ALGOS_1

PUSH SWAP

What is Push Swap:

Push swap is a stack sorting algorithm broken down into two concepts. The simple stack checker and the more complex stack sorting algorithm.

How to run PushSwap:

$>ARG="4 67 3 87 23";
Word-line count: [./push_swap $ARG | wc -l 6]

Check valid stack after sort: [./push_swap $ARG | ./checker $ARG]

Looks complex but all it describes is:
- A variable ARG will be fed into push_swap AND checker as argument
- push swap will generate a list of instructions that will print to terminal stdout(1) file descriptor
- the " | " (pipe) temporarily steals the file descriptor of stdout(1) and instead feeds these instructions into the checker as "arguments"
- Checker will then print OK or ERROR to stdout on the terminal

Important to note on input:

- Use your get next line and libft functions to trim and check if your input is even valid. (gnl, strsub , strtrim , strsplit , strchr, isdigit) are functions you can use just make sure they don't leak memory first.
- HIGHLY!!! recommend learning linked lists before this project as they are perfect for pushing and popping on stacks. If you write this in an array you need to code like 2-3X more code to do a simple pop or push or swap… sis.
- Get next line can be used to read input from stdin as well specifically for the checker.

Your output:

- Outputs are relatively straight forward and described well in the pdf.
- Push_swap: Outputs the instruction list to stdout with each instruction separated by a '/n'
- Checker prints OK or ERROR.

First steps:

1. Code the checker first because it’s going to be testing your push swap.
2. First step is to code a lexer and parser
    - Lexer: simply cuts out all the information/instructions that you need out of the args passed into your checker. (linked list or array of instruction strings sounds easy to make and check)
    - Parser: checks if the instructions/information you copied from the arguments is even correct if it isn't print ERROR.

3. Code your stack operations to manipulate the numbers on the checker stack must be able to do all the operations your push_swap program will print.
4. Code the actual check itself to see if your operations sorted the stack correctly.

Next steps:

1. Use the same parser and lexer your checker uses to parse and check the numbers as args.
2. Create a stack in the exact same way as your checker and use the functions your created to manipulate the stack in your checker to manipulate your stack in push swap.
3. Push swap will now be about coding an algorithm that can sort that stack using the stack functions you created in your checker and recording the instructions/operations you need to solve the puzzle in an array or list

Notes:

- Linked lists make (popping, pushing and reversing stacks quit simple although you need an in depth understanding of how they work before you can use them effectively.
- The algorithm you use to sort the stack needs to be so refined that its mad efficient cause if it's not as efficient as a tesla YOU WILL FAIL!!! No jokes it's in the marking sheet and its bloody difficult good luck.