726. Number of Atoms

Problem Explanation

This problem asks to count the number of atoms in a given chemical formula. A chemical formula can consist of a single element with or without a count (e.g. "O" or "O2"), two elements concatenated together (e.g. "H2O"), or a formula placed in parentheses with an optional count (e.g. "(H2O2)" or "(H2O2)3").

The program should break down the formula into its constituent elements and their counts. The output should be a string with each element and its count (if more than 1) in alphabetical order.

For instance, if the input is "H2O", the program should return "H2O" because there are 2 Hydrogen atoms and 1 Oxygen atom. Similarly, if the input is "Mg(OH)2", the program should return "H2Mg02" because there are 2 Hydrogen atoms, 1 Magnesium atom and 2 Oxygen atoms.

Solution Approach

The problem can be proceeded by using recursion and a stack data structure. When we traverse the formula from the left, if we meet an uppercase character, we will parse the rest of the element and add the element and the count to a map. If we meet a '(', we will parse the subformula and multiply the count of all elements in the subformula by the count following the ')'.

Here is how we use recursion and stack. Whenever meet an '(' brace, we create a new stack top and start to trace a new map. When meet a ')' brace, we pop up the map from stack top and merge it into the lower level map. During the process, we need to keep track of the count of each element.

Python Solution

```
python
class Solution:
    def countOfAtoms(self, formula):
        stack = [collections.Counter()]
        i = 0

        while i < len(formula):
            if formula[i] == '(':
                stack.append(collections.Counter())
                i += 1
            elif formula[i] == ')':
                top = stack.pop()
                i += 1
                start = i
                while i < len(formula) and formula[i].isdigit():
                    i += 1
                multiplier = int(formula[start:i] or 1)
                for name, v in top.items():
                    stack[-1][name] += v * multiplier
            else:
                start = i
                i += 1
                while i < len(formula) and formula[i].islower():
                    i += 1
                name = formula[start:i]
                start = i
                while i < len(formula) and formula[i].isdigit():
                    i += 1
                count = int(formula[start:i] or 1)
                stack[-1][name] += count

        return "".join(name + (str(stack[-1][name]) if stack[-1][name] > 1 else '')
                       for name in sorted(stack[-1]))
```

Java Solution

```
java
class Solution {
    public String countOfAtoms(String formula) {
        int N = formula.length();
        Stack<Map<String, Integer>> stack = new Stack<>();
        stack.push(new TreeMap<>());

        for (int i = 0; i < N;) {
            if (formula.charAt(i) == '(') {
                stack.push(new TreeMap<>());
                i++;
            } else if (formula.charAt(i) == ')') {
                Map<String, Integer> top = stack.pop();
                int start = ++i, multiplicity = 1;
                while (i < N && Character.isDigit(formula.charAt(i))) i++;
                if (start< i) multiplicity = Integer.parseInt(formula.substring(start, i));
                for (String c: top.keySet()) {
                    int v = top.get(c);
                    stack.peek().put(c, stack.peek().getOrDefault(c, 0) + v * multiplicity);
                }
            } else {
                int start = i++;
                while (i < N && Character.isLowerCase(formula.charAt(i))) i++;
                String name = formula.substring(start, i);
                start = i;
                while (i < N && Character.isDigit(formula.charAt(i))) i++;
                int multiplicity = i > start ? Integer.parseInt(formula.substring(start, i)) : 1;
                stack.peek().put(name, stack.peek().getOrDefault(name, 0) + multiplicity);
            }
        }

        StringBuilder ans = new StringBuilder();
        for (String name: stack.peek().keySet()) {
            ans.append(name);
            int multiplicity = stack.peek().get(name);
            if (multiplicity > 1) ans.append("" + multiplicity);
        }
        return new String(ans);
    }
}
```

Javascript Solution

```
javascript
var countOfAtoms = function(formula) {
    let hash = {};
    let i = 0;
    let count = 1;
    let stack = [];
    let name = "";
    for (let char of formula) {
        let code = char.charCodeAt(0);
        if (65 <= code && code <= 90) {
            if (name) addName(hash, name, count);
            name = char;
            count = 1;
        } else if (97 <= code && code <= 122) {
            name += char;
        } else if ("0" <= char && char <= "9") {
            count = count * 10 + parseInt(char);
        } else if (char === "(") {
            let newObj = {};
            stack.push(hash, name, count]);
            hash = newObj;
            name = "";
            count = 1;
        } else if (char === ")") {
            if (name) addName(hash, name, count);
            let oldHash = stack.pop();
            let outsideCount = 0;
            while (i + 1 < formula.length && "0" <= formula[i + 1] && formula[i + 1] <= "9") {
                outsideCount = outsideCount * 10 + parseInt(formula[i + 1]);
                i++;
            };
            if (outsideCount === 0) outsideCount = 1;
            combine(oldHash[0], hash, name, oldHash[2] * outsideCount);
            hash = oldHash[0];
            name = oldHash[1];
            count = 1;
        } else { // char == '-'
            if (name) addName(hash, name, count);
            count = 1 - parseInt(char);
            name = "";
        };
        i++;
    }
    if (name) addName(hash, name, count);

    let result = "";
    let sortedKeys = Object.keys(hash).sort((a, b) => a.localeCompare(b));
    for (let sortedKey of sortedKeys) {
        result += sortedKey;
        if (hash[sortedKey] > 1) result += hash[sortedKey];
    };
    return result;
};

function addName(hash, name, count) {
    if (hash[name]) {
        hash[name] += count;
    } else {
        hash[name] = count;
    };
};

function combine(hash, oldHash, name, count) {
    Object.keys(oldHash).forEach(key => {
        hash[key] = hash[key] + oldHash[key] * count || oldHash[key] * count;
    })
};
```

C++ Solution

```
c++
class Solution {
public:
    string countOfAtoms(string formula) {
        int i = 0;
        map<string, int> counts = parse(formula, i);
        string ans;
        for (pair<string, int> p : counts) {
            ans += p.first;
            if (p.second > 1)
                ans += to_string(p.second);
        }
        return ans;
    }

private:
    map<string, int> parse(string& formula, int& i) {
        map<string, int> counts;
        while (i != formula.size() && formula[i] != ')') {
            if (formula[i] == '(') {
                i++; // Skip '('
                for (auto& p : parse(formula, i)) { counts[p.first] += p.second; }
            } else {
                int iStart = i++;
                while (i != formula.size() && islower(formula[i])) { i++; }
                string name = formula.substr(iStart, i - iStart);
                iStart = i;
                while (i != formula.size() && isdigit(formula[i])) { i++; }
                int count = iStart == i ? 1 : stoi(formula.substr(iStart, i - iStart));
                counts[name] += count;
            }
        }
        int iStart = ++i;
        while (i != formula.size() && isdigit(formula[i])) { i++; }
        if (iStart < i) {
            int multiplicity = stoi(formula.substr(iStart, i - iStart));
            for (auto& p : counts) { p.second *= multiplicity; }
        }
        return counts;
    }
};
```

C# Solution

```
csharp
public class Solution {
    public string CountOfAtoms(string formula) {
            int i = 0, len = formula.Length;
            Stack<Dictionary<string, int>> stack = new Stack<Dictionary<string, int>>();
            stack.Push(new Dictionary<string, int>());
            while (i < len) {
                    if (formula[i] == '(') {
                            stack.Push(new Dictionary<string, int>());
                            i++;
                    } else if (formula[i] == ')') {
                            var top = stack.Pop();
                            int start = ++i, num = 1;
                            while (i < len && Char.IsDigit(formula[i])) i++;
                            if (i > start) num = int.Parse(formula.Substring(start, i - start));
                            foreach (var kvp in top)
                                    if (stack.Peek().ContainsKey(kvp.Key)) stack.Peek()[kvp.Key] += kvp.Value * r
                                    else stack.Peek().Add(kvp.Key, kvp.Value * num);

                    } else {
                            int start = i++;
                            while (i < len && Char.IsLower(formula[i])) i++;
                            string name = formula.Substring(start, i - start);
                            start = i;
                            while (i < len && Char.IsDigit(formula[i])) i++;
                            int num = i > start ? int.Parse(formula.Substring(start, i - start)) : 1;
                            if (stack.Peek().ContainsKey(name)) stack.Peek()[name] += num;
                            else stack.Peek().Add(name, num);
                    }
            }
            var sb = new StringBuilder();
            var list = stack.Peek().Keys.ToList();
            list.Sort();
            foreach (string c in list)
                    sb.Append(c + (stack.Peek()[c] > 1 ? stack.Peek()[c].ToString() : ""));
            return sb.ToString();
    }
}
```

All these solutions involved employing a stack data structure for handling parentheses and a map for keeping storing the count of the elements. The stack would keep track of the corresponding map that contains the element and its count. If the program encounters an open parentheses or an upper case alphabet that signifies the beginning of a chemical symbol, it either begins by creating a new map or extracts the chemical symbol and its count and stores them into the map. For any closing parentheses, it intelligently reflects the multiplication factor (if any) associated with the chemical compounds enclosed by the parentheses on the corresponding counts, and merges them into a single map.

Any programming language with built-in or library-supported data structures like stack and map like Python, Java, JavaScript, C++ and C# can solve this problem with a similar approach. The differences lie in the syntax, and the way built-in methods are used. The Java solution, for instance, uses the 'peek()' function of the Stack data structure, which is used to look at the top element of the stack without deleting it. The JavaScript solution, on the other hand, extracts the character codes to determine whether a character is uppercase, lowercase, or a number. The C++ solution uses a map instead of counter to track each atom and its counts.

Solution Implementation

Python

Java

C++

TypeScript

Recommended Readings