CSE 12 HW4 Getting Started --- Calc

Motivation

This PA is going to be an application of the Stack ADT that we covered last week. We will be implementing two algorithms that can be used to write a basic calculator.

Note:

This calculator is very basic, and does not perform input checking.

We don't try to validate parenthesis, or validate expressions, or input out of bound, or anything of that sort.

This means that expressions like

(3 + 5 will not work. The input

(just a close paren-thesis) will infinitely loop.·

As a side note, validating parenthesis could be done easily with a stack. Try thinking about how.

Side Note: Leetcode problem for validating (): https://leetcode.com/problems/valid-parentheses/

Function Explanations

Intopost

There are three kind of expressions, *postfix, prefix, infix*

- Postfix: 1 3 +
- Prefix: + 1 4
- Infix: 1 + 3

It is easier for computer to parse Postfix and Prefix, so we use this into post function to convert infix to postfix.

The algorithm is given to you in the comment block, but let's do some example here:

- Parse 3*5+4
  - 3 5 * 4 +
- Parse 3 + 4 * 5
  - 3 4 5 * +
- Parse (3 + 5) * 4
  - 3 5 + 4 *
- Parse 3! + 4

- 3!4+

Eval

The algorithm is again given to you. The somewhat unintuitive part is to use an array of functions.

See line 10, we declared this array of functions for you.

Long operator

You should do `functions[idx].operation(op1, op2)` to call the corresponding method. The `idx` is calculated in the set up word.

Note: you need to check the factorial case, because in that sensorial you don't need an op2, and thus you should not remove it from the stack.

Setup Word

This is the algorithm that, it takes in a character. You should find the index of the character that you are going to set up in the `operator` array. Then you need to do the following line to setup the word correctly. This is covered in discussion I believe.

```
1 | SIGNBIT | (index << 8) | character;
```

SIGNBIT changes the sign bit from 0 to 1 (makes the number negative).

(index << 8) shifts the index we found over by a full byte.

Lastly, we put the value of the character in ASCII in the final 8 bits.

This allows us to use `getIndex, getOperator, getPriority, isOperator, isNumber` that we defined at the top of the file.

The other functions you need to write is intuitive enough.