JavaScript Number of Atoms

Challenge

Given a string `formula` representing a chemical formula, return the count of each atom.

The atomic element always starts with an uppercase character, then zero or more lowercase letters, representing the name.

One or more digits representing that element's count may follow if the count is greater than `1`. If the count is `1`, no digits will follow:

- For example, `'H2O'` and `'H2O2'` are possible, but `'H1O2'` is impossible.

Two formulas are concatenated together to produce another formula:

- For example, `'H2O2He3Mg4'` is also a formula.

A formula placed in parentheses, and a count (optionally added) is also a formula:

- For example, `'(H2O2)'` and `'(H2O2)3'` are formulas.

Return the count of all elements as a string in the following form: the first name (in sorted order), followed by its count (if that count is more than `1`), followed by the second name (in sorted order), followed by its count (if that count is more than `1`), and so on.

The test cases are generated so that all the values in the output fit in a 32-bit integer.

1st Example

```
Input: formula = 'H2O'
Output: 'H2O'
Explanation: The count of elements are {'H': 2, 'O': 1}.
```

2nd Example

```
Input: formula = 'Mg(OH)2'
Output: 'H2MgO2'
Explanation: The count of elements are {'H': 2, 'Mg': 1,
             'O': 2}.
```

3rd Example

```
Input: formula = 'K4(ON(SO3)2)2'
Output: 'K4N2O14S4'
Explanation: The count of elements are {'K': 4, 'N': 2,
             'O': 14, 'S': 4}.
```

Constraints

- `1 <= formula.length <= 1000`
- `formula` consists of English letters, digits, `'('`, and `')'`.
- `formula` is always valid.

Solution

```
const countOfAtoms = (formula) => {
    let stack = [],
        cur   = {},
        i     = 0;

    while (i < formula.length) {
        if (formula[i] === '(') {
            stack.push(cur);

            cur = {};

            i++;
        } else if (formula[i] === ')') {
            const [mult, newI] = readNextDigit(++i),
                          last = stack[stack.length - 1];

            i = newI;

            Object.keys(cur)
                  .forEach(key => cur[key] *= mult);

            Object.keys(last)
                  .forEach(key =>
                      last[key] = last[key] +
                                  (cur[key] ?? 0));

            Object.keys(cur).forEach(key => {
                if (last[key] === undefined) {
                    last[key] = cur[key];
                }
            });
```

Solution continues on next page...

```
            cur = stack.pop();
        } else {
            const [ele, newI] = readNextElement(i);

            i = newI;

            const [c, nI] = readNextDigit(i);

            i = nI;

            cur[ele] = (cur[ele] ?? 0) + c;
        }
    }

    return Object.entries(cur)
                 .sort((a,b) => a[0].localeCompare(b[0]))
                 .reduce((r, [key, val]) =>
                              r += `${key}${val === 1 ?
                              '' : val}`, '');

    function readNextElement(i) {
        if (!formula[i].match(/[A-Z]/)) {
            return null;
        }

        let res = formula[i++];

        while (formula[i]?.match(/[a-z]/)) {
            res += formula[i++];
        }

        return [res, i];
    }
```

Solution continues on next page...

```
    function readNextDigit(i) {
        if (!formula[i]?.match(/[0-9]/)) {
            return [1, i];
        }

        let res = 0;

        while (formula[i]?.match(/[0-9]/)) {
            res = res * 10 + +formula[i++];
        }

        return [res, i];
    }
};
```

Explanation

I've coded a function called `countOfAtoms` that takes in a chemical formula as a string and returns the count of each atom in the formula as a string. It uses a stack and two helper functions, `readNextElement` and `readNextDigit,` to parse the formula and keep track of the count of each atom.

The `countOfAtoms` function starts by initializing a stack, a `cur` object, and a counter `i` to `0`. It then enters a while loop that continues as long as `i` is less than the length of the formula.

Inside the loop, the function checks the current character in the formula. If it is an opening parenthesis, the current `cur` object is pushed onto the stack, and a new empty `cur` object is created.

If the current character is a closing parenthesis, the function reads

the next character(s) to determine the multiplier for the current `cur` object. It then retrieves the last object on the stack and merges it with the current `cur` object, taking into account the multiplier. Finally, the current `cur` object is set to the last object on the stack, which is popped off the stack.

If the current character is not a parenthesis, the function reads the next characters to determine the element and its count in the current `cur` object. If the element already exists in the `cur` object, its count is incremented; otherwise, it is added to the object with a count of `1`.

Once the entire formula has been parsed, the `cur` object is sorted alphabetically by element name. The function then concatenates the count for each element into a string.

The resulting string, representing the count of each atom in the formula, is returned as the output of the function.

In summary, the `countOfAtoms` function parses a chemical formula and counts the occurrences of each atom in the formula. It uses a stack and helper functions to handle parentheses, elements, and their respective counts. The function returns a string representation of the count of each atom.

Author: Trevor Morin