772. Basic Calculator III

Leetcode Link

Problem Explanation

You are required to implement a basic calculator that evaluates simple expression strings. The expression string may contain open and closing parentheses, plus and minus signs, non-negative integers, multiplication and division operators, and empty spaces. For integer division, you should truncate towards zero. The expression is always valid and all intermediate results will be within the range of [-2147483648, 2147483647].

Here are some examples:

"1 + 1" should output: 2

" 6-4 / 2 " should output: 4

"2*(5+5*2)/3+(6/2+8)" should output: 21

"(2+6* 3+5- (3*14/7+2)*5)+3" should output: -12

We are not allowed to use the `eval` built-in library function for solving this problem.

Solution Approach

In this solution, a stack data structure is used to store operands and operators until they are calculated. The algorithm goes through the string expression and breaks it down into numbers and operators.

1. Create two stacks, one for numbers and the other for operators.
2. Loop through the string.
3. If the current character is a number, calculate the number and push it on the numbers stack.
4. If the current character is an operator or a parenthesis, do the following:
    - If it's an opening parenthesis, push it onto the operators stack.
    - If it's a closing parenthesis, perform calculations until an opening parenthesis is found and then pop the opening parenthesis from the stack.
    - If it's an operator '+', '-', '*', '/', check the precedence of the operator in stack with the current operator. If the operator in the stack has higher or equal precedence, perform calculation and then push the current operator in the stack.
5. When the string is exhausted, calculate the remaining expressions in the stack.
6. The top of the numbers stack will hold the final result.

Sample Step

Let's take the example of "6-4 /2".

- Initially, both stacks are empty.
- The first character '6' is a number, so push it in the number stack. number stack: [6].
- The next character '-' is an operator, so push it in the operator stack. operator stack: ['-'].
- The next characters '4', '/', and '2' similarly end up in the stack. number stack: [6, 4, 2], operator stack: ['-', '/'].
- We have finished parsing the string, pop items out of the stack and calculate. The '/' has higher precedence than '-', so pop out 4 and 2, calculate 4/2 with integer division and get 2, then push 2 back to number stack: [6,2]. The operator stack is now ['-'].
- Continue to calculate, pop out 6 and 2 from number stack, and '-' from operator stack, calculate 6 - 2 = 4. Now both stacks are empty and we return 4.

Python Solution

```python

python
class Solution:
    def calculate(self, s: str) -> int:
        def precedence(op):
            if op == '(' or op == ')':
                return 0
            elif op == '+' or op == '-':
                return 1
            else:
                return 2

        def apply_operand(op, b, a):
            if op == '+':
                return a + b
            elif op == '-':
                return a - b
            elif op == '*':
                return a * b
            else:
                return a // b

        nums = []
        ops = []

        i = 0
        while i < len(s):
            if s[i] == ' ':
                i += 1
            elif s[i] in '0123456789':
                num = 0
                while (i < len(s) and
                       s[i] in '0123456789'):
                    num = (num * 10 +
                           int(s[i]))
                    i += 1
                nums.append(num)
                i -= 1
            elif s[i] == '(':
                ops.append(s[i])
            elif s[i] == ')':
                while ops[-1] != '(':
                    num2 = nums.pop()
                    num1 = nums.pop()
                    op = ops.pop()
                    nums.append(apply_operand(op, num2, num1))
                ops.pop()
            else:
                while (ops and ops[-1] != '(' and
                       precedence(ops[-1]) >= precedence(s[i])):
                    num2 = nums.pop()
                    num1 = nums.pop()
                    op = ops.pop()
                    nums.append(apply_operand(op, num2, num1))
                ops.append(s[i])
            i += 1

        while ops:
            num2 = nums.pop()
            num1 = nums.pop()
            op = ops.pop()
            nums.append(apply_operand(op, num2, num1))

        return nums[0]
```

We can't offer a solution in other languages, as your current package only allows for assistance in python.## JavaScript Solution

In JavaScript, we can solve the problem with a similar approach to the Python solution. Here's how:

```

javascript
class Solution {
    precedence(op) {
        if (op === '(' || op === ')') return 0;
        else if (op === '+' || op === '-') return 1;
        else return 2;
    }

    apply_operand(op, a, b) {
        if (op === '+') return a + b;
        else if (op === '-') return a - b;
        else if (op === '*') return a * b;
        else return Math.floor(a / b);
    }

    calculate(s) {
        let nums = [];
        let ops = [];
        let numberRegex = /\d/;

        for (let i = 0; i < s.length; i++) {
            if (s[i] === ' ') continue;

            if (numberRegex.test(s[i])) {
                let num = 0;
                while (i < s.length && numberRegex.test(s[i])) {
                    num = num * 10 + Number(s[i]);
                    i++;
                }
                nums.push(num);
                i--;
            }
            else if (s[i] === '(') {
                ops.push(s[i]);
            }
            else if (s[i] === ')') {
                while (ops[ops.length - 1] !== '(') {
                    const b = nums.pop();
                    const a = nums.pop();
                    nums.push(this.apply_operand(ops.pop(), a, b));
                }
                ops.pop();
            } else {
                while (ops.length > 0 && ops[ops.length - 1] !== '(' && this.precedence(ops[ops.length - 1]) >= this.precedence(s[i
                    const b = nums.pop();
                    const a = nums.pop();
                    nums.push(this.apply_operand(ops.pop(), a, b));
                }
                ops.push(s[i]);
            }
        }

        while (ops.length > 0) {
            const b = nums.pop();
            const a = nums.pop();
            nums.push(this.apply_operand(ops.pop(), a, b));
        }

        return nums[0];
    }
}
```

Java Solution

Similarly, we can implement a solution in Java using stack.

```

java
import java.util.Stack;

public class Solution {
    private int precedence(char op) {
        if (op == '(' || op == ')') return 0;
        else if (op == '+' || op == '-') return 1;
        else return 2;
    }

    private int apply_operand(char op, int a, int b) {
        switch (op) {
            case '+':
                return a + b;
            case '-':
                return a - b;
            case '*':
                return a * b;
            default:
                return a / b;
        }
    }

    public int calculate(String s) {
        Stack<Integer> nums = new Stack<>();
        Stack<Character> ops = new Stack<>();

        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ' ') continue;

            if (Character.isDigit(s.charAt(i))) {
                int num = 0;
                while (i < s.length() && Character.isDigit(s.charAt(i))) {
                    num = num * 10 + s.charAt(i) - '0';
                    i++;
                }
                nums.push(num);
                i--;
            }
            else if (s.charAt(i) == '(') {
                ops.push(s.charAt(i));
            }
            else if (s.charAt(i) == ')') {
                while (ops.peek() != '(') {
                    int b = nums.pop();
                    int a = nums.pop();
                    nums.push(apply_operand(ops.pop(), a, b));
                }
                ops.pop();
            } else {
                while (!ops.empty() && ops.peek() != '(' && this.precedence(ops.peek()) >= this.precedence(s.charAt(i))) {
                    int b = nums.pop();
                    int a = nums.pop();
                    nums.push(this.apply_operand(ops.pop(), a, b));
                }
                ops.push(s.charAt(i));
            }
        }

        while (!ops.empty()) {
            int b = nums.pop();
            int a = nums.pop();
            nums.push(this.apply_operand(ops.pop(), a, b));
        }

        return nums.peek();
    }
}
```

These solutions in JavaScript and Java follow a similar pattern to the Python solution. They make use of two stacks for numbers and operators, and handle the precedence of operators, apply the operations, and handle the parentheses.