同济大学
TONGJI UNIVERSITY

项目说明文档

算数表达式求解

指导老师：张颖

1851009 沈益立

目录

1.分析

1.1 背景分析

算数表达式有前缀表示法、中缀表示法和后缀表示法等方式。人们在日常表达算式时最常使用的是中缀表达式，但是在计算机中，使用中缀表达式有操作符的优先顺序问题，还有可加括号改变优先 级的问题，与之相比，利用栈可以方便的对后缀表达式进行计算，因此在编译程序中一般采用后缀表达式求解表达式的值。

因此，本项目要求设计一个输入表达式，输出计算结果的程序。

1.2 功能分析

本次设计的项目应该首先满足题目给出的基本要求，对于基本的数字和样例应该实现正确的转化，并且在有错误时给出一些合适的提示。基于此，本程序对于括号、负号、带符号的数等进行了一些处理，在考虑这些的前提下对程序进行了改进。

2.设计

2.1 数据结构设计

人们在日常表达算式的时候最常使用的是中缀表达式，但是使用中缀表达式需要理解操作符的优先级顺序，因此如果要计算出中缀表达式的值，首先需要将中缀表达式转为计算机容易理解的后缀表达式。采用栈的数据结构，让先出现的符号先进栈等待，当下一个出现的符号优先级没有栈内的符号高中，再进行输出。

在上述的数据结构的基础上，再附加一些边界条件的判断即可对错误进行适当的提示。

2.2 类结构设计

由于本题目不允许使用STL等库，因此采用了自己设计的栈`Stack`。同时，为了面向对象的思想和避免全局变量，设计了类`PolandConvert`。

2.3 成员和操作设计

栈类`Stack`

公有操作：

```
Stack();                        //无参数构造函数
~Stack();                       //析构函数
void push(const T& val);        //将新元素推入栈
void showElements();            //展示栈中的元素
bool isEmpty();                 //判断是否栈空
void makeEmpty();               //清空栈
T pop();                        //弹出元素
T top();                        //得到栈顶元素
int getSize();                  //得到栈的长度
T* getFirstAddress();           //得到栈首地址
```

私有成员:

```
int m_top;                      //表示栈顶
int m_maxSize;                  //栈的最大长度
T* m_elements;                  //模拟栈的数组
```

波兰表达式转换类`PolandConvert`

公有操作:

```
void IOLoop();                  //基本IO的处理
bool convertExpression();       //转换表达式为后缀
bool proceedPost();             //处理后缀表达式
bool isOpChr(string s);         //判断s是否为操作符
int getPriorVal(char chr);      //得到操作符的优先级
int getValue();                 //得到结果
```

私有成员:

```
int m_val = 0;                  //结果的暂存成员
Stack<int> valStack;            //处理中缀表达式时的值的栈
Stack<char> charStack;          //处理中缀表达式时的符号栈
string midExpression;           //中缀表达式
Stack<string> postExpression;//后缀表达式栈
```

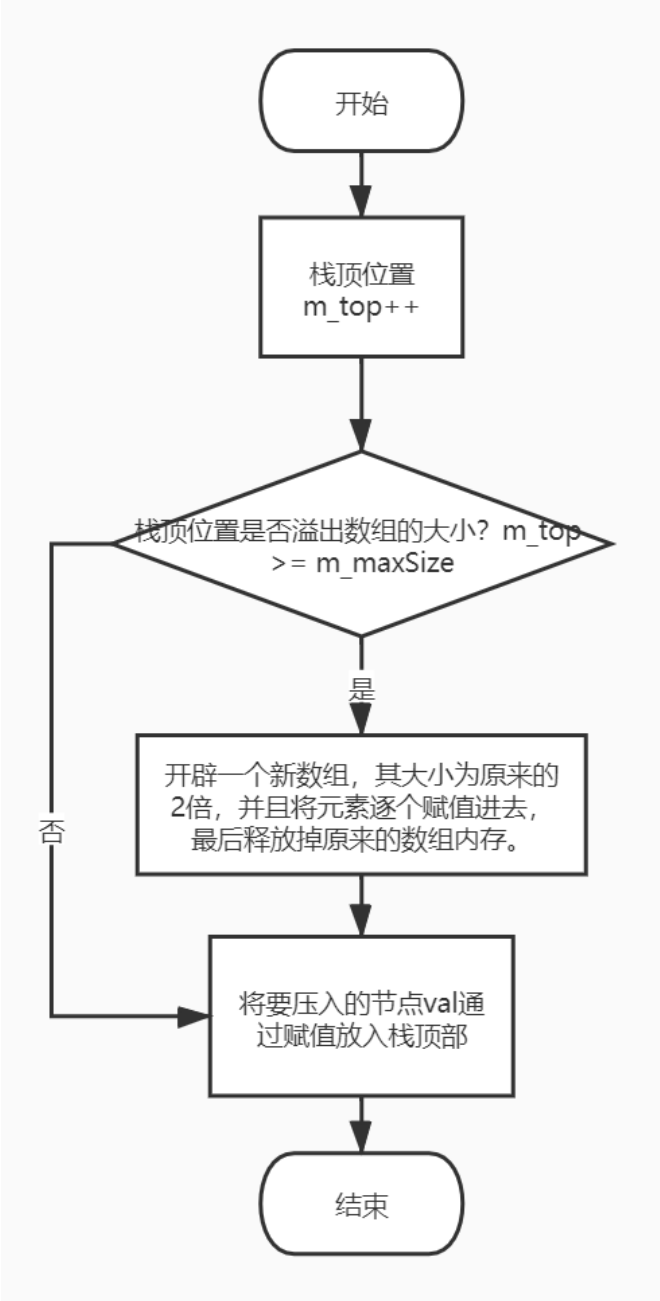
2.4 系统设计

程序运行后，系统会自动实体化一个`PolandConvert`的实体类`sol`，随后调用其成员函数`IOLoop()`，实现基本IO，并且处理输入的中缀表达式，逐级检查和处理后，输出其计算结果。

3.实现

3.1 栈压入功能的实现

3.1.1 栈压入功能流程图

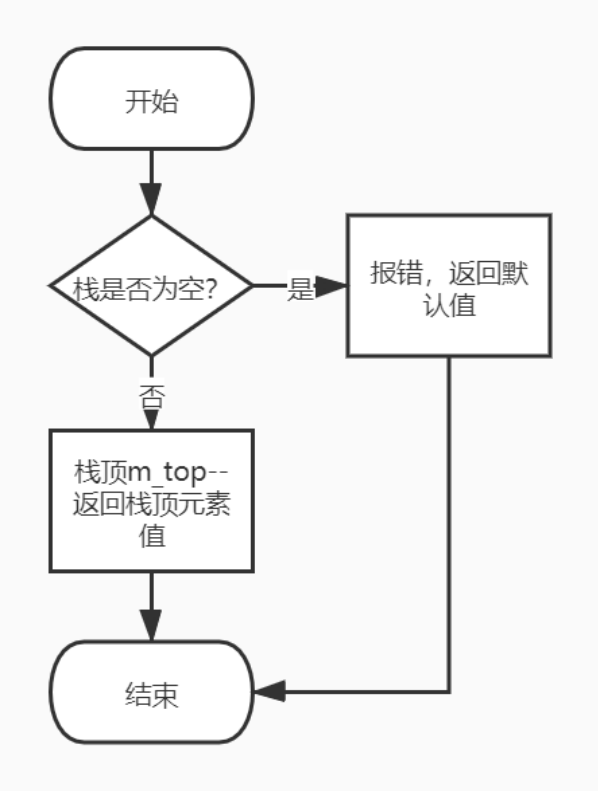


3.1.2 栈压入功能核心代码实现

```
template<class T>
void Stack<T>::push(const T& val){
    m_top++;
    if (m_top >= m_maxSize) {//栈溢出
        T* tmp = m_elements;
        m_maxSize *= 2;
        m_elements = new T[m_maxSize];
        for (int i = 0; i < m_maxSize / 2; i++) {
            m_elements[i] = tmp[i];
        }//copy
        delete[] tmp;
    }
    m_elements[m_top] = val;
}
```

3.2 栈弹出功能的实现

3.2.1 栈弹出功能流程图

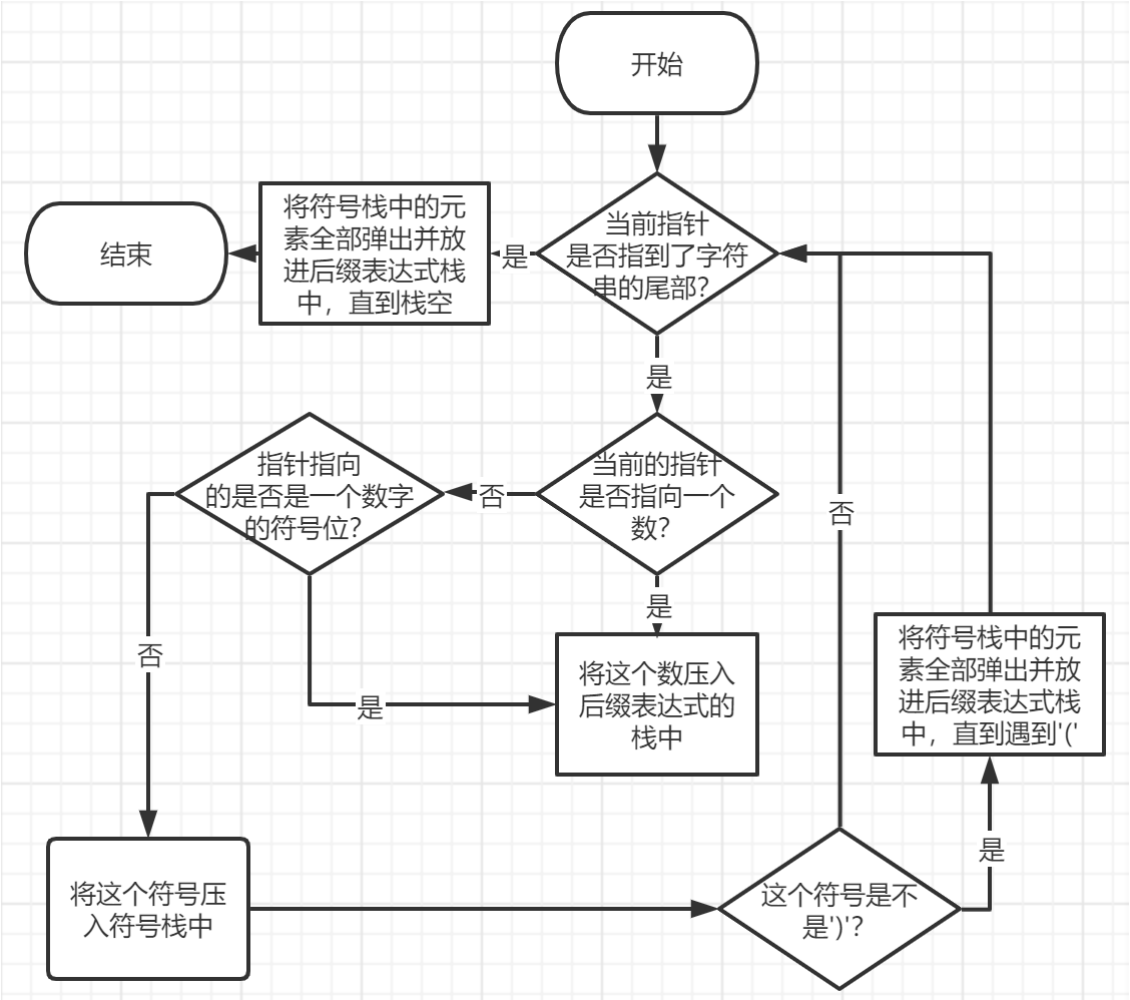


3.2.2 栈弹出功能核心代码实现

```
template<class T>
T Stack<T>::pop(){
    if (isEmpty()) {
        cerr << "Empty stack, nothing to pop." << endl;
        return T();
    }
    T retVal = m_elements[m_top--];
    return retVal;
}
```

3.3 中缀表达式转换功能的实现

3.3.1 中缀表达式转换功能流程图



3.3.2 中缀表达式转换功能核心代码实现

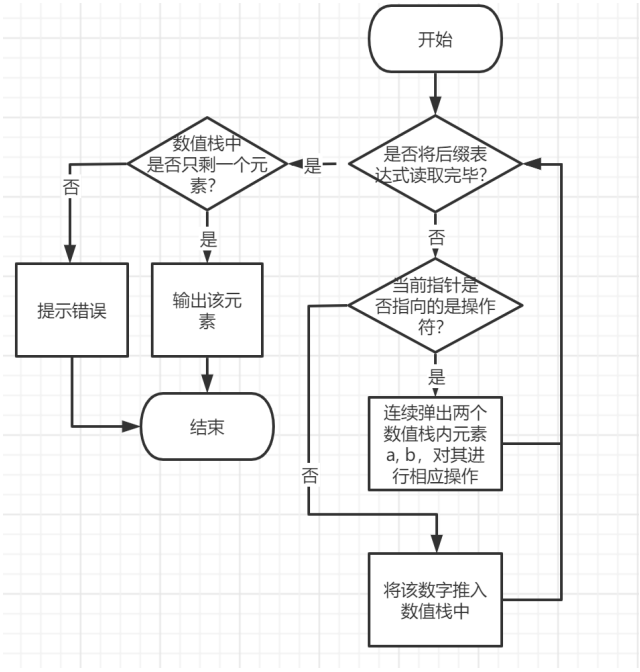
```
bool PolandConvert::convertExpression() {//中缀表达式用栈处理，之后更新val值
    string tmp;
    for (int i = 0; i < midExpression.size() - 1; i++) {
        //操作数处理
        if ((midExpression[i] == '-' || (midExpression[i] == '+') && (i == 0
            || string("+-*/(^%").find(midExpression[i - 1]) != string::npos))//正负号处理
            || getPriorVal(midExpression[i]) == -1) {
            tmp = midExpression[i];
            while (i + 1 < midExpression.size()
                && getPriorVal(midExpression[i + 1]) == -1) {
                tmp += midExpression[++i];
            }

            if (tmp == "+" || tmp == "-") {
                while (charStack.getSize() && (getPriorVal(tmp[0]) <= getPriorVal(charStack.top()))) {
                    string t = "";
                    t += charStack.pop();
                    postExpression.push(t);
                }
                charStack.push(midExpression[i]);
            }
            else {
                postExpression.push(tmp);
            }
        }
        else {//出现操作符
            if (midExpression[i] == '(') {
                charStack.push(midExpression[i]);
            }
            else if (midExpression[i] == ')') {
                while (charStack.top() != '(') {
                    string t = "";
                    t += charStack.pop();
                    postExpression.push(t);
                }
                charStack.pop();
```

```
            }
            else {
                while (charStack.getSize() &&
(getPriorVal(midExpression[i]) <=
getPriorVal(charStack.top()))) {
                    string t = "";
                    t += charStack.pop();
                    postExpression.push(t);
                }
                charStack.push(midExpression[i]);
            }
        }
    }
    //最终处理
    while (charStack.getSize()) {
        string t = "";
        t += charStack.pop();
        postExpression.push(t);
    }
    return true;
}
```

3.4 后缀表达式计算功能的实现

3.4.1 后缀表达式计算功能流程图



3.4.2 后缀表达式计算功能核心代码实现

```
bool PolandConvert::proceedPost() {
    postExpression.showElements();
    string* vec = postExpression.getFirstAddress();
    int vLen = postExpression.getSize();
    for (int i = 0; i < vLen; i++) {
        if (isOpChr(vec[i])) {
            int b = valStack.pop();
            if (!valStack.getSize()) {
                return false;
            }
            int a = valStack.pop();
            int res;
            if (vec[i] == "+") {
                res = a + b;
            }
            else if (vec[i] == "-") {
                res = a - b;
            }
            else if (vec[i] == "*") {
                res = a * b;
            }
            else if (vec[i] == "/") {
                if (b == 0) {
                    cerr << "不可以除以0" << endl;
                    return false;
```

```
                }
                res = a / b;
            }

            else if (vec[i] == "%") {
                if (b == 0) {
                    cerr << "不可以模0" << endl;
                    return false;
                }
                res = a % b;

            }
            else if (vec[i] == "^") {
                res = pow(a, b);
            }
            valStack.push(res);
        }
        else {
            valStack.push(atoi(vec[i].c_str()));
        }
    }
    if (valStack.getSize() != 1) {
        return false;
    }
    m_val = valStack.top();

    return true;
}
```

4.测试

4.1 常规测试

4.1.1 正常测试六种运算符

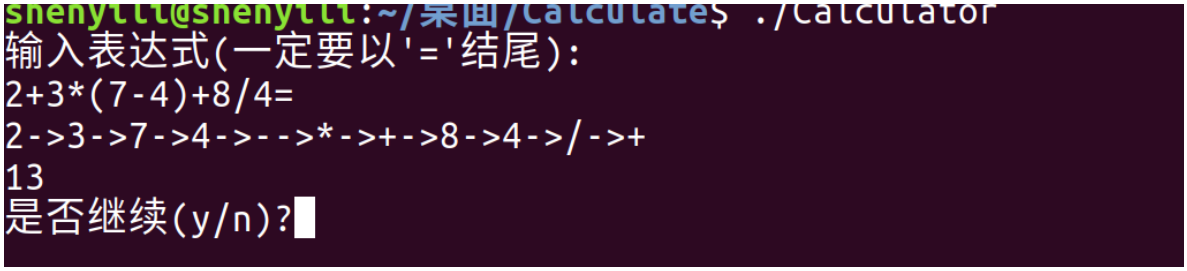
测试用例：

```
2+3*(7-4)+8/4=
```

预期结果：

```
13
```

实验结果：



4.1.2 嵌套括号

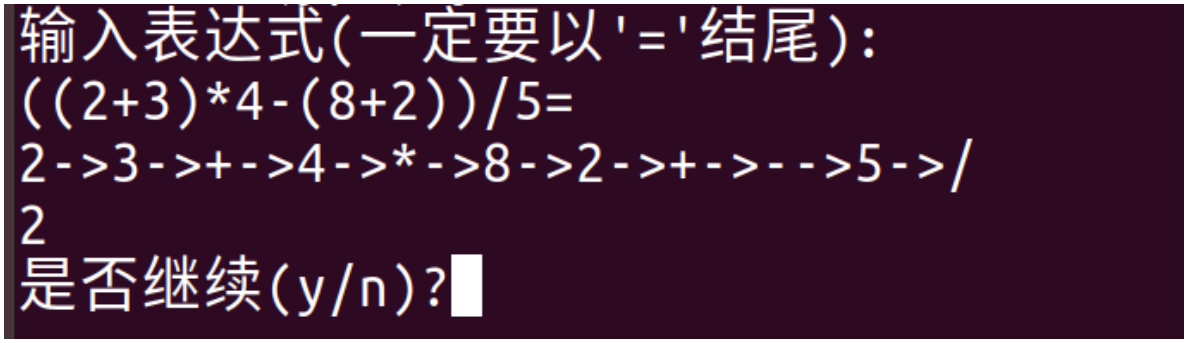
测试用例：

```
((2+3)*4-(8+2))/5
```

预期结果：

```
2
```

实验结果：



4.1.3 运算数有正负号

测试用例：

```
-2*(3)
```

预期结果：

```
-6
```

实验结果：

4.1.4 只有一个数字

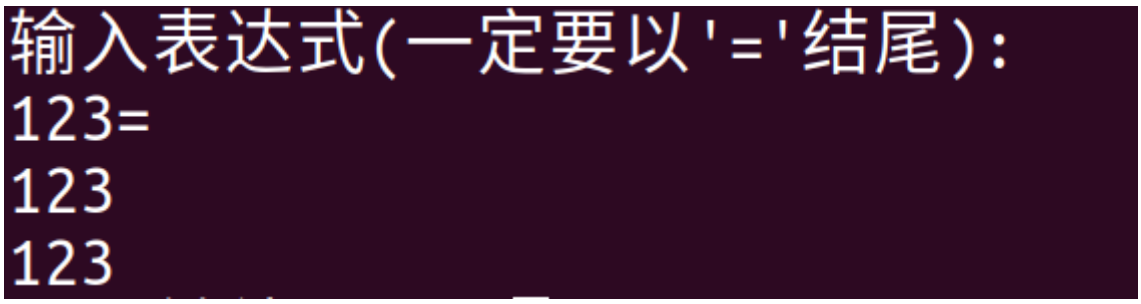
测试用例：

```
123
```

预期结果：

```
180
```

实验结果：



4.1.5 许多括号

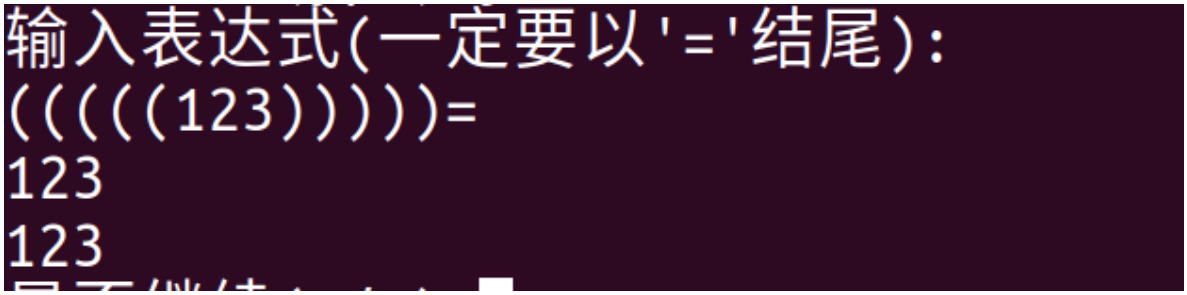
测试用例:

```
(((((123)))))
```

预期结果:

```
123
```

实验结果:



4.2 特殊输入

4.2.1 连续符号输入

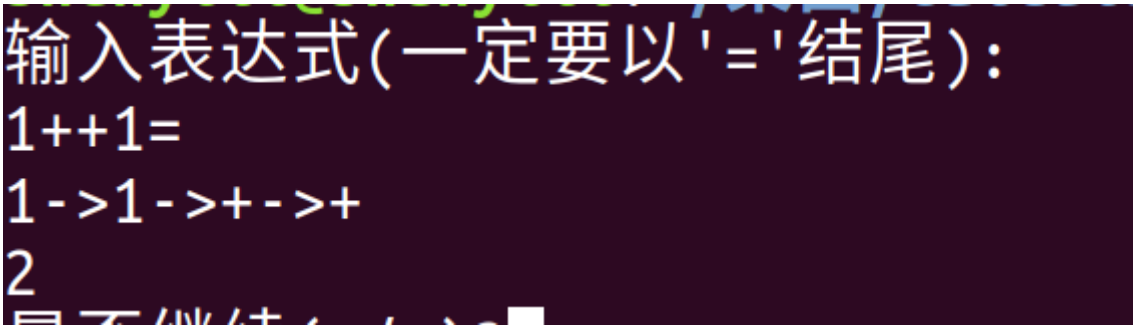
测试用例:

```
1++1
```

预期结果:

```
2
```

实验结果:



4.2.2 无意义操作符输入

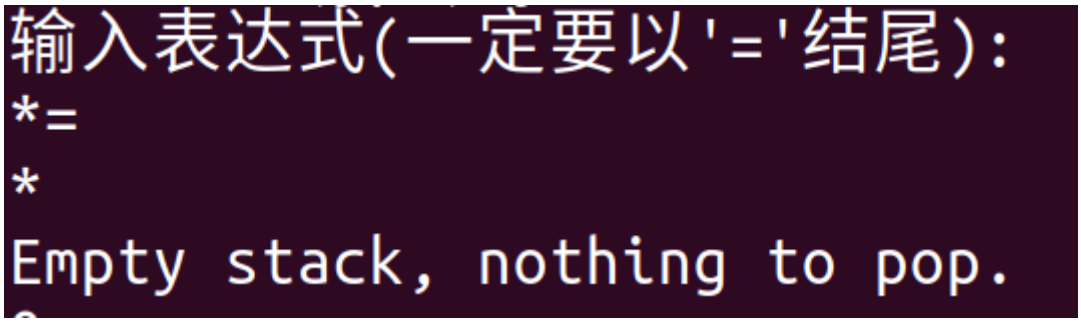
测试用例:

```
*
```

预期结果:

```
报错
```

实验结果:



4.2.3 非法字符串输入

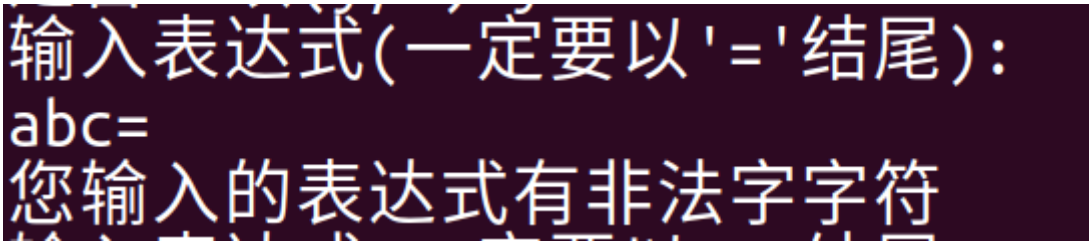
测试用例:

```
abc
```

预期结果:

```
报错
```

实验结果:



4.3 鲁棒性测试

4.3.1 模0测试

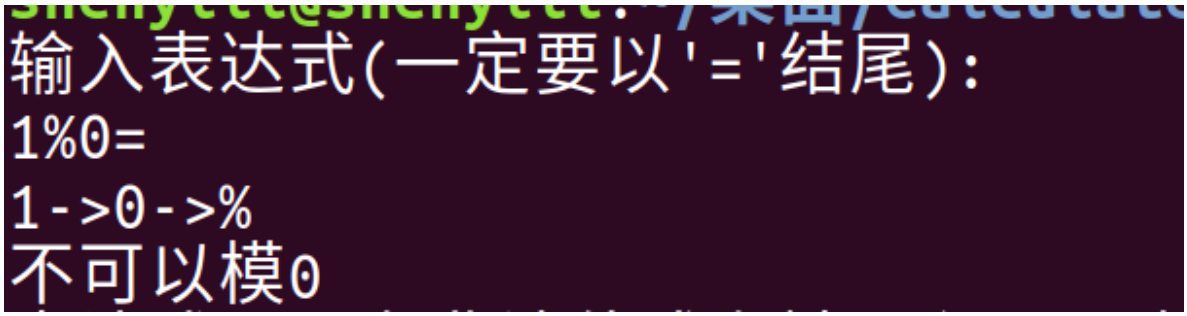
测试用例:

```
1%0
```

预期结果:

```
提示模0无意义
```

实验结果:



4.3.2 除0测试

测试用例:

```
1/0
```

预期结果:

```
提示除0无意义
```

实验结果:

4.3.3 无等号输入测试

测试用例:

```
1+1
```

预期结果:

```
提示缺少等号
```

实验结果: