|  |
| --- |
| **《数据结构》**  **实验报告**  项目名称 实验二 栈的基本操作  专业班级 软工2304  学 号 8214230404  姓 名 王银波  **实验成绩：**  **批阅教师：**  2025年 4月 25日 |

# **实验二 实验二 栈的基本操作**

实验学时： 2 实验地点： 综合楼404 实验日期： 2025年4月25日

1. **需求分析**

**一、实验目的**

熟练掌握栈的基本操作：初始化栈、判栈为空、出栈、入栈等运算。

理解栈在数据处理中扮演的角色，运用栈结构实现具体的应用。

**二、实验内容**

栈是数据结构中一个非常重要的结构。表达式计算是掌握程序设计语言的重要部分之一，也是栈的应用的一个典型例子。本次实验的主要内容是利用栈的基本操作，设计一个程序，实现用算符优先法对算术表达式求值的过程。对本设计系统实现+、-、\*、/、%和乘方（^）运算。符合要求，同时提高自己的编程能力。实现算术表达式求值。

选作内容：提供对小数点的支持，允许输入的表达式中出现多位数字和小数点。

**三、实验要求**

1．认真阅读和掌握本实验的算法。

2．上机将本算法实现。

3．在程序的编写中尽量与专业的编程规范靠拢，系统代码采用结构化的编程方式，力求设计代码以及注释等规范，

4．保存和打印出程序的运行结果，并结合程序进行分析。

1. **概要设计**

根据实验要求进行分析，实验的主要内容是利用栈的基本操作，设计一个程序，实

现用算符优先法对算术表达式求值的过程。我们可以定义一个栈结构来进行设计实现。



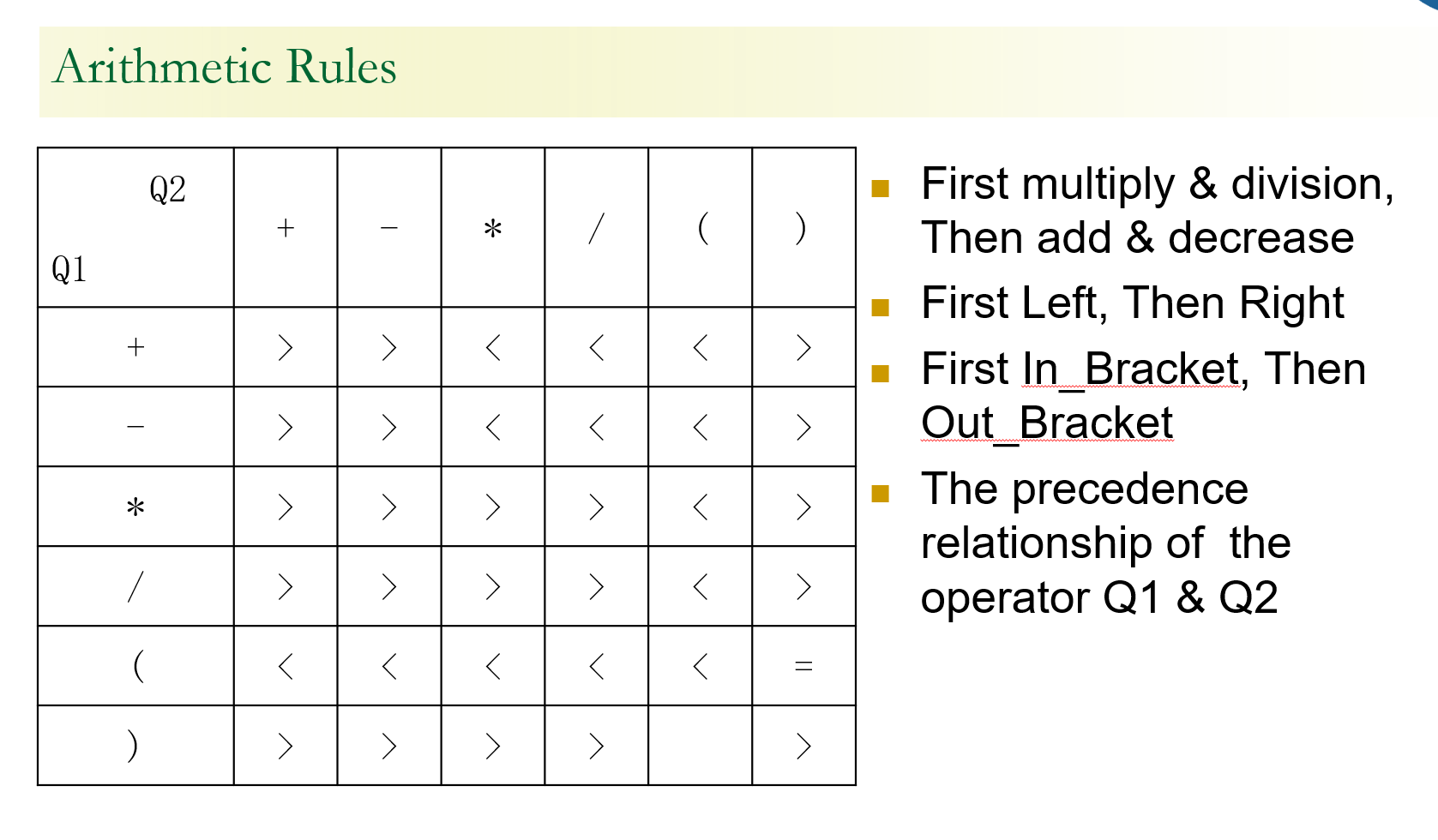
这样我们可以通过这么模板栈分别实现对于数字和操作符的存储。

**运算符优先级函数**：precedence(char op) 该函数根据运算符返回其优先级，优先级高的运算符先进行计算。

**运算执行函数**：applyOp(double a, double b, char op) 该函数根据运算符进行相应的数学运算。

在主程序中，根据算符优先文法的描述，如图所示：

我们可以对栈进行如下设计：



① 运算数：直接输出；

② 左括号：压入堆栈；

③ 右括号：将栈顶的运算符弹出并输出，直到遇到左括号（出栈，不输出）；

④ 运算符：

• 若优先级大于栈顶运算符时，则把它压栈；

• 若优先级小于等于栈顶运算符时，将栈顶运算符弹出并输出；再比较新的栈顶运算符，直到该运算符大于栈顶运算符优先级为止，然后将该运算符压栈；

⑤ 若各对象处理完毕，则把堆栈中存留的运算符一并输出。

由此我们得到了主程序的流程：

**输入：**程序从用户处读取一个算术表达式字符串，支持 + - \* / % ^ 运算符和小数

① 处理表达式：

② 使用两个栈：values 栈用于存放数值，ops 栈用于存放运算符；

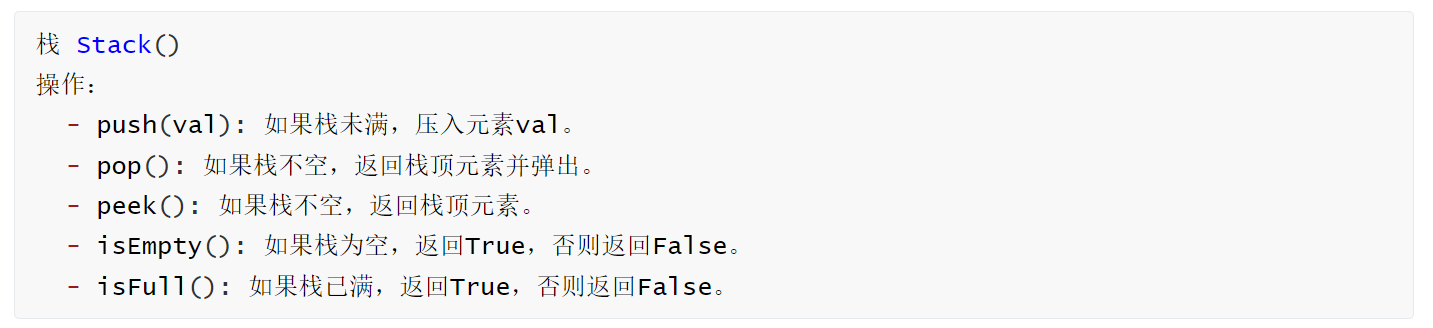
遍历表达式，逐个字符解析：

③ 若为数字或小数点，提取完整数字并压入 values 栈。若为运算符，依照运算符优先级进行处理，确保栈顶运算符优先执行。遇到括号时，根据括号的匹配情况进行栈的压入或弹出操作。在遍历结束后，继续处理栈中剩余的运算符，直到所有运算符都被执行。

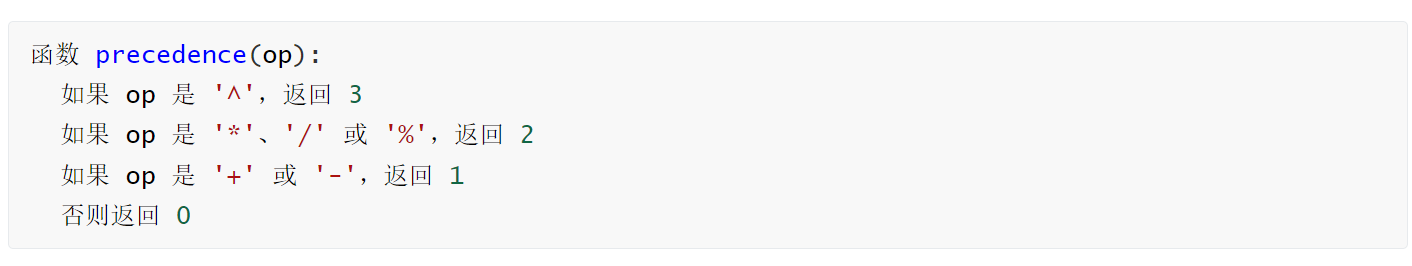
④ 输出：输出计算结果或异常信息。

**3．详细设计**

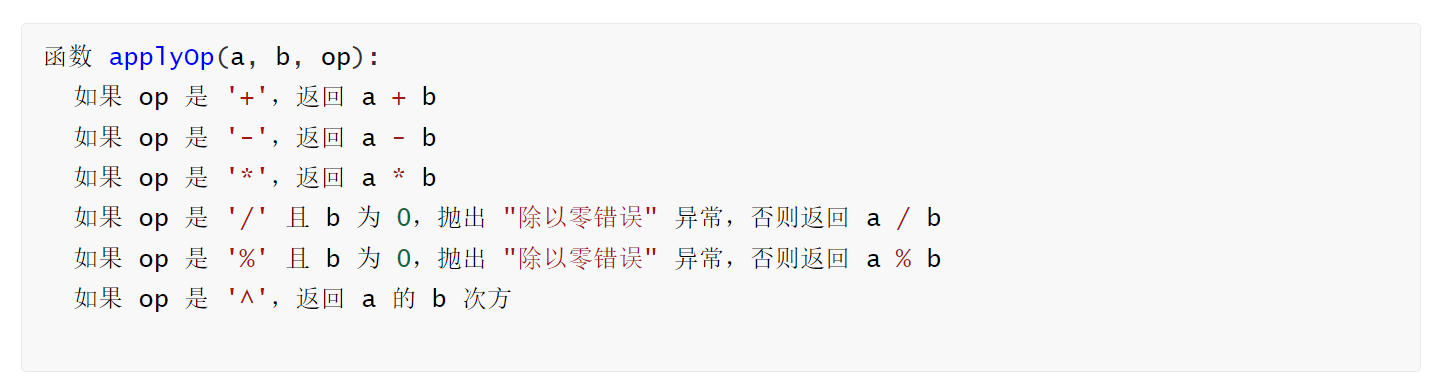
**栈的设计：**



**运算优先级：**



**运算处理：**



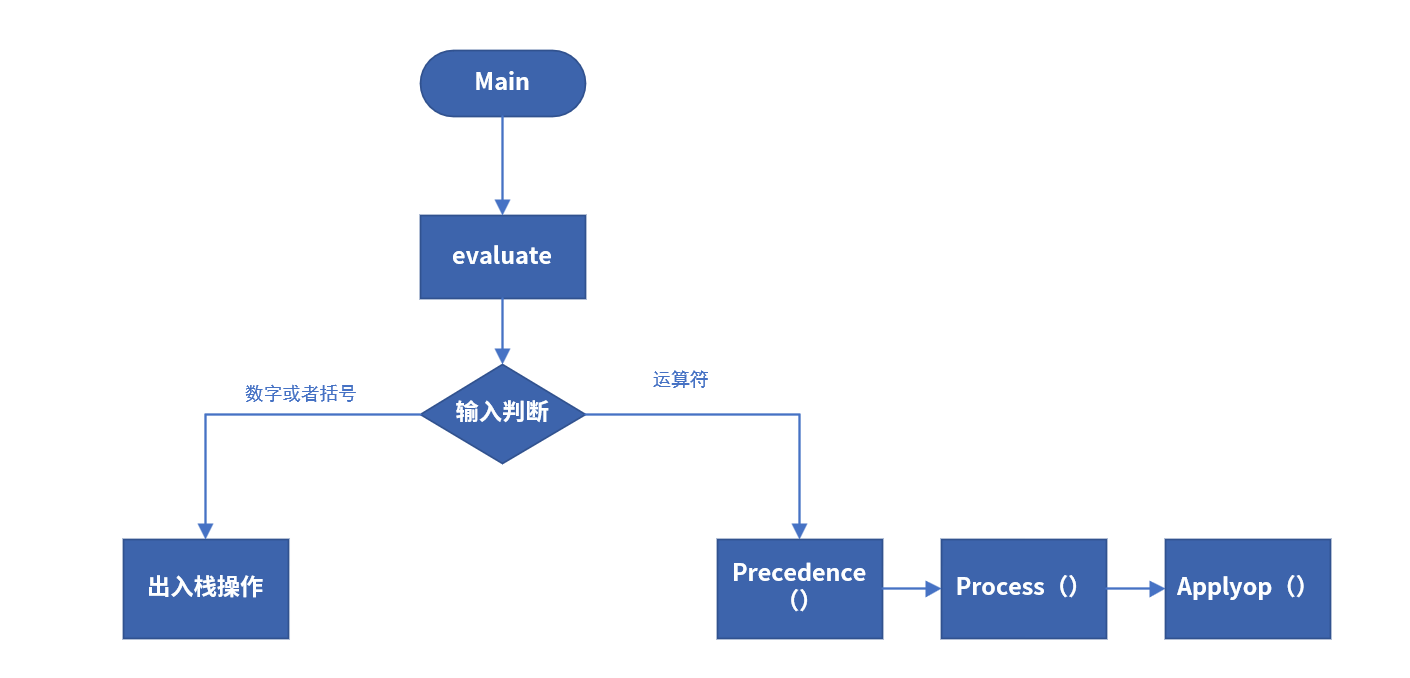
**数字计算：**



**压栈处理：**



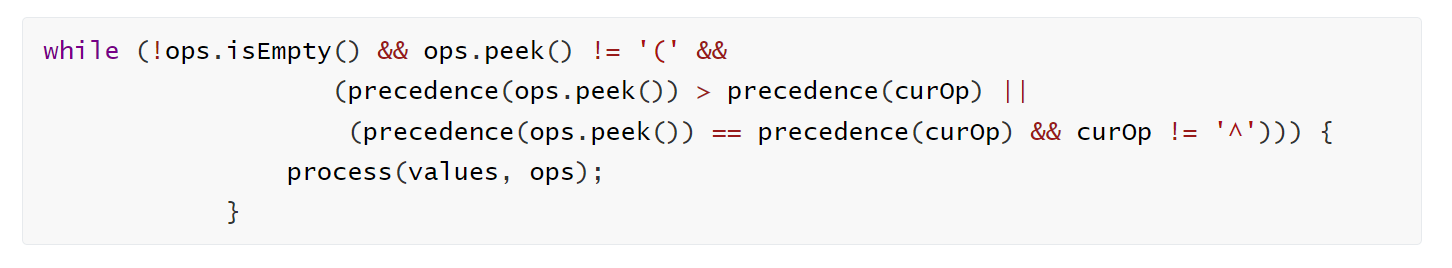
**函数调用关系图：**



**4．调试分析**

**遇到的问题：**

1. 首先就是关于对于小数点的读取，如果是2.5这样的数字需要如何进行处理，后来采用按照字符读取，最后转换成字符串再转换成浮点数的形式来进行解决。
2. 关于优先级的判断，乘方运算是左结合的运算，需要额外单独进行判断，不能和其他运算符直接比较优先级。



**设计与实现的回顾分析：**

本次实验主要是对栈的设计和应用来解决算符优先文法的问题，使用栈保存操作数和运算符，符合算符优先法的设计思路，采用手动处理字符串、数字、小数点，兼顾了多位数字和小数输入的需求。整体模块划分合理，evaluate 函数控制主流程，process 封装了运算执行，applyOp 封装了运算逻辑，栈操作独立，清晰易维护。

**算法的时空分析：**

**时间复杂度：**

遍历表达式每个字符，单次扫描；每次遇到运算符最多进行一次或多次 process 操作（栈顶元素弹出并计算），总体时间复杂度为 O(n)，其中 n 是输入字符串的长度。

**空间复杂度：**

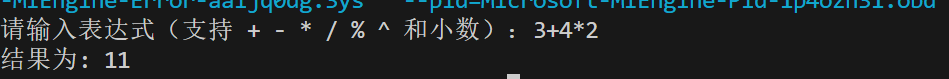
使用了两个栈（values 和 ops），每个栈最多存储 n/2 个元素（假设每两个字符一组），总体空间复杂度为 O(n)。

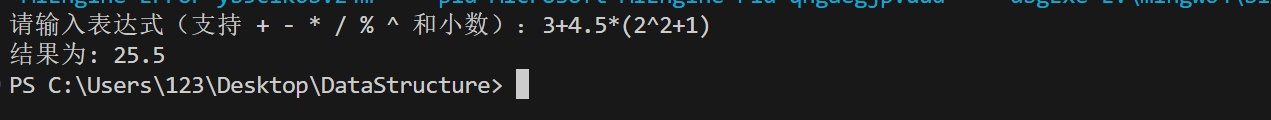
1. **用户使用说明**

用户输入表达式（支持 + - \* / % ^ 和小数），然后程序会直接输出结果。

1. **测试结果**

**如图所示：**





**7．附录**

源代码：

#include **<iostream>**
  
#include **<cstring>**
  
#include **<cmath>**
  
#include **<cctype>**
  
#include **<stdexcept>**
  
**using** **namespace** std;
  
  
const int MAXSIZE = 1000;
  
  
*// ===== 自定义栈结构 =====*  
**template**<**typename** T>
  
**struct** Stack {
  
 T data[MAXSIZE];
  
 int top = -1;
  
  
 bool isEmpty() { **return** top == -1; }
  
 bool isFull() { **return** top == MAXSIZE - 1; }
  
  
 void push(T val) {
  
 **if** (isFull()) **throw** runtime\_error("栈溢出");
  
 data[++top] = val;
  
 }
  
  
 T pop() {
  
 **if** (isEmpty()) **throw** runtime\_error("栈为空");
  
 **return** data[top--];
  
 }
  
  
 T peek() {
  
 **if** (isEmpty()) **throw** runtime\_error("栈为空");
  
 **return** data[top];
  
 }
  
};
  
  
*// ===== 运算符优先级 =====*  
int precedence(char op) {
  
 **if** (op == '^') **return** 3;
  
 **if** (op == '\*' || op == '/' || op == '%') **return** 2;
  
 **if** (op == '+' || op == '-') **return** 1;
  
 **return** 0;
  
}
  
  
*// ===== 运算执行 =====*  
double applyOp(double a, double b, char op) {
  
 **switch** (op) {
  
 **case** '+': **return** a + b;
  
 **case** '-': **return** a - b;
  
 **case** '\*': **return** a \* b;
  
 **case** '/':
  
 **if** (b == 0) **throw** runtime\_error("除以零错误");
  
 **return** a / b;
  
 **case** '%':
  
 **if** (b == 0) **throw** runtime\_error("除以零错误");
  
 **return** fmod(a, b);
  
 **case** '^': **return** pow(a, b);
  
 **default**: **throw** runtime\_error("未知运算符");
  
 }
  
}
  
  
*// ===== 运算处理函数 =====*  
void process(Stack<double>& values, Stack<char>& ops) {
  
 **if** (values.top < 1) **throw** runtime\_error("表达式格式错误");
  
 double b = values.pop();
  
 double a = values.pop();
  
 char op = ops.pop();
  
 values.push(applyOp(a, b, op));
  
}
  
  
*// ===== 表达式求值主函数 =====*  
double evaluate(const string& expr) {
  
 Stack<double> values;
  
 Stack<char> ops;
  
 int i = 0;
  
  
 **while** (i < expr.length()) {
  
 **if** (isspace(expr[i])) {
  
 i++;
  
 **continue**;
  
 }
  
  
 *// 数字或小数点*  
 **if** (isdigit(expr[i]) || expr[i] == '.') {
  
 string numStr;
  
 **while** (i < expr.length() && (isdigit(expr[i]) || expr[i] == '.')) {
  
 numStr += expr[i++];
  
 }
  
 values.push(stod(numStr));
  
 }
  
 *// 左括号*  
 **else** **if** (expr[i] == '(') {
  
 ops.push(expr[i]);
  
 i++;
  
 }
  
 *// 右括号*  
 **else** **if** (expr[i] == ')') {
  
 **while** (!ops.isEmpty() && ops.peek() != '(') {
  
 process(values, ops);
  
 }
  
 **if** (!ops.isEmpty() && ops.peek() == '(')
  
 ops.pop();
  
 else
  
 **throw** runtime\_error("括号不匹配");
  
 i++;
  
 }
  
 *// 运算符*  
 **else** **if** (strchr("+-\*/%^", expr[i])) {
  
 char curOp = expr[i];
  
 **while** (!ops.isEmpty() && ops.peek() != '(' &&
  
 (precedence(ops.peek()) > precedence(curOp) ||
  
 (precedence(ops.peek()) == precedence(curOp) && curOp != '^'))) {
  
 process(values, ops);
  
 }
  
 ops.push(curOp);
  
 i++;
  
 }
  
 **else** {
  
 **throw** runtime\_error(string("非法字符：") + expr[i]);
  
 }
  
 }
  
  
 **while** (!ops.isEmpty()) {
  
 process(values, ops);
  
 }
  
  
 **if** (values.top != 0) **throw** runtime\_error("表达式格式不正确");
  
  
 **return** values.pop();
  
}
  
  
*// ===== 主函数测试 =====*  
int main() {
  
 string expr;
  
 cout << "请输入表达式（支持 + - \* / % ^ 和小数）：";
  
 getline(cin, expr);
  
  
 **try** {
  
 double result = evaluate(expr);
  
 cout << "结果为: " << result << endl;
  
 } **catch** (exception& e) {
  
 cout << "错误: " << e.what() << endl;
  
 }
  
  
 **return** 0;
  
}