Assignment 3

For this assignment, we’ll be building a simple stack-based 4-function calculator. **You must use monads in your program to be eligible for full credit.** Note that the Maybe and Either types are both monads.

A stack-based calculator uses postfix notation to avoid the need for parentheses; a sequence of operands and operators is always unambiguous. 2 + 3 is written as 2 3 +. The expression 2\*(3-1) + 2 is written as 3 1 – 2 \* 2 +.

An interactive calculator is even simpler. A stack of numbers is maintained. Initially, the stack is empty. We then get input from user and apply the following rules:

* If the input is a number: Put it onto the stack.
* If the input is an algebraic operator: Pop the appropriate number of operands off the stack, apply the operator, and put the result back on top of the stack. Our calculator will have the operators: ADD, SUB, MUL, DIV.
* If the input is another command: Carry out the command. Our calculator will have a few additional commands:
  + CLR: Clear the stack (removing everything).
  + SHOW: Show (display) the current state of the entire stack; does not change the stack.
  + TOP: Show (display) the top item on the stack without removing it.
  + SIZ: Returns the size of the stack, that is, the count of how many numbers are on it.
  + DUP: Duplicate the first item on the stack (put another copy of it onto the stack).
  + END: End program (stop getting input, clear stack).

You will need to write code for managing a list as a stack; that is, you’ll want push and pop functions. Your program should also allow the user to work interactively, taking single commands from the keyboard and carrying out the appropriate actions, or by taking a list of commands and carrying them out sequentially. (Hint: handling a single item, and handling a list that happens to have only 1 item in it, are equivalent.)

For example, if the top of the stack contains an integer *n*, this code will compute the sum of the integers from 1 to *n* using the standard formula *n*(*n+*1)/2: DUP 1 ADD MUL 2 DIV.

Your program needs to handle the following errors:

1. Division by 0: Returns a failure, displays error message.
2. Unknown command: Any command other than one listed above. Returns a failure, displays error message.
3. Too few items on stack. An operator was called that takes 2 operands, but there is only 1 item on the stack. Returns a failure, displays error message.

All of these errors should print an error message, show the current state of the stack, and end the program.

Design notes:

1. Start simple and build up.
2. Remember the design principles, and that you’re using Maybe or Either types. Take advantage of that.
3. Think functionally! For example, an operation should take a stack and an item (either to be pushed onto the stack, or a command), and return the stack after the operation is complete (if successful).