# 1 需求

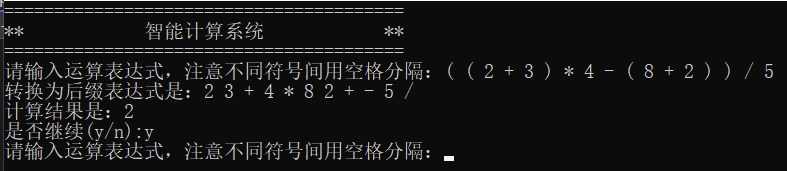
## 1.1 项目背景

算数表达式有前缀表示法，中缀表示法和后缀表示法等形式。日常使用的算术表达式是采用中缀表示法，即二元运算符位于两个运算数中间。中缀表达式作为通用的算术或逻辑公式表示方法，非常符合人的思维习惯，但对于计算机而言却并不是最高效的处理方式。故本项目拟构建一个智能计算系统，将用户输入的前缀表达式转换为后缀表达式并求出最终结果，该计算器功能要求如下：

1. 程序支持对所有输入的表达式作简单的判断，如表达式有错，能给出适当的提示。
2. 程序支持整数与浮点数的运算
3. 程序支持的运算符包括：' + ', ' - ', ' \* ', ' / ' ,' ( ' ,' ) '
4. 程序支持的运算操作包括：加、减、乘、除
5. 程序支持处理单目运算符：+或-
6. 程序最终输出表达式转换后的结果与计算结果
7. 输入说明：输入在一行中给出以空格分隔不同对象的中缀表达式，可包含+, -, \*, /, -, \*, /以及左右括号，表达式不超过20个字符（不包括空格）。
8. 输出说明：在一行中输出转换后的后缀表达式，要求不同对象（运算数，运算符号）之间以空格分隔，但是结尾不得有多余空格

## 1.2 功能分析

本项目功能可主要分为三项：输入错误处理、转换中缀表达式为后缀表达式和后缀表达式求解。中缀表达式转后缀表达式可用栈的数据结构实现，先将中缀表达式的字符保存在暂存栈中，根据字符优先级决定是否立即存入后缀表达式的结果数组。后缀表达式的计算只需依次访问后缀表达式字符数组并压栈，读到运算符便将栈顶的两个运算数弹出进行运算，最后栈中剩下的数字即为最终结果。若最后栈中元素个数不为1，则用户输入的表达式有误，程序给出提示并让用户再次输入。

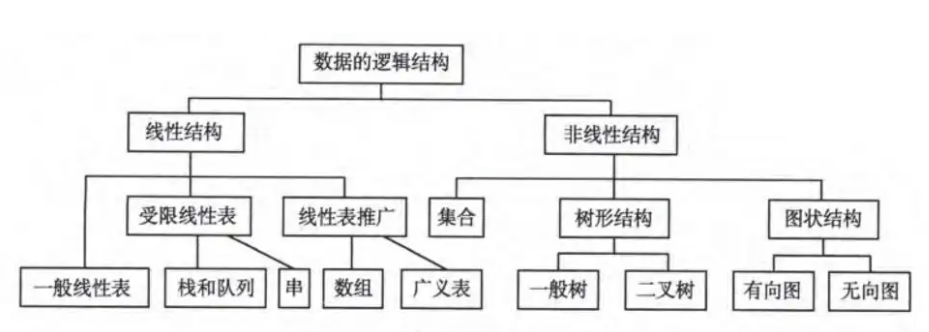


# 2 设计

## 2.1 数据结构设计

### 2.1.1 逻辑结构设计

本项目的核心功能是完成中缀表达式的转换与结果计算。由于每个字符信息相对独立，各个元素之间不存在明显的偏序关系或层次关系，也不存在“一对多”与“多对一”的映射，所以可以采用线性结构进行存储。



从1.2功能分析可知中缀表达式的转换与求值过程对元素的读取和插入是满足“先进后出”规则的，即栈的“Last In First Out”形式，因此可以建立受限线性表栈对中缀表达式的转换结果进行存储。而程序对用户输入的字符只需做读取操作，故使用数组存储即可。

### 2.1.2 存储结构设计

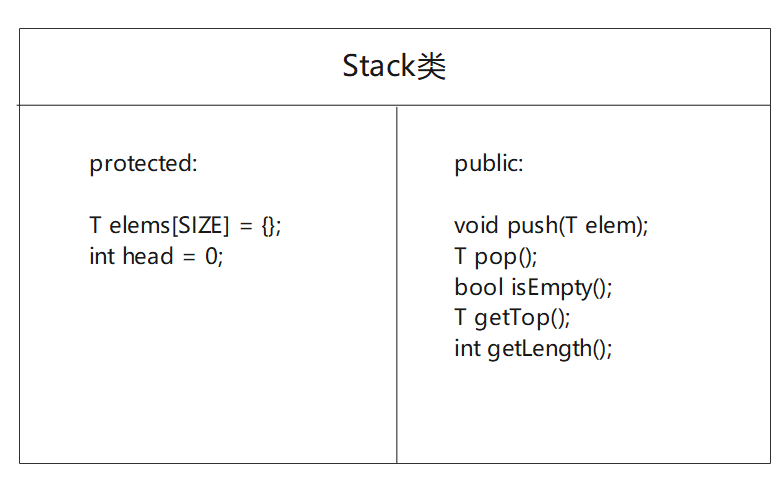
本项目对字符元素的操作主要为插入与删除，每次操作均在栈顶，若使用顺序存储方式实现栈的push与pop操作时间复杂度均为O(1)，而链式存储为O(n),，且顺序存储的空间开销较小，故本项目选用顺序存储方式。

## 2.2 类设计

本项目共定义了两个类，分别是栈Stack和计算器calculator类。其中Stack类主要用于暂存中缀表达式转换得到的中间结果以及进行算术运算，calculator类则用于实现项目主体功能，比如对用户输入进行错误处理、对用户输入进行运算、打印最终结果等。为了提高代码的重用性，项目在定义Stack类时采用了类模板的形式以适应不同的数据类型。

### 2.2.1 Stack类

项目通过Stack类实现了栈的数据结构，Stack类的protected属性有两个，一是存储数组elem，二是栈顶指针head，指向当前可以访问或增加元素的位置。Stack类的public属性主要存放栈的各项操作，包括元素入栈、元素出栈、判断栈是否为空、读取栈顶元素、获取栈元素个数等。



//顺序栈的模板类

**template** <**class** T>

**class** Stack {

**protected**:

    T elems[SIZE] = {};    //存储数组

**int** head = 0;    //栈顶指针

**public**:

**void** push(T elem);    //元素入栈

    T pop();    //元素出栈

**bool** isEmpty();    //判断栈是否为空

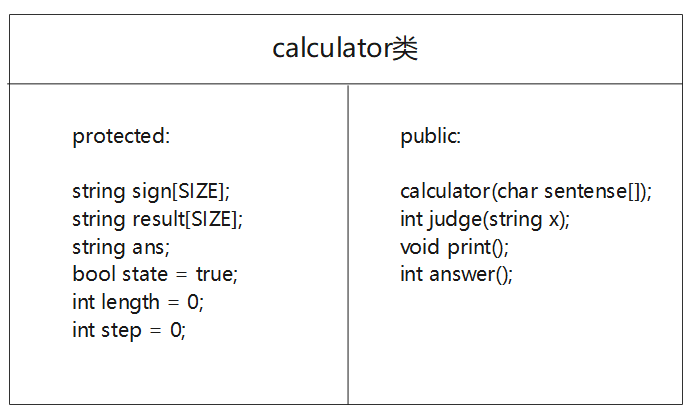
    T getTop();    //读取栈顶元素

**int** getLength();    //获取栈元素个数

};

### 2.2.2 calculator类

计算器类calculator实现了项目的主体功能，calculator类的protected属性包括输入字符数组sign[]及其长度length、后缀表达式字符数组result[]及其长度step、表达式计算结果ans以及判断用户输入是否正确的标志state。public属性用于存放操作方法，包括构造函数、查找字符优先级函数、转换函数与求解函数等。



//计算器类

**class** calculator {

**protected**:

    string sign[SIZE];    //输入字符

    string result[SIZE];    //后缀表达式字符

    string ans;    //表达式计算结果

**bool** state = **true**;    //判断是否输入错误

**int** length = 0;    //输入字符个数

**int** step = 0;    //后缀表达式字符个数

**public**:

    calculator(**char** sentense[]);    //构造函数

**int** judge(string x);    //查找字符优先级

**void** print();    //转换为后缀表达式并打印结果

**int** answer();    //求出表达式的计算结果

};

# 

# 3 实现

## 3.1 中缀表达式转后缀表达式功能实现

### 3.1.1 中缀表达式转后缀表达式功能实现思路

中缀表达式转后缀表达式主要通过栈的数据结构实现，转换过程中依次读取用户输入字符数组的元素并压栈，根据栈顶字符优先级与进栈字符优先级的大小关系决定是否将该元素保存至后缀表达式的结果数组中，具体流程如下：

1.对中缀表达式数组从左到右进行遍历。

2.若当前字符为运算数,直接输出。

3.若当前字符为左括号,直接压入堆栈。

4.若当前字符为右括号,意味着括号已结束，不断弹出栈顶运算符并输出直到遇到左括号。

5.若当前字符为运算符,将该运算符与栈顶运算符进行比较：

如果优先级高于栈顶运算符则压入堆栈；

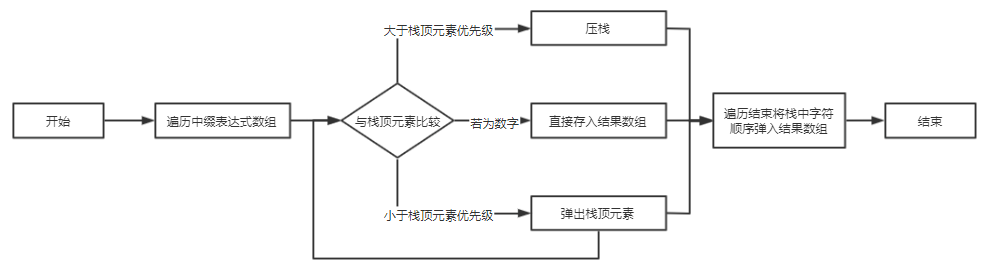
如果优先级低于等于栈顶运算符则将栈顶运算符弹出并输出,然后比较新的栈顶运算符；

直到优先级大于栈顶运算符或者栈空,再将该运算符入栈。

如果对象处理完毕,则按顺序弹出并输出栈中所有运算符。



### 3.1.2 中缀表达式转后缀表达式功能流程图



### 3.1.3 中缀表达式转后缀表达式功能实现代码

//判断字符优先级

**int** calculator::judge(string x) {

**int** num = 0;

**for** (**int** i = 0; i < x.length(); i++) {

**if** (i == 0 && x[i] == '-')

**continue**;

**if** ((x[i] < '0' || x[i] > '9') && x[i] != '.')

**break**;

**if** (x[i] == '.') {

            num += 1;

**if** (num > 1)

**break**;

        }

**if** (i == x.length() - 1)

**return** 0;    //字符为数字

    }

**if** (x.length() == 1) {

**for** (**int** i = 0; i < 6; i++) {

**if** (x[0] == ope[i]) {

**if** (x[0] == '(')

**return** 4;    //字符为'('

**if** (x[0] == '\*' || x[0] == '/')

**return** 3;    //字符为'\*'或'/'

**if** (x[0] == '+' || x[0] == '-')

**return** 2;    //字符为'+'或'-'

**if** (x[0] == ')')

**return** 1;    //字符为')'

            }

        }

    }

**return** -1;    //报错返回-1

}

//求出后缀表达式并打印结果

**void** calculator::print() {

    Stack<string> temp;

**for** (**int** i = 0; i < length; i++) {

**if** (judge(sign[i]) == 0) {

            result[step] = sign[i];

            step += 1;

        }

**else** {

**if** (temp.isEmpty() || judge(sign[i]) > judge(temp.getTop()))

                temp.push(sign[i]);

**else** {

**while** (!temp.isEmpty() && judge(sign[i]) <= judge(temp.getTop())) {

**if** (judge(temp.getTop()) != 4) {

                        result[step] = temp.pop();

                        step += 1;

                    }

**else**

**break**;

                }

                temp.push(sign[i]);

            }

        }

    }

**while** (!temp.isEmpty()) {

        result[step] = temp.pop();

        step += 1;

    }

**if** (answer() == -1 && state == **true**) {

        cout << "缺少运算数" << endl;

        state = **false**;

    }

**if** (answer() == -2 && state == **true**) {

        cout << "缺少运算符" << endl;

        state = **false**;

    }

**if** (answer() == -3 && state == **true**) {

        cout << "除数不能为0" << endl;

        state = **false**;

    }

**if** (answer() == -4 && state == **true**) {

        cout << "括号不匹配" << endl;

        state = **false**;

    }

**if** (state) {

        cout << "转换为后缀表达式是：";

**for** (**int** i = 0; i < step; i++) {

            cout << result[i];

**if** (i != step - 1)

                cout << " ";

        }

        cout << endl;

        cout << "计算结果是：" << ans << endl;

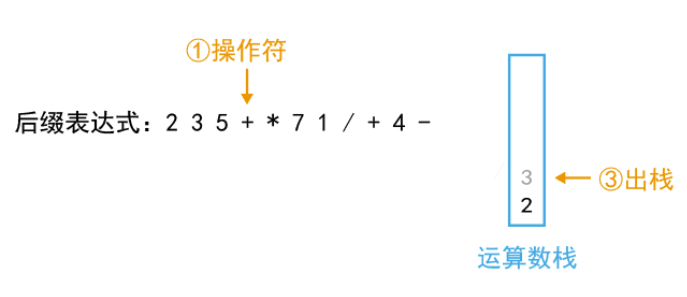
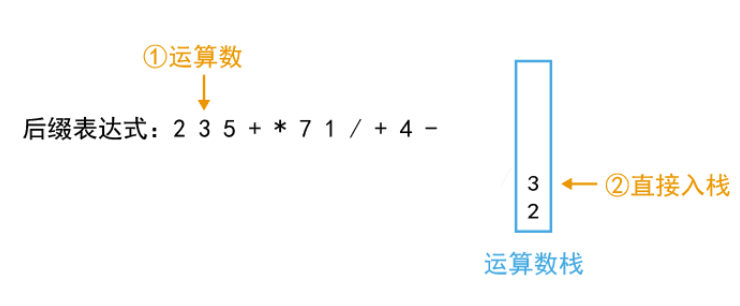
    }

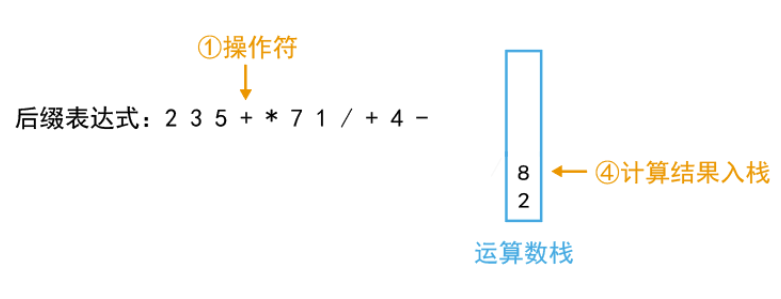
}

## 3.2 后缀表达式计算功能实现

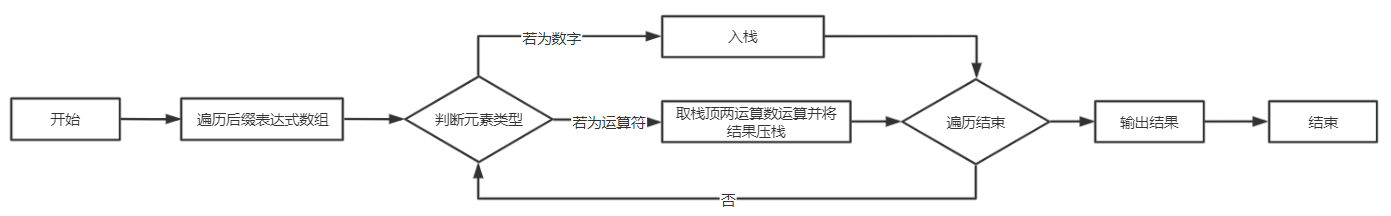
### 3.2.1后缀表达式计算功能实现思路

[后缀表达式](https://so.csdn.net/so/search?q=%E5%90%8E%E7%BC%80%E8%A1%A8%E8%BE%BE%E5%BC%8F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/sjhdxpz/article/details/_blank)的求值规则为：从左到右扫描后缀表达式，如果遇到操作数，将其压入栈中，如果遇到操作符，则从栈中弹出两个操作数，计算结果，然后把结果入栈，直到遍历完后缀表达式，则计算完成，此时的栈顶元素即为计算结果。





### 3.2.2后缀表达式计算功能实现流程图



### 3.2.3后缀表达式计算功能实现代码

//求出表达式的计算结果

**int** calculator::answer() {

    Stack<string> stack;

**for** (**int** i = 0; i < step; i++) {

**if** (judge(result[i]) > 0) {

**if** (judge(result[i]) == 4 || judge(result[i]) == 1)

**return** -4;

**if** (stack.getLength() < 2)

**return** -1;

            stringstream s1(stack.pop());

            stringstream s2(stack.pop());

**double** x1, x2, newNumber = 0;

            s1 >> x1;

            s2 >> x2;

**if** (result[i] == "+")

                newNumber = x2 + x1;

**else** **if** (result[i] == "-")

                newNumber = x2 - x1;

**else** **if** (result[i] == "\*")

                newNumber = x2 \* x1;

**else** **if** (result[i] == "/") {

**if** (x1 == 0)

**return** -3;

                newNumber = x2 / x1;

            }

            ostringstream os;

            os.precision(16);

            os << newNumber;

            stack.push(os.str());

        }

**else** **if** (judge(result[i]) == 0)

            stack.push(result[i]);

    }

**if** (stack.getLength() != 1)

**return** -2;

    ans= stack.getTop();

**return** **true**;

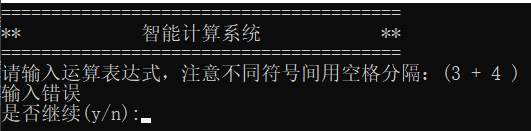
}

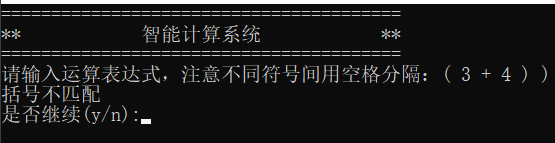
## 3.3 用户输入错误处理功能实现

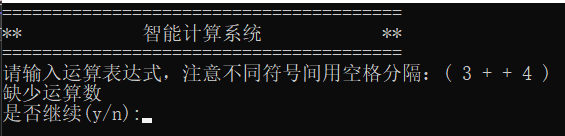
### 3.3.1用户输入错误处理功能实现思路

用户输入错误类型主要分为两类，一是输入不合法的字符，比如输入时没有按空格分隔，导致系统读入的字符为’3)’，该字符既不为数字也不为运算符，系统报错；二是存在运算逻辑问题，比如两个数字缺少运算符、两个运算符间缺少数字、左右括号不匹配等。对不合法字符的判别可以在读取中缀表达式的同时进行，读取到既不为数字也不为运算符的字符时给出提示并让用户再次输入。而运算逻辑的判别可以在计算后缀表达式过程中进行，当读取到运算符而此时栈内元素个数少于2时说明缺少运算数，当最后栈内元素个数不为1时说明左右括号不匹配或缺少运算符......由于项目对用户输入的错误判别在几个阶段中进行，故可定义一个变量state=1,任何阶段查找到用户输入表达式有误后直接将state置0，程序跳出并给出提示。

### 3.3.2用户输入错误处理功能实现样例







# 

# 4 测试

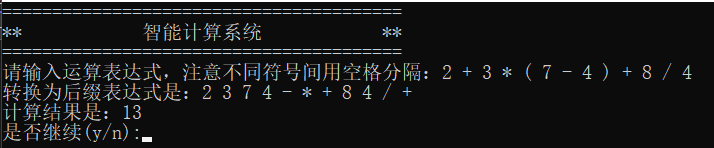
## 4.1 计算功能测试

### 4.1.1 正常测试6种运算符

输入：2 + 3 \* ( 7 – 4 ) + 8 / 4

预期输出：2 3 7 4 - \* + 8 4 / +

实际结果：

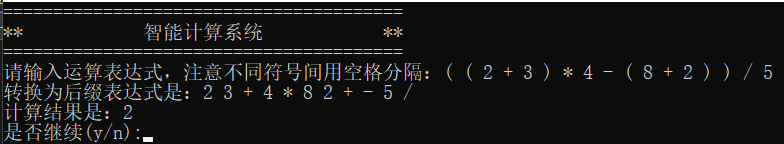


### 4.1.2 嵌套括号

输入：( ( 2 + 3 ) \* 4 – ( 8 + 2 ) ) / 5

预期输出：2 3 + 4 \* 8 2 + - 5 /

实际结果：

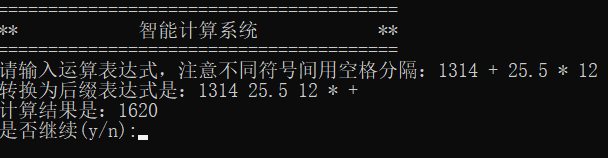


### 4.1.3 运算数超过1位整数且有非整数出现

输入：1314 + 25.5 \* 12

预期输出：1314 25.5 12 \* +

实际结果：

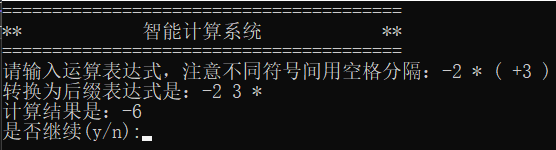


### 4.1.4 运算数有正或负号

输入：-2 \* ( +3 )

预期输出：-2 3 \*

实际结果：

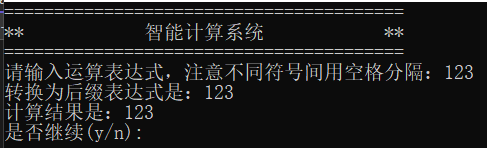


### 4.1.5 只有1个数字

输入：123

预期输出：123

实际结果：

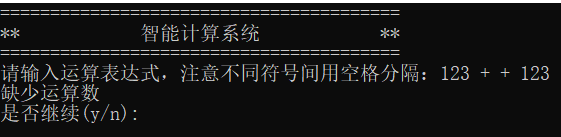


### 4.1.6 错误测试—缺少运算数

输入：123 + + 123

预期输出：报错

实际结果：

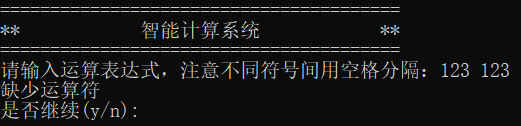


### 4.1.7 错误测试—缺少运算符

输入：123 123

预期输出：报错

实际结果：

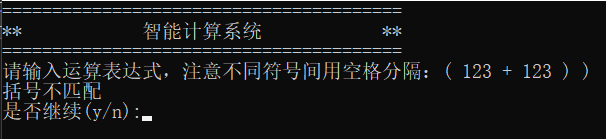


### 4.1.8 错误测试—括号不匹配

输入：( 123 + 123 ) )

预期输出：报错

实际结果：

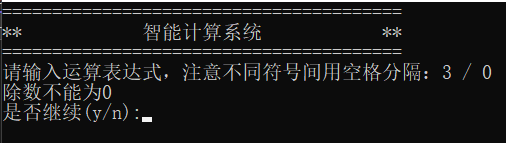


### 4.1.9 错误测试—除数为0

输入：3 / 0

预期输出：报错

实际结果：

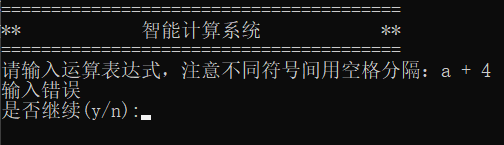


### 4.1.9 错误测试—输入错误

输入：a + 4

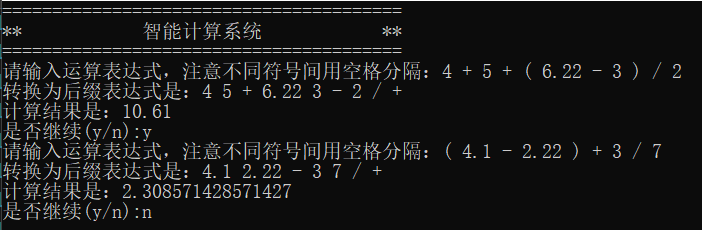
预期输出：报错

实际结果：



## 4.2 循环功能测试

### 4.2.1 正常循环



### 4.2.1 输入错误

