栈的应用

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.08

目录

01

出栈合法

TEACHING
COURSEWARE

TEACH

出栈合法

栈是一种 “先进后出（LIFO）” 的数据结构，其入栈和出栈操作的顺序存在严格限制。在已知入栈顺序的前提下，判断一个序列是否为合法的出栈顺序，是栈的经典应用问题。

出栈合法

3. 示例解析

入栈顺序：1, 2, 3, 4

待验证出栈顺序 1：2, 1, 4, 3（合法）

模拟过程：

压入 1 → 栈顶 1 ≠ 2 → 继续压入 2；

栈顶 2 == 2 → 弹出 2，j=1（指向 1）；

栈顶 1 == 1 → 弹出 1，j=2（指向 4）；

压入 3 → 栈顶 3 ≠ 4 → 压入 4；

栈顶 4 == 4 → 弹出 4，j=3（指向 3）；

栈顶 3 == 3 → 弹出 3，j=4（遍历完成）→ 合法。

待验证出栈顺序 2：3, 1, 2, 4（不合法）

模拟过程：

压入 1 → 压入 2 → 压入 3；

栈顶 3 == 3 → 弹出 3，j=1（指向 1）；

此时栈顶为 2 ≠ 1 → 入栈序列已结束，继续弹出 2 → 2 ≠ 1 → 栈空，j=1 未遍历完成 → 不合法。

02

后缀表达式

TEACHING COURSEWARE

TEACH

后缀表达式

后缀表达式是一种不依赖括号的表达式表示方法，运算符位于操作数之后，其计算顺序由元素位置直接决定，适合计算机通过栈高效处理。

核心概念

中缀表达式：日常使用的表达式（如 3 + 4 * 2），需考虑运算符优先级和括号。

后缀表达式：运算符在操作数之后（如 3 4 2 * +），无括号，运算顺序由左至右唯一确定。

后缀表达式

题目描述

复制 Markdown 展开 进入 IDE 模式

所谓后缀表达式是指这样的一个表达式：式中不再引用括号，运算符号放在两个运算对象之后，所有计算按运算符号出现的顺序，严格地由左而右新进行（不用考虑运算符的优先级）。

本题中运算符仅包含 +-*/。保证对于 / 运算除数不为 0。特别地，其中 / 运算的结果需要**向 0 取整**（即与 C++ `/` 运算的规则一致）。

如：`3*(5-2)+7` 对应的后缀表达式为：`3.5.2.-*7.+@`。在该式中，`@` 为表达式的结束符号。`.` 为操作数的结束符号。

输入格式

输入一行一个字符串 s，表示后缀表达式。

输出格式

输出一个整数，表示表达式的值。

输入输出样例

输入 #1	复制	输出 #1	复制
3. 5. 2. -*7. +@		16	

输入 #2	复制	输出 #2	复制
10. 28. 30. /*7. -@		-7	

说明/提示

数据保证，$1 \leq |s| \leq 50$，答案和计算过程中的每一个值的绝对值不超过 10^9。

后缀表达式

遍历输入字符串，分离出操作数和运算符；
遇到操作数时，将其转换为整数并压入栈；
遇到运算符时，从栈中弹出两个操作数（注意顺序：先弹出的是右操作数，后弹出的是左操作数），计算结果后将结果压回栈；
遍历结束后，栈中仅剩的元素即为表达式的结果。

后缀表达式

```cpp
#include <iostream>
#include <stack>
#include <string>
using namespace std;
int main() {
    string s;
    cin >> s;  // 读取后缀表达式
    stack<int> st;  // 存储操作数的栈
    int n = s.size();
    int i = 0;
    while (i < n) {
        if (s[i] == '@') {
            // 表达式结束，退出循环
            break;
        } else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/') {
            // 遇到运算符，弹出两个操作数计算
            int b = st.top();  // 右操作数（先弹出）
            st.pop();
            int a = st.top();  // 左操作数（后弹出）
            st.pop();
            int res;
            // 根据运算符计算结果
            if (s[i] == '+') {
                res = a + b;
            } else if (s[i] == '-') {
                res = a - b;
            } else if (s[i] == '*') {
                res = a * b;
            } else {  // '/' 向0取整（C++默认行为）
                res = a / b;
            }
            st.push(res);  // 结果压栈
            i++;  // 移动到下一个字符
```

后缀表达式


```
        } else {
            // 处理操作数（可能包含数字和负号）
            bool is_negative = false;
            if (s[i] == '-') {
                is_negative = true;
                i++;  // 跳过负号
            }
            // 提取数字部分（直到遇到'.'）
            int num = 0;
            while (s[i] != '.') {
                num = num * 10 + (s[i] - '0');  // 字符转数字
                i++;
            }
            // 处理负数
            if (is_negative) {
                num = -num;
            }
            st.push(num);  // 操作数压栈
            i++;  // 跳过 '.'
        }
    }

    // 栈顶元素即为结果
    cout << st.top() << endl;
    return 0;
}
```

后缀表达式

后缀表达式 `6 2 3 + - 3 8 2 / + * 2 ^ 3 +` 对应的中缀表达式是

- A. ((6-(2+3))*(3+8/2))^2+3
- B. 6-2+3*3+8/2^2+3
- C. (6-(2+3))*((3+8/2)^2)+3
- D. 6-((2+3)*(3+8/2))^2+3

表达式 a*(b+c)*d 的后缀表达式为( )，其中 * 和 + 是运算符。

- A. **a+bcd
- B. abc+*d*
- C. abc+d**
- D. *a*+bcd

CB