**数据结构课程设计**

**项目说明文档**

**表达式转换**

作 者 姓 名： 苏家铭

学 号： 2151299

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

**Tongji University**



**目录**

[1 项目分析 1](#_Toc23685)

[1.1 项目背景 1](#_Toc19217)

[1.2项目要求 1](#_Toc25807)

[1.2.1 功能要求 1](#_Toc11809)

[1.2.2 输入要求 1](#_Toc31355)

[1.2.3 输出要求 1](#_Toc6313)

[1.2.4 项目实例 1](#_Toc25283)

[2 项目实现 2](#_Toc6020)

[2.1 数据结构设计思路 2](#_Toc11270)

[2.2 类实现 2](#_Toc13472)

[2.2.1 链表 2](#_Toc27933)

[2.2.2 链式栈 4](#_Toc7859)

[2.3 项目算法 5](#_Toc17147)

[2.3.1 实现思路 5](#_Toc3192)

[2.3.2 代码实现 7](#_Toc10422)

[3 项目测试 9](#_Toc22216)

[3.1 正常测试6种运算符 9](#_Toc15710)

[3.2 嵌套括号 10](#_Toc2990)

[3.3 运算数超过1位整数且有非整数出现 10](#_Toc23867)

[3.4 运算数有正或负号 10](#_Toc11020)

[3.5 只有1个数字 11](#_Toc21086)

[3.6 输入非法字符的情况（健壮性） 11](#_Toc24965)

[4 算法性能分析 12](#_Toc15984)

[4.1 正确性 12](#_Toc31613)

[4.2 可使用性 12](#_Toc12310)

[4.3 可读性 12](#_Toc22907)

[4.4 效率 12](#_Toc10348)

[4.5 健壮性 12](#_Toc14379)

# 1 项目分析

## 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。

由于中缀表示中有操作符的优先级问题，还有可加括号改变运算顺序的问题，所以对于编译程序来说，一般不适用中缀表示处理表达式。解决办法是用后缀表达式（较为常用）和前缀表达式表示。因为用后缀表示计算表达式的值只用一个栈，而前缀表示和中缀表示同时需要两个栈，所以编译程序一般使用后缀表示求解表达式的值。

明晰将中缀表达式转换成为后缀表达式的方法，熟悉转换的过程，对于理解计算机原理有重要的作用，这也对于锻炼计算机思维是大有裨益的。

## 1.2项目要求

### 1.2.1 功能要求

将中缀表达式转换成为后缀表达式。

### 1.2.2 输入要求

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）。

### 1.2.3 **输出要求**

### 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。

### 1.2.4 项目实例

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 2 + 3 \* ( 7 – 4 ) + 8 / 4 | 2 3 7 4 - \* + 8 4 / + | 正常测试6种运算符 |
| 2 | ( ( 2 + 3) \* 4 – ( 8 + 2 ) ) / 5 | 2 3 + 4 \* 8 2 + - 5 / | 嵌套括号 |
| 3 | 1314 + 25.5 \* 12 | 1314 25.5 12 \* + | 运算数超过1位整数且有非整数出现 |
| 4 | -2 \* ( +3 ) | -2 3 \* | 运算数有正或负号 |
| 5 | 123 | 123 | 只有1个数字 |

# 2 **项目实现**

## 2.1 **数据结构设计思路**

根据项目要求，本例应当使用栈作为存储结构将中缀表达式转换为后缀表达式，在完成项目要求的基础上，本人还完成了后缀表达式的求解。这两个过程都需要用到栈作为辅助工具，本人基于链表实现了链式栈，以做到将元素先进后出(FILO)。

## 2.2 **类实现**

### 2.2.1 链表

虽然带头结点的双向循环链表结构复杂，但使用代码后会发现许多优势。该链表一般包括两个抽象数据类型（ADT）：链表结点类（DblNode）与链表类（DblList），而两个类之间的耦合关系可以采用嵌套、复合等多种关系。若结点类定义为class那么需要在类内定义友元函数链表类才能访问，为了实现代码的复用性，本人使用结构体来定义链表的结点，此方法虽然使得DblNode类失去了封装性，但链表类可以直接访问结点中的数据成员。

节点类数据成员：

template<class T>

T data;//链表结点数据

DblNode<T>\* lLink,\*rLink;//链指针域,前驱（左链）、后继（右链）

链表类数据成员：

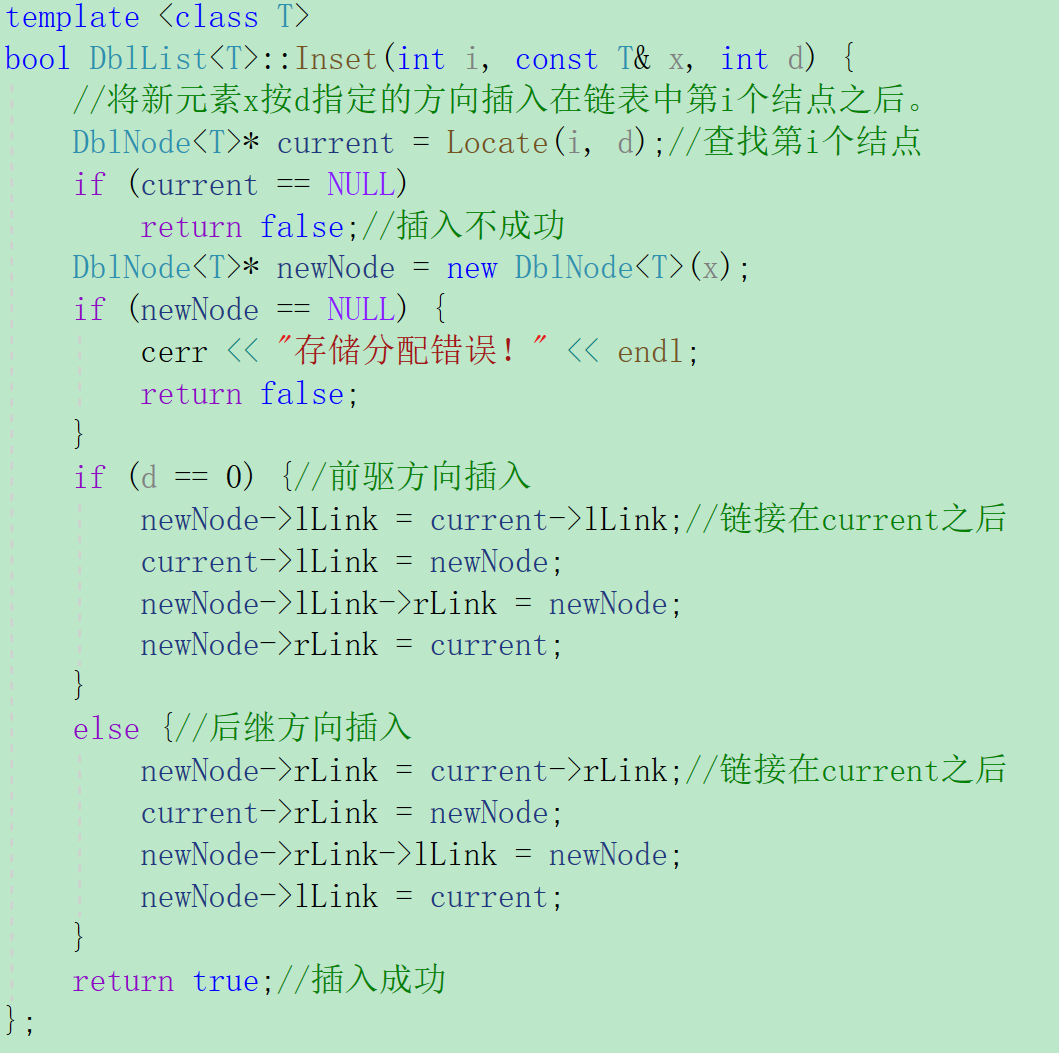
DblNode<T>\* first;//链表头结点

该链表已经实现19种功能：置空、计算长度、返回头结点、设置头结点、搜索、定位、返回元素地址、修改元素的值、插入、删除、置空、判空、判满、尾部插入、头部删除、向后插入i个元素、向后插入若干元素（直到遇到endTag）、输出链表、等号重载。

具体函数为：

* void makeEmpty();//将链表置为空表
* int Length()const;//计算双链表的长度
* void setHead(DblNode<T>\* ptr)//设置附加头结点地址
* DblNode<T>\* getHead()const//返回附加头结点地址
* DblNode<T>\* Search(const T& x);//在链表中沿后继寻找等于数据x的元素
* DblNode<T>\* Locate(int i,int d);//在链表中定位序号为i（>=0）的结点，d=0按前驱方向，d!=0按后继方向
* bool getDate(int i, T& x,int d);//取出按d方向第i个元素的地址
* void setData(int i, T& x,int d);//用x修改按d方向第i个元素的值
* bool Inset(int i, const T& x,int d);//在第i个元素后插入x,d=0按前驱方向，d!=0按后继方向
* bool Remove(int i, T& x,int d);//删除第i个元素,x返回该元素的值,d=0按前驱方向，d!=0按后继方向
* bool IsEmpty()const//判表空否?空则返回true
* bool IsFull()const {return false;}//判表满否？不满则返回false
* void push\_back(T data);//在尾部插入一个数据
* void push\_front(T data);//在前部插入一个数据
* void input\_num(int i);//输入i个元素
* void input\_endTag(T endTag);//输入若干元素，直到遇到endTag
* void output(int d);//输出
* DblList<T>& operator= (const DblList<T>&L);//重载函数：赋值

插入算法的代码实现：



主要算法的时间复杂度：

* 插入：O(n)
* 删除：O(n)
* 搜索：O(n)
* 遍历：O(n)
* 修改：O(n)

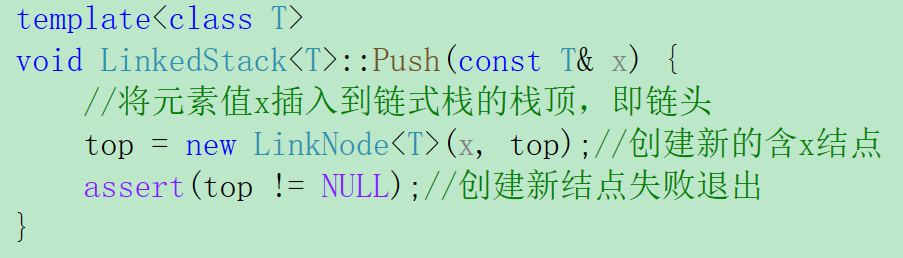
### 2.2.2 链式栈

[栈](https://so.csdn.net/so/search?q=%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41575507/article/details/_blank)是重要的线性结构，从数据结构的角度看，栈也是线性表，其特殊性在于栈的基本操作是线性表的子集。这是一种是操作受限的线性表，因此，可称为限定性的数据结构。但从数据类型角度看，他是和[线性](https://so.csdn.net/so/search?q=%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41575507/article/details/_blank)表大不相同的重要的抽象数据类型。栈(stack)是限定仅在表尾进行插入或者删除的线性表。对于栈来说，表尾端称为栈顶(top)，表头端称为栈低(bottom)。因为栈限定在一端进行插入或者删除，所以栈又被称为后进先出的线性表（LIFO）。

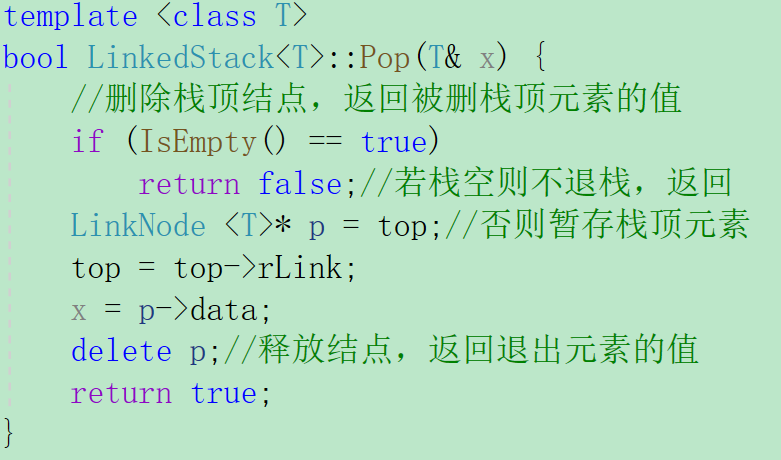
本人基于链表实现了链式栈的进栈、退栈、返回栈顶元素、返回栈内元素个数、置空、判空以及左移运算符的重载操作

* void Push(const T& x);//一个元素进栈
* bool Pop(T& x);//一个元素退栈
* bool getTop(T& x)const;//返回栈顶元素
* bool IsEmpty()const //判空
* int getSize()const;//返回栈的元素个数
* void makeEmpty();//清空栈
* friend ostream& operator << (ostream& os, LinkedStack<T>& s);//输出

进栈算法代码实现：



出栈算法代码实现：



主要算法的时间复杂度：

* 进栈：O(1)
* 退栈：O(1)
* 返回栈顶元素：O(1)
* 返回元素个数：O(n)

## 2.3 **项目算法**

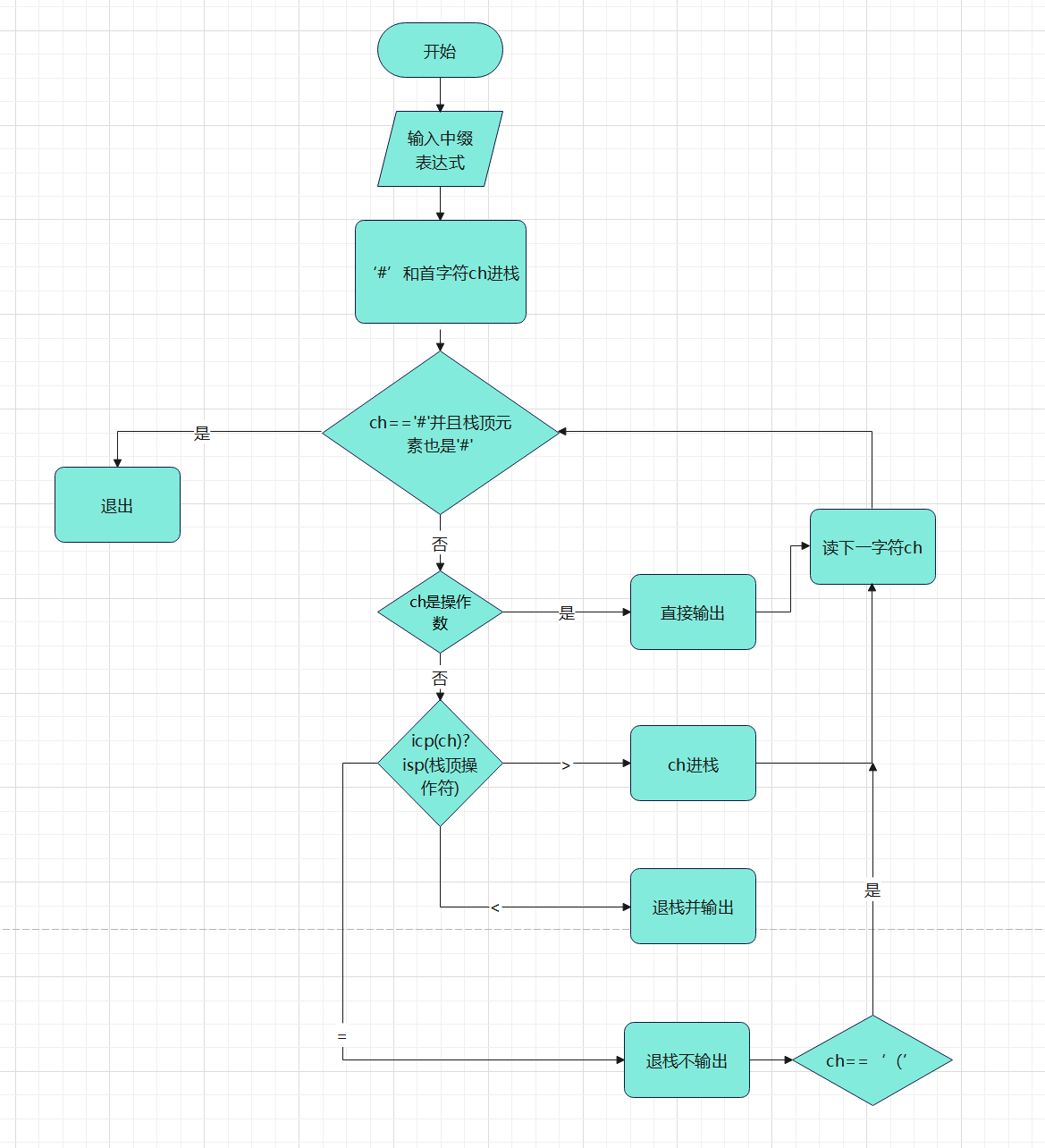
### 2.3.1 实现思路

* 中缀转后缀：

为实现中缀表达式转换为后缀表达式，这边为5类操作符定义了栈内优先数isp(in stack priority)和栈外优先数icp(in coming priority)：

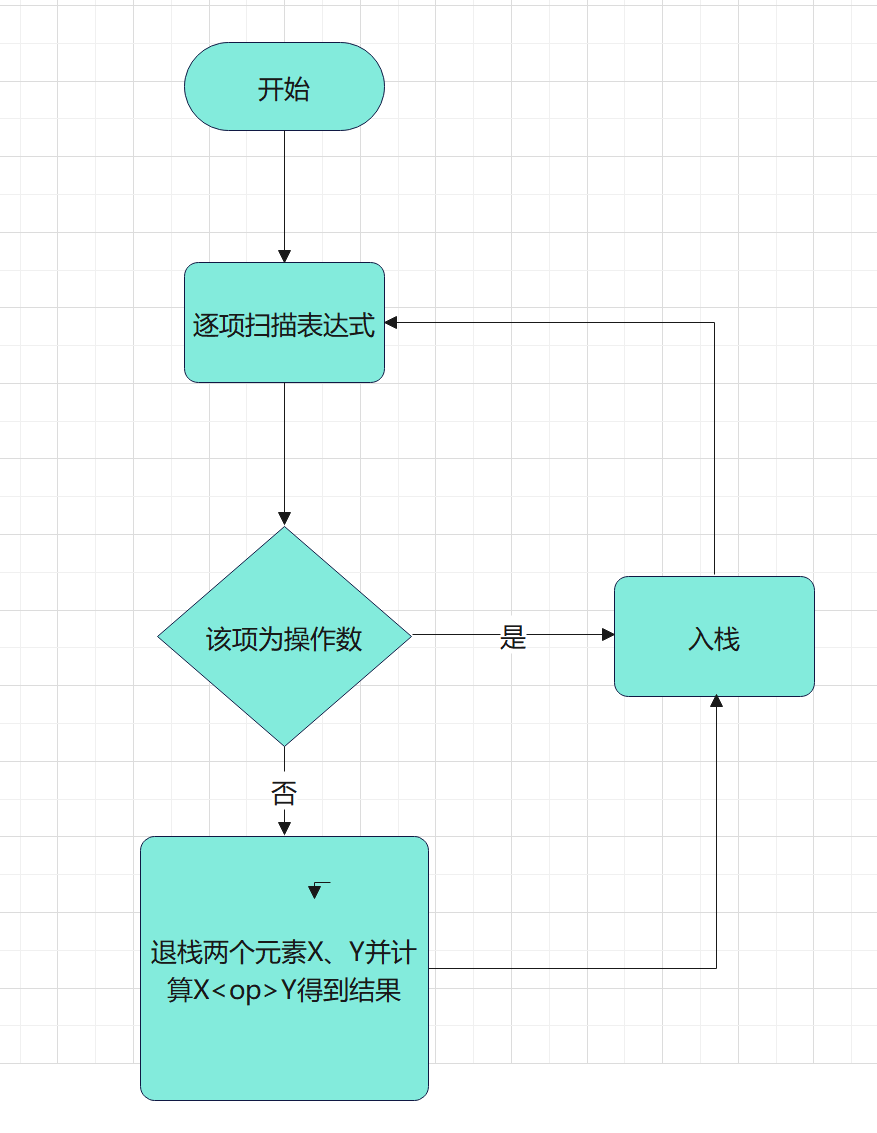
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 操作符ch | # | ( | ) | \*,/,% | +,- |
| Isp | 0 | 1 | 6 | 5 | 3 |
| Icp | 0 | 6 | 1 | 4 | 2 |

对于一个已知的中缀表达式，将其转化为后缀表达式的流程图为：



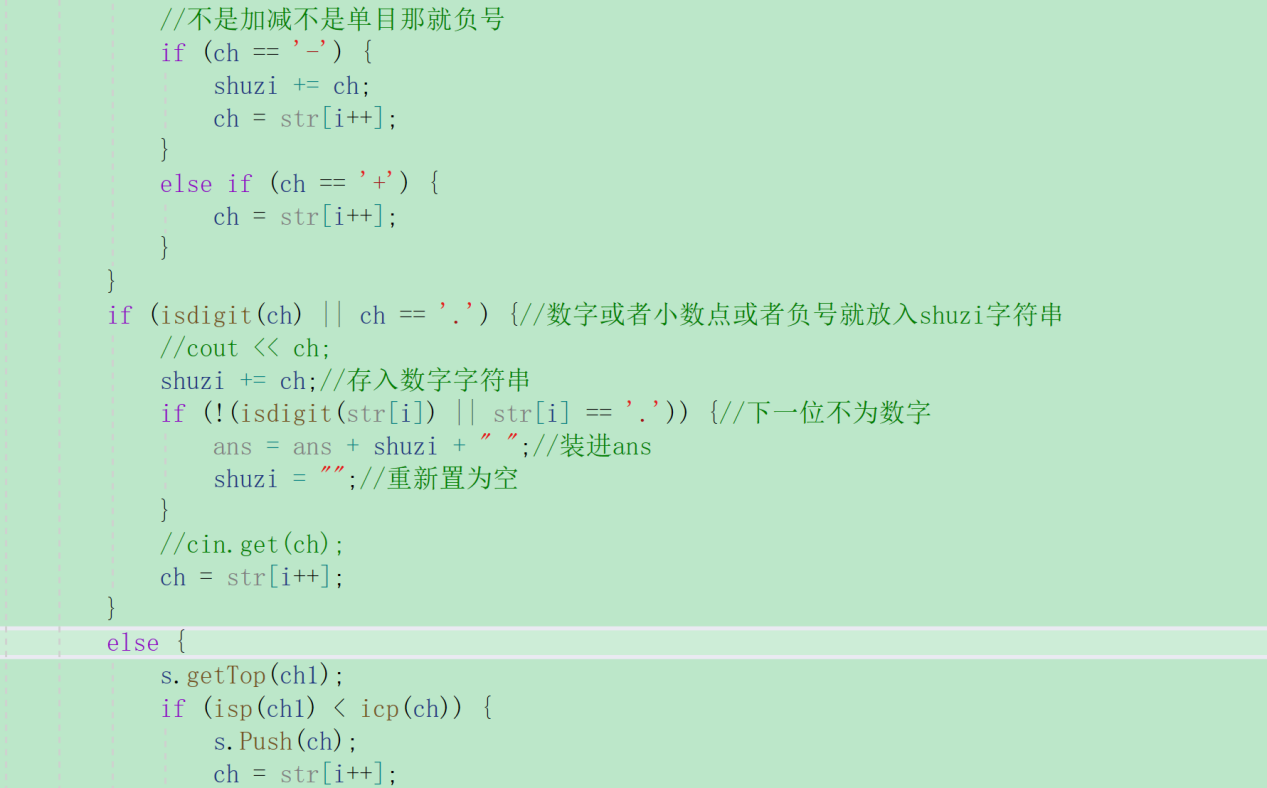
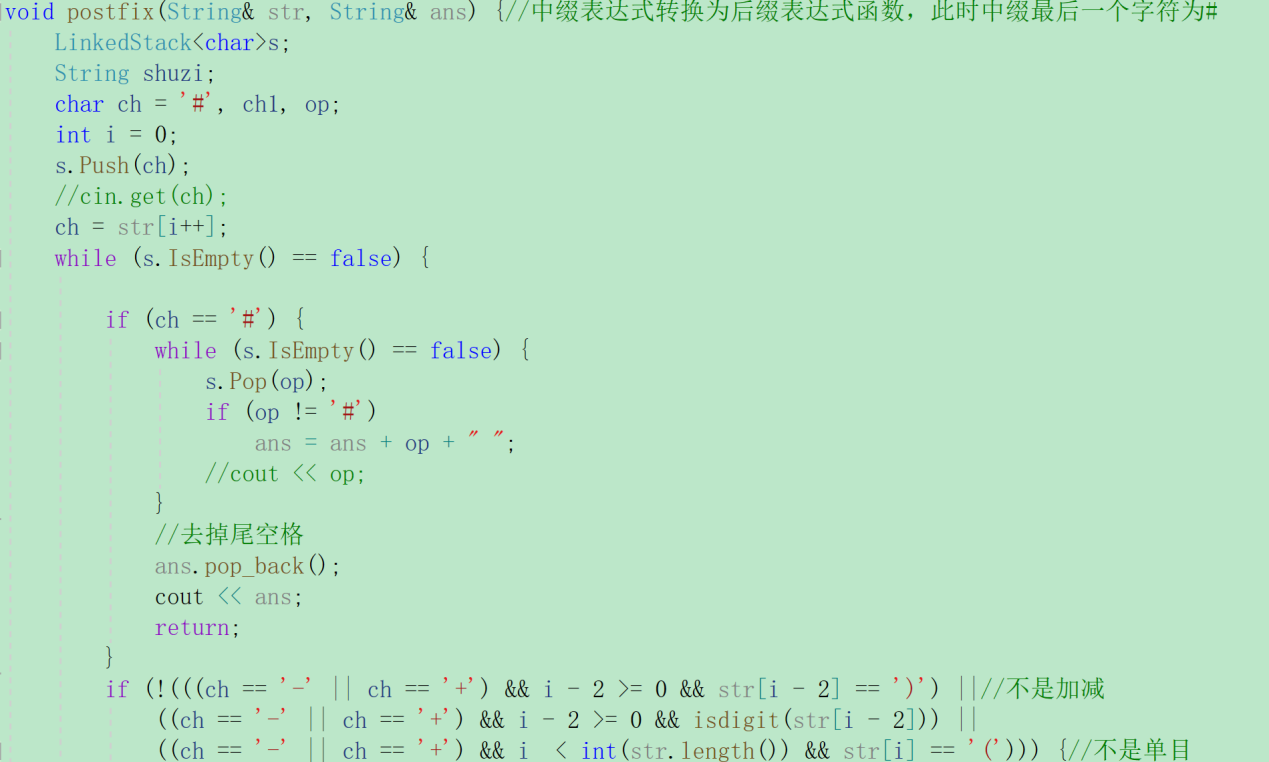
* 后缀表达式（RPN/逆波兰表达式）求值：

顺序扫描所得的后缀表达式，以项为单位，判别该项是操作数还是操作符。若是操作数则压入栈内， 若是操作符则从栈中取两个操作数进行运算，所得结果再压入栈，直到所有项都被扫描完毕，栈顶的元素即为所求。

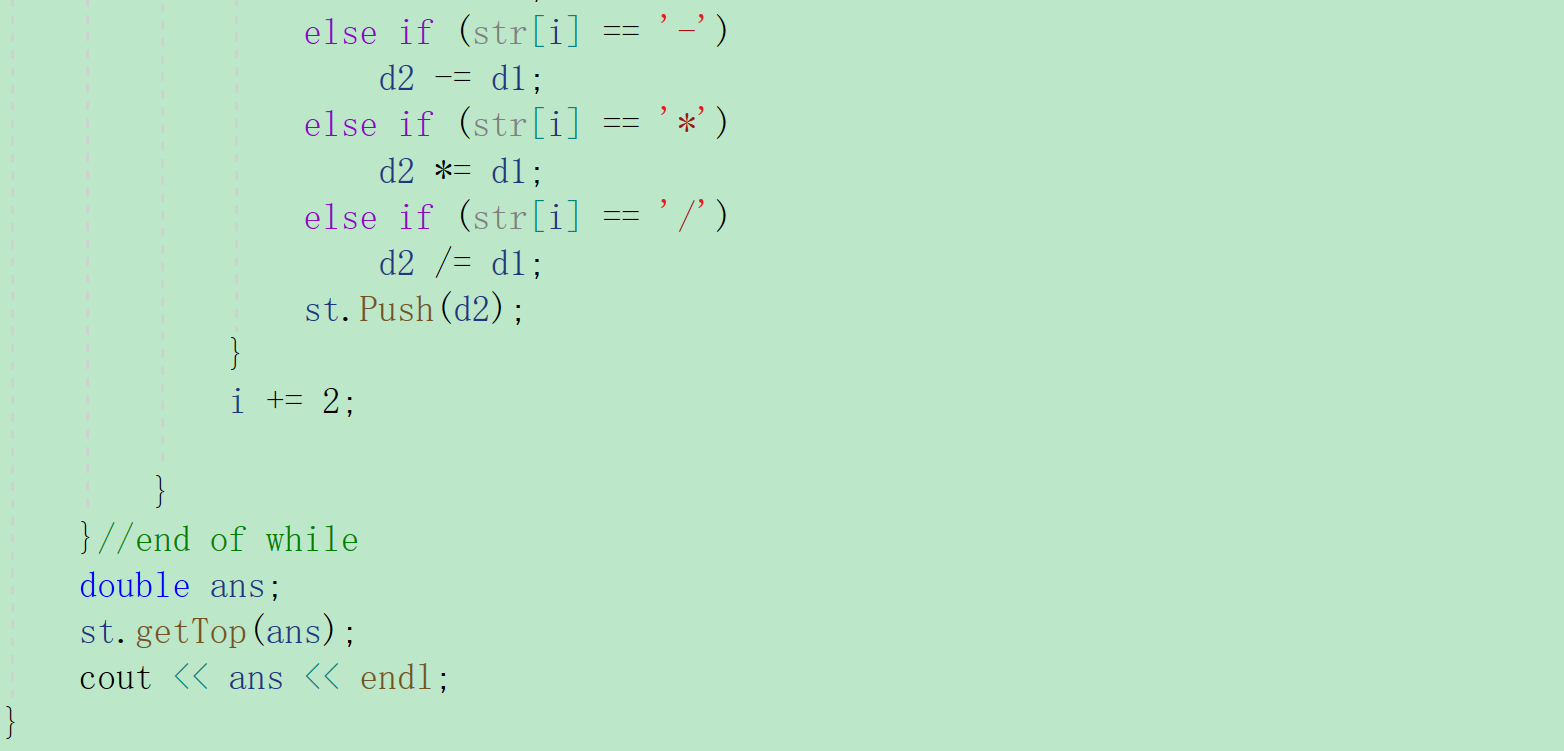
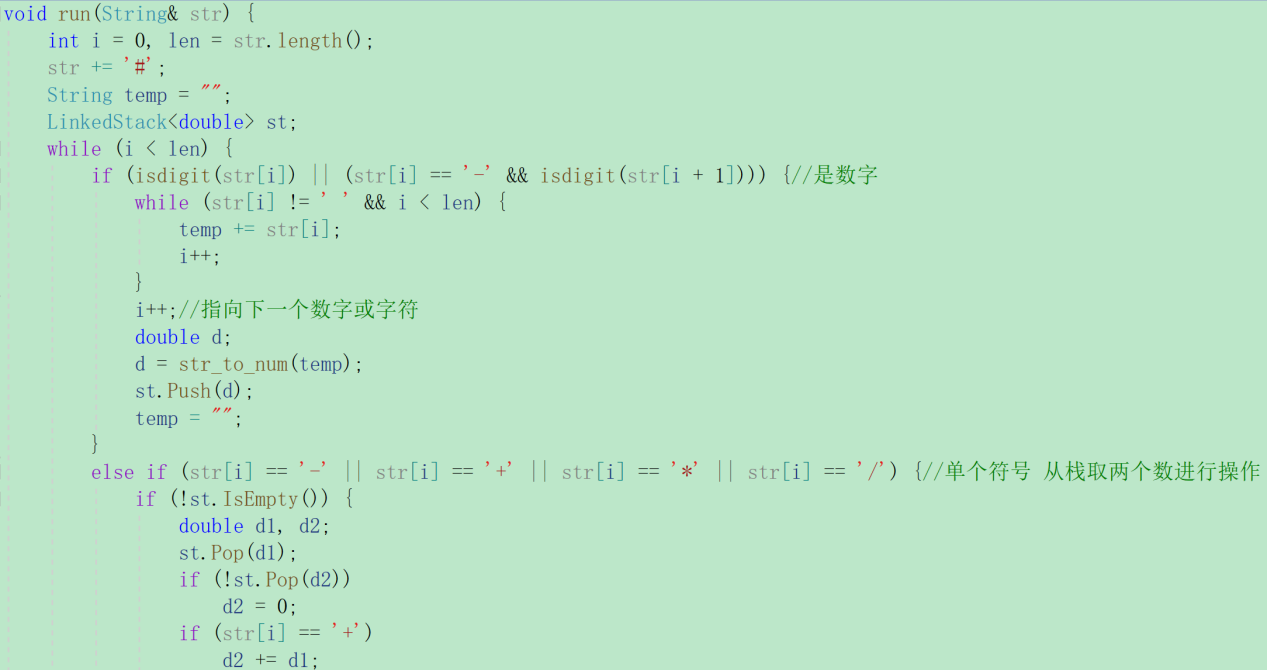


### 2.3.2 代**码实**现

* 中缀表达式转换为后缀表达式：



* 后缀表达式求值：



# 3 项目测试

## 3.1 正常测试6种运算符

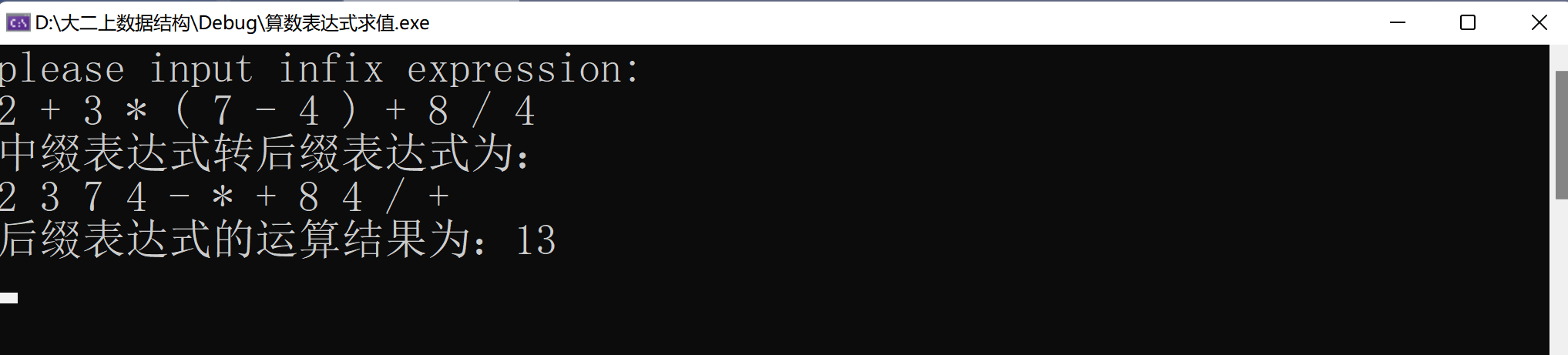
测试输入：

2 + 3 \* ( 7 - 4 ) + 8 / 4

预期输出：

2 3 7 4 - \* + 8 4 / +

实验结果：



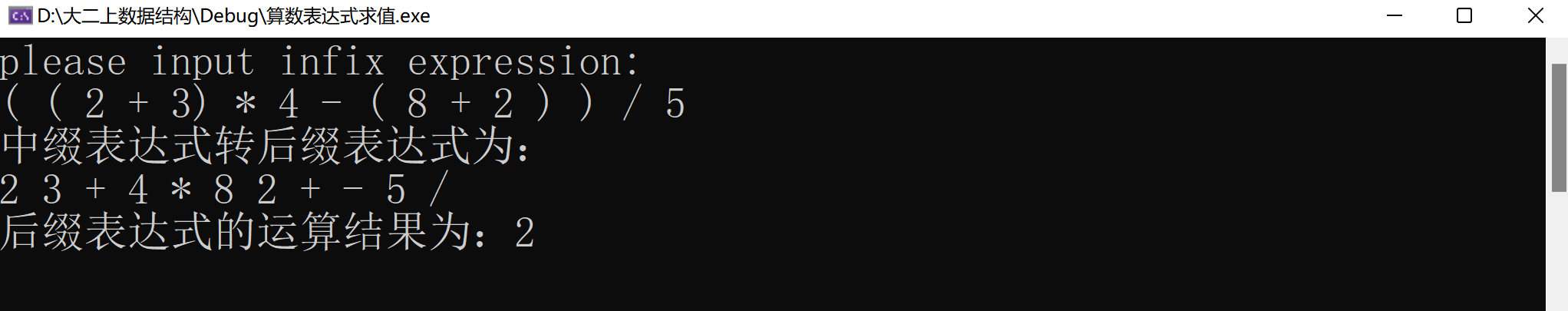
## 3.2 嵌套括号

测试输入：

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

预期输出：2 3 + 4 \* 8 2 + - 5 /

实验结果：



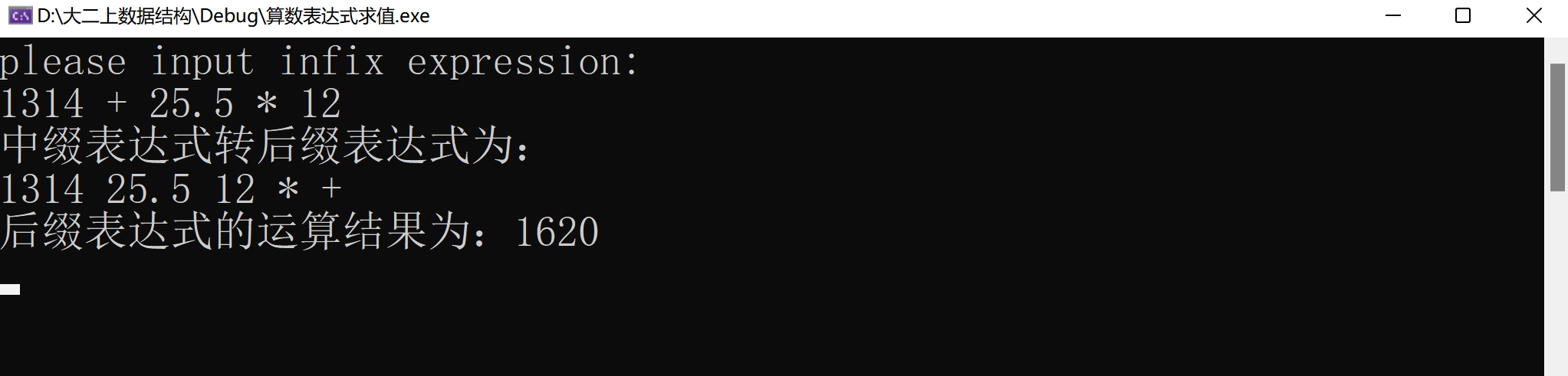
## 3.3 运算数超过1位整数且有非整数出现

测试输入：

1314 + 25.5 \* 12

预期输出：1314 25.5 12 \* +

实验结果：



## 3.4 运算数有正或负号

测试输入：

-2 \* ( +3 )

预期输出：

-2 3 \*

实验结果：



## 3.5 只有1个数字

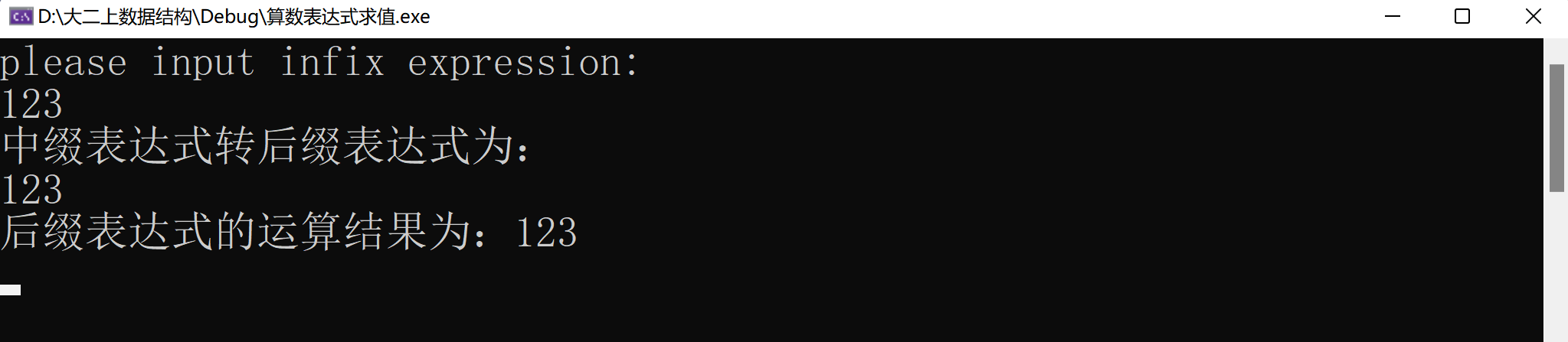
测试输入：

123

预期输出：

123

实验结果：



## 3.6 输入非法字符的情况（健壮性）

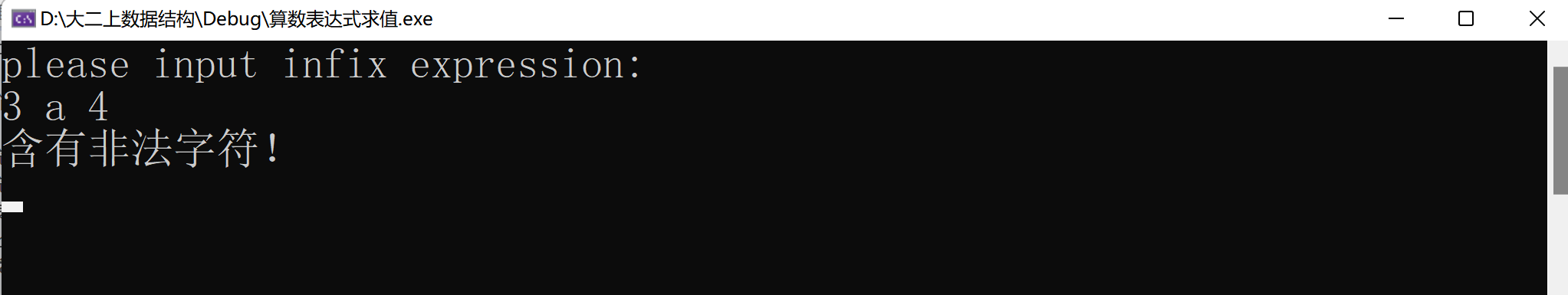
测试输入：

3 a 4

预期输出：

有错误提示

实验结果：



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，所有示例中的中缀表达式转化而成的后缀表达式与结果一致，后缀表达式的求值结果也经过计算器验算正确。

## 4.2 可使用性

本算法可以很方便地使用，中缀表达式转化后缀表达式和后缀表达式求值的功能已分别封装在两个函数中。并且该算法有良好的界面和完备的用户文档。没有使用公用变量或全局变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

该项目两个操作的时间复杂度为O(n)。

## 4.5 健壮性

本算法对于边界条件，诸如：嵌套括号、运算数超过1位整数且有非整数出现、运算数有正或负号、只有1个数字甚至有非法字符的情况也有相应的提示。