# 题目

给你一个字符串数组 tokens ，表示一个根据 逆波兰表示法 表示的算术表达式。

请你计算该表达式。返回一个表示表达式值的整数。

注意：

有效的算符为 '+'、'-'、'\*' 和 '/' 。

每个操作数（运算对象）都可以是一个整数或者另一个表达式。

两个整数之间的除法总是 向零截断 。

表达式中不含除零运算。

输入是一个根据逆波兰表示法表示的算术表达式。

答案及所有中间计算结果可以用 32 位 整数表示。

示例 1：

输入：tokens = ["2","1","+","3","\*"]

输出：9

解释：该算式转化为常见的中缀算术表达式为：((2 + 1) \* 3) = 9

示例 2：

输入：tokens = ["4","13","5","/","+"]

输出：6

解释：该算式转化为常见的中缀算术表达式为：(4 + (13 / 5)) = 6

示例 3：

输入：tokens = ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

输出：22

解释：该算式转化为常见的中缀算术表达式为：

((10 \* (6 / ((9 + 3) \* -11))) + 17) + 5

= ((10 \* (6 / (12 \* -11))) + 17) + 5

= ((10 \* (6 / -132)) + 17) + 5

= ((10 \* 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

= 22

提示：

1 <= tokens.length <= 104

tokens[i] 是一个算符（"+"、"-"、"\*" 或 "/"），或是在范围 [-200, 200] 内的一个整数

逆波兰表达式：

逆波兰表达式是一种后缀表达式，所谓后缀就是指算符写在后面。

平常使用的算式则是一种中缀表达式，如 ( 1 + 2 ) \* ( 3 + 4 ) 。

该算式的逆波兰表达式写法为 ( ( 1 2 + ) ( 3 4 + ) \* ) 。

逆波兰表达式主要有以下两个优点：

去掉括号后表达式无歧义，上式即便写成 1 2 + 3 4 + \* 也可以依据次序计算出正确结果。

适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中

注意：本题与主站 150 题相同：

<https://leetcode-cn.com/problems/evaluate-reverse-polish-notation/>

# 分析

## 方法一：栈

思路：

代码：

class Solution {

public:

int evalRPN(vector<string>& tokens) {

stack<int> numStack; // 用于存放数字的栈

// 遍历 tokens

for (string token : tokens) {

if (isOperator(token)) { // 如果是运算符

// 从栈中弹出两个数字

int num2 = numStack.top();

numStack.pop();

int num1 = numStack.top();

numStack.pop();

// 计算结果并将结果入栈

int result = calculate(num1, num2, token);

numStack.push(result);

} else { // 如果是数字

// 将数字转换为整数并入栈

numStack.push(stoi(token));

}

}

// 返回栈顶元素，即为计算结果

return numStack.top();

}

private:

// 判断是否为运算符

bool isOperator(string& token) {

return token == "+" || token == "-" || token == "\*" || token == "/";

}

// 根据运算符计算结果

int calculate(int num1, int num2, string& op) {

if (op == "+") {

return num1 + num2;

} else if (op == "-") {

return num1 - num2;

} else if (op == "\*") {

return num1 \* num2;

} else if (op == "/") {

return num1 / num2;

}

return 0;

}

};

复杂度分析：

时间复杂度：O(n)，其中n是数组tokens的长度。需要遍历数组tokens一次，计算逆波兰表达式的值。

空间复杂度：O(n)，其中n是数组tokens的长度。使用栈存储计算过程中的数，栈内元素个数不会超过逆波兰表达式的长度。