605.202: Introduction to Data Structures (Spring 2022)

Mukul Sherekar

Lab-1 Analysis

Due Date: March 1, 2022

Date Turned In: March 1, 2022

Justification of data structure/design:

- Stacks are the main data structure for this project. The use of stacks was mandatory for this project. Initial stack helped to store the given string and eventually have the input characters in reverse order in order to process them right to left (algorithmically). This provided a nice flow to the execution without any extra steps.
- The second stack helped to construct the converted string and then store it. Pushing and popping, automatically, enabled algorithmic translation into code application without doing any extra steps.
- Although, printing out of stacks was simple, an extra loop had to be used because write method cannot take stack as an argument
- Python provides a nice dynamic application of stacks using list so there was no need to keep an eye on size
- This package has 3.py files, a __main__ and __init__ file.
- Since stack class is a separate module, it can be reused later or modified which provides flexibility. As an example, stack application can be changed without affecting other parts of package
- Similar, conversion part of package is in a separate module. This will help in future to include or exclude new operators or operands
- Finally, lab1 script which converts at file level, can be changed to accommodate any error handling at file level
- Overall, design of this package is very flexible.

Description of stack implementation and justification

-As stated above, python provides dynamic implementation. I used a list implementation but python provides for deque, lifoqueue or linked list

-This project was simple and just required reversing and storing so simple list implementation was fine.

-Basic list methods like append, pop, length were utilized

-Because of simplicity of lists, couple of easy methods like is empty or is_notempty were made. This helped in readibilty of code and made it more pythonic

-But, because of flexible nature of design, other implementation should be tried to check/compare space and time complexity

Were stacks a good choice for this lab?

Yes, for reasons stated above and given inputs were files of strings that needed to be tackled in linear fashion i.e each character had to be just popped or pushed without the need for insertion or search or replace. So, stacks provided a simple and fast data structure for the problem at hand.

Potential use of recursion

Yes, recursion can be applied and it will result in simplicity of code but might not be helpful in reducing time complexity or space complexity

Time and space efficiency

Popping and pushing functions of stack result in O(1) complexity. This package and problem just needed pushing and popping so it was a time efficient data structure. As far as space complexity goes, usage of two stacks might not be ideal to store all the characters of string because if the length of string increases, space might become an issue.

What did I learn?

-First and foremost, algorithmic steps to solve a problem don't match coding steps. Wasted lot of time because of this !
-Learned that modularizing code is so useful and easy to manage
-Learned how to make a package and excuse it from command line
-Learned how to construct a python project from pseudo code to command line execution

What would I do differently?

-Make sure my algorithmic pseudo code translates to python code from the word go
-I did not handle errors so would like to do a better job at it
-I assumed lot of things like all the strings are convertible, which were not. Hence wasted a lot of time which will cost me points because couldn't focus on error handling