C++ 算法讲义：栈 (Stack)

1. 引言

- **问题引入**：
  想象你在餐厅洗盘子。客人们不断把用过的盘子堆在桌上，你每次洗的时候，是从这叠盘子的**最底端**抽，还是从**最顶端**拿？显然，你只能拿走最后放上去的那个。
- **价值**：
  这种“后来者居上”的逻辑在计算机世界无处不在：浏览器的“后退”按钮、编辑器的“撤销”功能(Ctrl+Z)、甚至复杂的表达式求值和函数调用，背后都是栈在支撑。它是掌握高级数据结构（如树、图）的敲门砖。
- **学习目标**：
  理解栈的 **LIFO (后进先出)** 原理，学会手动模拟栈以及使用 C++ STL 标准库中的 `stack`。

---

2. 核心思想

- **栈 (Stack)** 是一种遵循“后进先出”（Last In, First Out，简称 **LIFO**）原则的线性数据结构。
  - **入栈 (Push)**：把新元素放到最顶端。
  - **出栈 (Pop)**：把最顶端的元素移走。
  - **栈顶 (Top)**：当前最后进入、最上面的那个元素。

---

3. 代码实现

方法一：手动数组模拟

使用数组 `S[]` 存数据，用变量 `top_ptr` 指向栈顶下标（初始为 -1）。

```
int S[100];
int top_ptr = -1; // 初始为空

void push(int x) {
    top_ptr++;      // 腾出地方
    S[top_ptr] = x; // 放下盘子
}

void pop() {
    if (top_ptr != -1) top_ptr--; // 逻辑删除：指针下移，下次push会覆盖旧数据
}

int top() {
    return S[top_ptr]; // 返回当前的“顶端”
}
```

方法二：使用 C++ STL 标准库 (推荐用法)

实际开发中，我们直接调用 `#include <stack>`。

成员函数	含义	意义
`s.push(x)`	**入栈**	把元素 x 压到最顶端
`s.pop()`	**出栈**	把最顶端的元素删除
`s.top()`	**看栈顶**	返回最顶端的元素值
`s.empty()`	**判空**	判断栈中是否还有元素（返回 `true`/`false`）
`s.size()`	**查大小**	返回栈中元素的个数

```cpp
#include <iostream>
#include <stack> //栈的头文件
using namespace std;

int main() {
    stack<int> s; // 定义一个整数栈

    s.push(10);   // 入栈
    s.push(20);

    if (!s.empty()) { // 访问前务必检查是否为空!
        cout << "栈顶元素: " << s.top() << endl; // 输出 20
    }

    s.pop(); // 弹出 20
    return 0;
}
```

---

4. 经典应用

A. 括号匹配 (例 15-5)

括号匹配的本质是**最近匹配原则**：最后打开的左括号，必须最先被闭合。

核心逻辑拆解

我们可以把栈想象成一个“待完成任务清单”：

1. **遇到左括号** `(` `[` `{`：
   - **操作**：`s.push(ch)`。
   - **含义**：你许下一个承诺，后面必须有一个对应的“盖子”来找它。
2. **遇到右括号** `)` `]` `}`：
   - **第一步：检查是否有盒子。** 如果此时栈是空的（`s.empty()`），说明你手里拿着盖子却没盒子，直接判定 **非法**。

- **第二步：检查类型。** 看一眼栈顶 `s.top()`。如果我手里是 `]`，栈顶必须是 `[`。
  - **匹配**：`s.pop()`。说明这个盒子成功盖好了，从清单中划掉。
  - **不匹配**：比如 `( ]`，直接判定 **非法。**

当整个字符串读完后，**必须检查栈是否为空**：

- 如果 `s.empty()` 为真，说明所有承诺都兑现了 $\rightarrow$ **成功!**
- 如果栈里还剩东西，说明有的括号"开了口却没盖上" $\rightarrow$ **非法。**

B. 后缀表达式 (例 15-6)

后缀表达式（又称逆波兰表达式）最大的魅力在于：**它不需要括号，也不需要考虑加减乘除的优先级**，因为位置已经决定了一切。

步骤：

假设我们要计算：`3 4 2 * + 5 -`

1. **扫描到数字**：
   - 直接 `s.push(num)`。
   - *注意*：在代码实现时，如果输入是 `123.` 这种带点的格式，记得用循环把 `'1'`,`'2'`,`'3'` 拼接成整数 123 再入栈。
2. **扫描到运算符（如 `+`, `-`, `\*`, `/`）**：
   - **关键动作**：从栈中弹出两个数。
   - **注意顺序**：
     - 第一个弹出的是**右操作数** $b = s.top()$; `s.pop()`;
     - 第二个弹出的是**左操作数** $a = s.top()$; `s.pop()`;
   - **计算并将结果压回**：计算 a 运算符 b 的结果，然后 `s.push(result)`。
3. **扫描到终止符 `@`**：
   - 此时栈顶剩下的唯一一个数字，就是最终答案。

为什么第一个弹出的是右操作数?

看这个例子：`10 2 /` (即 $10 \div 2$)

- 栈里先入 10，后入 2。
- 弹出第一个数是 2（右边），第二个数是 10（左边）。
- 计算 $10/2 = 5$。如果顺序反了，变成 $2/10$ 就全错了!

---

5. 总结

- **要点回顾**：
  - 栈是 **LIFO**（后进先出）。
  - `stack` 的常用操作：`push` (入), `pop` (出), `top` (看栈顶), `empty` (判空)。
  - **注意**：`pop()` 和 `top()` 前一定要判断栈是否为空，否则程序会崩溃。