**实验报告一**

1. **实验内容**

中缀表达式求解

1. **数据结构设计与思路分析**
2. 需要用两个栈s1和s2，s1装操作数，s2装操作符
3. 扫描到左括号直接入栈s2，如果栈顶元素是左括号，运算符无条件入栈
4. 扫描到操作数直接入栈s1
5. 扫描到右括号时将s2中从栈顶元素到左括号之间的依次出栈s2,运算得到相应的结果后入栈s2，最终要将左括号舍弃
6. 扫描到的运算符元素的优先级大于s2栈顶元素的优先级，扫描到的运算符直接入s2；反之（小于等于）则将s2栈顶元素出栈，计算得出结果并入栈s1；然后再将扫描到的运算符元素与栈顶元素比较；合适再入栈
7. 具体运算方法：运算时要对s1进行两次出栈操作，先出栈的作为右操作数，后出栈的为左操作数

注：本题在实现时与一般中缀表达式不同的地方在于其操作数含有一位以上的整数以及浮点数，因此我们从字符串中获取操作数时需要对字符串进行解析操作，从而成功分离操作数。

1. **代码呈现**

源程序见cpp程序

**此处着重呈现解析字符串的过程：**

double getOpnd(string str, int &i)

{

double data = 0.0;

while (i < str.size())

{

if (!isdigit(str[i]))

break;

else

{

while (isdigit(str[i]))

{

data = data \* 10 + str[i] - '0';

i++;

}

if (str[i] == '.')

{

i++; //跳过小数点

double num = 0.0;

int k = 0;

while (isdigit(str[i]))

{

--k;

num = pow(10, k);

data = data + (str[i] - '0') \* num;

i++;

}

}

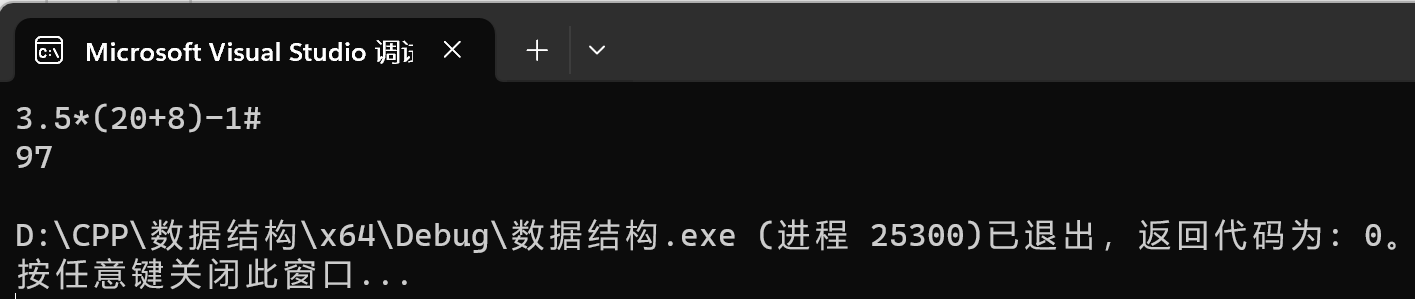
}

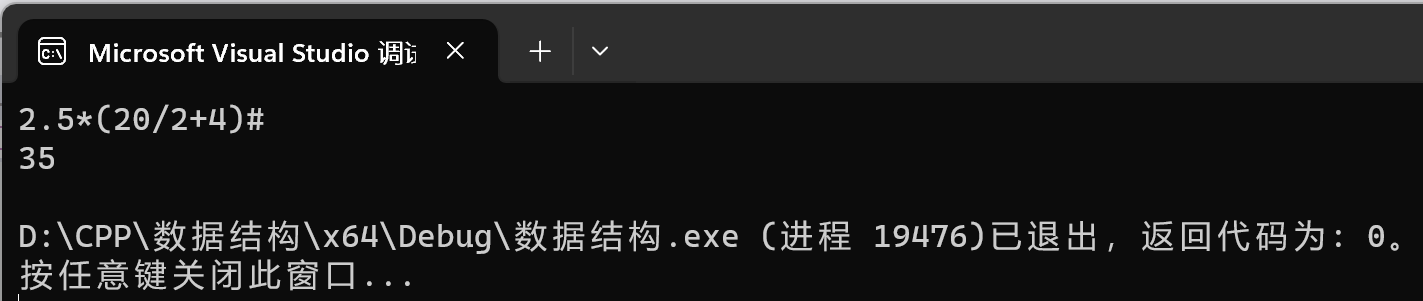
}

return data;

}

1. **测试结果**

****

****

测试用例选取的表达式中均含有多位整数和浮点数，符合题目要求

由上述测试结果可见：所编写的程序结果是正确的。

1. **分析小结**

本题是学习栈这一数据结构时必须亲身实践的一道题，在学会操作数只有一位的最基本的情况的思路分析与代码编写后，要学会去观察本题的特殊之处：操作数的不同。这也教会了我们要结合一个问题的具体情况进行具体分析，不能照本宣科。

1. **附加题思考**
2. **后缀表达式（逆波兰表达式）求解**
3. 逆波兰表达式

思路分析：遇到操作数就压栈，遇到运算符就进行出两次栈，先出的为操作符的右数，后出的为操作符的左数，然后将运算的结果进行压栈，最后留在栈里的数据就是最终的结果。

private:

    stack<int> st;

public:

    int evalRPN(vector<string>& tokens) {

        for(int i=0;i<tokens.size();i++){

//这里是一个逆向思维：不好判断是否为数字字符串，则判断是否为运算符

            if(tokens[i]!="+"&&tokens[i]!="-"&&tokens[i]!="\*"&&tokens[i]!="/"){

                st.push(stoll(tokens[i]));      //stoll将字符转化为数字

            }

            else{

                int right\_val=st.top();

                st.pop();

                int left\_val=st.top();

                st.pop();

                int res;

                if(tokens[i]=="+"){

                    res=left\_val+right\_val;

                }

                else if(tokens[i]=="-"){

                    res=left\_val-right\_val;

                }

                else if(tokens[i]=="\*"){

                    res=left\_val\*right\_val;

                }

                else if(tokens[i]=="/"){

                    res=left\_val/right\_val;

                }

                st.push(res);

            }

        }

        return st.top();

    }

1. **括号的匹配**

思路分析：括号匹配不成功共有三种情况

第一种情况：遍历完了字符串，但是栈不为空，说明左括号多了

第二种情况：访问字符串的过程中栈已经空了，说明右括号多了

第三种情况：访问过程中栈不为空，但是左右括号不匹配

 bool isValid(string s) {

        stack<char> st;

        for(int i=0;i<s.size();i++){

            if(s[i]=='('||s[i]=='['||s[i]=='{'){

                st.push(s[i]);

            }

            else{ //说明遍历到的是右括号

//第二种情况//如果遍历到右括号的时候栈为空 说明栈里面没有左括号 则直接返回否定

                if(st.empty()){

                    return false;

                }

//第三种情况//

                char x=st.top();

                if((x=='('&&s[i]!=')')||(x=='{'&&s[i]!='}')||(x=='['&&s[i]!=']')){

                    return false;

                }

                st.pop();

            }

        }

//第一种情况

        return st.empty();

    }