题目描述

LISP 语言唯一的语法就是括号要配对。

形如 (OP P1 P2 ...)，括号内元素由单个空格分割。

其中第一个元素 OP 为操作符，后续元素均为其参数，参数个数取决于操作符类型。

注意：

参数 P1, P2 也有可能是另外一个嵌套的 (OP P1 P2 ...)，当前 OP 类型为 add / sub / mul / div（全小写），分别代表整数的加减乘除法，简单起见，所有 OP 参数个数均为 2 。

举例：

- 输入：(mul 3 -7)输出：-21
- 输入：(add 1 2) 输出：3
- 输入：(sub (mul 2 4) (div 9 3)) 输出： 5
- 输入：(div 1 0) 输出：error

题目涉及数字均为整数，可能为负；

不考虑 32 位溢出翻转，计算过程中也不会发生 32 位溢出翻转，

除零错误时，输出 “error”，

除法运除不尽，向下取整，即 3/2 = 1

输入描述

输入为长度不超过512的字符串，用例保证了无语法错误

输出描述

输出计算结果或者“error”

用例

输入	(div 12 (sub 45 45))
输出	error
说明	45减45得0，12除以0为除零错误，输出error

输入	(add 1 (div -7 3))
输出	-2
说明	-7除以3向下取整得-3，1加-3得-2

题目解析

纯逻辑题，难点在于将括号中的片段截取出来，我的处理方案是，遍历输入的每一个字符，当遇到")"时，则在其前面必然存在一个"("，找到其前面第一个"("，然后截取"("和")"之间的内容（从栈中截取走），进行计算，将结果回填如栈中。

Java算法源码

```java
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println(getResult(sc.nextLine()));
  }

  public static String getResult(String s) {
    LinkedList<Character> stack = new LinkedList<>();
    LinkedList<Integer> leftIdx = new LinkedList<>();

    for (int i = 0; i < s.length(); i++) {
      char c = s.charAt(i);

      if (c == ')') {
        List<Character> fragment = stack.subList(leftIdx.removeLast(), stack.size());

        StringBuilder sb = new StringBuilder();
        for (int j = 1; j < fragment.size(); j++) sb.append(fragment.get(j));

        fragment.clear();

        String[] tmp = sb.toString().split(" ");

        String op = tmp[0];
        int p1 = Integer.parseInt(tmp[1]);
        int p2 = Integer.parseInt(tmp[2]);

        String res = operate(op, p1, p2);
        if ("error".equals(res)) {
          return "error";
        } else {
          for (int k = 0; k < res.length(); k++) stack.add(res.charAt(k));
        }
      } else if (c == '(') {
        leftIdx.add(stack.size());
        stack.add(c);
      } else {
        stack.add(c);
      }
    }
```

```java

    StringBuilder ans = new StringBuilder();
    for (Character c : stack) ans.append(c);
    return ans.toString();
  }

  public static String operate(String op, int p1, int p2) {
    switch (op) {
      case "add":
        return p1 + p2 + "";
      case "sub":
        return p1 - p2 + "";
      case "mul":
        return p1 * p2 + "";
      case "div":
        return p2 == 0 ? "error" : (int) Math.floor(p1 / (p2 + 0.0)) + "";
      default:
        return "error";
    }
  }
}
```

JS算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  console.log(getResult(line));
});

function getResult(s) {
  const stack = [];
  const leftIdx = [];

  for (let i = 0; i < s.length; i++) {
    if (s[i] === ")") {
      const fragment = stack.splice(leftIdx.pop());

      const [op, p1, p2] = fragment.slice(1).join("").split(" ");
      const res = operate(op, p1 - 0, p2 - 0);

      if (res === "error") return "error";
      else stack.push(...String(res));
    } else if (s[i] === "(") {
      leftIdx.push(stack.length);
      stack.push(s[i]);
    } else {
      stack.push(s[i]);
    }
  }

  return stack.join("");
}
```

```
36
37 function operate(op, p1, p2) {
38   switch (op) {
39     case "add":
40       return p1 + p2;
41     case "sub":
42       return p1 - p2;
43     case "mul":
44       return p1 * p2;
45     case "div":
46       return p2 === 0 ? "error" : Math.floor(p1 / p2);
47   }
48 }
```

Python算法源码

```
1  import math
2
3  # 输入获取
4  s = input()
5
6
7  def operate(op, p1, p2):
8      p1 = int(p1)
9      p2 = int(p2)
10     if op == "add":
11         return str(p1 + p2)
12     elif op == "sub":
13         return str(p1 - p2)
14     elif op == "mul":
15         return str(p1 * p2)
16     elif op == "div":
17         if p2 == 0:
18             return "error"
19         else:
20             return str(int(math.floor(p1 / p2)))
21     else:
22         return "error"
23
24
```

```python
# 算法入口
def getResult():
    stack = []
    leftIdx = []

    for i in range(len(s)):
        if s[i] == ')':
            l = leftIdx.pop()
            fragment = stack[l:]
            del stack[l:]

            op, p1, p2 = "".join(fragment[1:]).split(" ")

            res = operate(op, p1, p2)

            if res == "error":
                return "error"
            else:
                stack.extend(list(res))
        elif s[i] == '(':
            leftIdx.append(len(stack))
            stack.append(s[i])
        else:
            stack.append(s[i])

    return "".join(stack)


# 调用算法
print(getResult())
```