CPSC3383 Assignment 1.1

This is the final Assignment for class CPSC3383 Spring Semester

Motivation

The goal of this project is to create a calculator that will preform calculations in "Reverse Polish Notation", This is to demonstrate my ability design, implement and evaluate a computing-based solution to meet a given set of computing requirements

Getting Started

```
git clone https://github.com/skibaalex/cpsc3383.git

npm install / yarn
```

to run unit tests

```
yarn test
```

the UI is available to watch here

Description

In the assignment's instruction it states

```
The program should make use of standard tools for creating/handling syntax
analysis and parsing such as lex(1)3
and yacc(1)4
```

Since this project is implemented in TypeScript, Regex in combination with a `Stack` data structure will be used to handle and parse the syntax of the calculations. The assignment requires testing, Unit tests will be implemented using Jest Testing Library.

- Although not required this project will be developed in a TTD approach to improve my skills in the process.
- The project will be managed in a kanban board on the github website.
- At the end the project will be compiled into JS and served over the web using gh-pages.

Discussion

Table of contents

Syntax Parsing

The expected values to be received are any number in a string format. In addition there are 7 different mathematical operations that are allowed `+, -, /, *, sin, con, tan`, and expression grouping using `(, )`.

Some use cases to take into consideration are

- Trigonometric operations only take **one** parameter as an input while regular operations take **two**
- Any open parentheses has to have a corresponding closing parentheses.
- inside each parentheses has to accrue a valid operation.
- does the expression has to start with a parentheses?

In order to parse the syntax all the strings will be tested against a regular expression and then stored into a stack, which is there will be test for validity.

Stack Validity Check

In order to check if a stack is a valid RPN we will use the algorithm provided in this StackOverflow Answer, due to a lack of my ability to come up with another more robust alternative.

pseudo code for the algorithm

```
Set stack_size to 0;
For Each token In expression:
    Set stack_size to stack_size + 1 - valence(token)
    If stack_size <= 0: Report failure
If stack_size == 1: Report success
Else              : Report failure
```

Stack Calculation

At this point we have a stack full with syntactically valid, now we have to calculate that the contents of the Stack.

We will iterate over each individual element in out stack to see what action has to be taken Order of operation

1. [x] if the element is a number we will push it into an array of awaiting to be operated on.
2. [x] if the element is an operation we will make sure that we have enough elements in the awaiting elements array that is required to preform the operation then preform it and put the value back into out stack.
3. [x] if the element is an opening parentheses we will compute a new stack until the closing parentheses and calculate the value inside these parentheses in a recursive manner.
4. [x] if byt the end of this function the length of the answers stack is not exactly ***1*** something went wrong.

User Interface

For the user Interface we will use a simple html page using Bootstrap On the user interface page you can find a simple input field and a button that will calculate RPN.

References

1. TypeScript - TypeScript a programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language.
2. Regex - Regular Expression
3. Stack - "First in last out" data type
4. Jest - Standard JS testing library
5. TTD - Test Driven Development
6. gh-pages - Static web hosting service offered by GitHub
7. Reverse Polish Notation
8. Bootstrap - Make HTML simple