**Data Structures: Solving Problems with Stacks and Queues**

**Instructions**

In this challenge you will work on several algorithms that may be solved with stacks and queues.

**Existing Files**

| **File** | **Description** |
| --- | --- |
| src/postfix.js | Write your solution to the postfix evaluation problem in this file |
| src/binary.js | Write your solution to the binary numbers problem in this file |
| src/brackets.js | Write your solution to the bracket matching problem in this file |
| \_\_tests\_\_/\*.test.js | Contains tests for the various problems. You are welcome to look at these files but do not modify them. |
| src/lib/\* | The Stack, Queue and Node library code may be found in this folder. No need to modify these files. |

**Tasks**

**Evaluating postfix expressions**

Recall that an arithmetic expression may be written in postfix notation. The following are a few examples.

| **Infix** | **Postfix** |
| --- | --- |
| (2 + 3) \* 4 | 2 3 + 4 \* |
| (2 + (4 - 5) \* 3) | 2 4 5 - 3 \* + |
| 8 / (6 + 2) | 8 6 2 + / |

Given an arithmetic expression in postfix form, evaluate the expression and return the final value.

Assume:

* all operands a single digit numbers
* only the operators +, -, / and \* will be used
* all expressions are valid, no need to validate them
* only valid operations will be provided. That is no need to check for division by zero and other arithmetic anomalies.

A simple algorithm may be outlined as follows:

1. initialize an empty stack
2. remove all spaces from the expression
3. for each character in the expression
4. if the character is an operand push it unto the stack
5. if the character is an operator
6. pop two operands a and b from the stack
7. evaluate a op b where op is whichever operation is being considered
8. push the results back unto the stack
9. return the value on the stack

Implement your solution in the file named src/postfix.js.

**Generate binary numbers**

Given a number max, write an algorithm that generates all binary integers from 1 to max.

Examples:

Input: max = 2

Output: ["1", "10"]

Input: max = 5

Output: ["1", "10", "11", "100", "101"]

An algorithm that uses a queue to solve the problem is given below.

1. Initialize an empty queue
2. Enqueue the string "1". This represents binary number 1.
3. Initialize an empty array named result
4. Iterate from 1 to max and do:
   1. Dequeue a value from the queue
   2. Push the value into result
   3. Append a "0" to the value and enqueue it
   4. Append a "1" to the value and enqueue it
5. return result

Implement your solution in the file named src/binary.js.

**Extend Parentheses to other types of brackets**

Recall the algorithm that was used to determine if a given expression contained matching parentheses. It is repeated in pseudocode below:

1. Initialize a new empty stack
2. start a loop to iterate through each character in the expression
   1. if the current character is '('
      1. push it unto the stack
   2. else
      1. if the current character is ')'
         1. if the stack is not empty
            1. pop one item off the stack
         2. else
            1. return false
3. if the stack is empty
   1. return true
4. else
   1. return false

Extend the algorithm to recognize 3 different types of brackets: (), [], and {}. These must be correctly nested; "([)]" is incorrect, and should return false;

Implement your solution in the file named src/brackets.js.