

Lab Requirements

Download all source code files and alter palindrome.py and postfix.py as described above. You may add any additional helper functions as needed. Helper functions should be named with an underscore at the front as they are not designed to be publicly accessible If you add a helper function, be sure to comment it in the same style as the other, included functions.

Additionally, the "testing structure" of `if __name__ == '__main__'` has been included at the bottom of each file. You should use that section for your test code. I may look at it for understanding of how you are approaching testing but you will not be explicitly graded on code included there.

Remember: These requirements may not be all encompassing. Use your brain, your knowledge of the system, and any descriptions within the code as sanity checks and reminders to make a complete system.

Submission: Submit your altered palindrome.py and postfix.py and a file with the solution to Question 03. Submissions must be submitted in unzipped files.