项目说明文档

**数据结构课程设计**

**——算术表达式求解**

作 者 姓 名： 吴英豪

学 号： 1953608

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

1.分析

1.1内容分析

从键盘上输入中缀算数表达式，包括括号，计算出表达式的值。此系统可进行运算的有：int范围内的整数中缀字符串算式,含小数点的中缀字符串运算。

1.2功能分析

(1)程序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。支持包括加减，乘除取余，乘方和括号等操作符，其中优先级是等于<括号<加减<乘除取余<乘方

(2)能处理单目运算符：+或-。

(3)能处理含有小数以及小数点的中缀表达式,例如0.1+0.2=

2.设计

2.1数据结构设计

因为涉及中缀字符串转换为后缀字符串的操作，结合栈后进先出的特点以及队列先进先出的特点，本系统从采用栈和队列的数据结构来解决问题。

首先是用户输入，以＇=＇为结束符，在用户输入的同时按照顺序将字符存在**队列(1)infix\_queue**中，得到中缀表达式之后借助**栈(2)char\_stack**将中缀字符串转换为后缀字符串，结果存在**队列(3)postfix\_queue**中，最后结合postfix\_queue和**栈(4)double\_queue**计算出最终的结果。

考虑到代码的复用性,使用模板来解决问题，由于本题考虑了含小数的表达式，所以最终的结果队列double\_queue中存储的数据为double类型,而用户输入的数据类型为char型

2.2类结构设计

因为涉及到中缀字符串转换为后缀字符串，所以需要栈和队列.与之相关的类有**StackNode栈结点类**,**Stack栈类**,**队列结点类QueueNode类**,**队列Queue类。**

其中，因为栈Stack类需要调用栈结点的成员,队列Queue需要调用队列结点的参数，所以将Stack设置为了StackNode的友元，将Queue设置为了QueueNode的友元。同时在字符串的转换中，栈结点和队列结点需要相互访问， 所以它们互为友元。

2.3成员与操作设计

**(1)栈结点类(StackNode)**

template<class T>

class StackNode //栈结点

{

private:

T \_data; //存放的数据

bool \_isSign; //用于判断这个符号是不是正负号

StackNode<T>\* \_next; //栈的下一个元素

public:

StackNode();

StackNode(T data,bool \_isSign=false);

~StackNode();

friend class Stack<T>;

friend class QueueNode<T>;

};

**(2)栈类(Stack)**

template<class T>

class Stack //栈

{

private:

StackNode<T>\* \_top; //栈顶指针

public:

Stack();

~Stack();

void push(const T data,bool isSign=false);

bool pop(T& data);

T getTopData()const { return (\_top->\_next) ? \_top->\_next->\_data : ' '; }; //获取栈顶元素的值

void clear();

};

**(3)队列结点类(QueueNode)**

template<class T>

class QueueNode //队结点

{

private:

T \_data;

bool \_isSign;

QueueNode<T>\* \_next;

int \_bit\_num; //只用于记录小数点所保持的位数

int \_int\_num; //用于记录整数部分的位数

public:

QueueNode();

QueueNode(T data,bool isSign=false,int bit\_num=0,int int\_num=1); //参数分别为数据、是否为正负号、小数位数、整数位数

~QueueNode();

bool isSign()const { return \_isSign; }; //判断是不是正负号

int getBitNum()const { return \_bit\_num; }; //返回小数点包含的位数

T getData()const { return \_data; }; //返回数据的内容

int getIntNum()const { return \_int\_num; };

void setData(T data) { \_data = data; };

void setBitNum(int num) { \_bit\_num = num; };

void setSign(bool sign) { \_isSign = sign; };

void setIntNum(int num) { \_int\_num = num; };

friend class Queue<T>;

friend class StackNode<T>;

};

**(4)队列类（Queue）**

template<class T>

class Queue

{

private:

QueueNode<T>\* \_top; //队头指针

QueueNode<T>\* \_last; //队尾指针

public:

Queue();

~Queue();

void push(const T data,bool isSign=false,int bit\_num=0,int int\_num=1);

bool pop(T &data);

bool pop(QueueNode<T>& node);

int getQueueLength()const; //获取当前队列的长度

T getTopData()const { return (\_top->\_next) ? \_top->\_next->\_data : ' '; }; //获取队首元素的值

bool isEmpty()const { return(\_top->\_next) ? false : true; };

void clear();

};

2.4文件设计

本系统支持windows系统和Linux系统。

(1)Windows系统

windows系统的相关文件在文件夹04\_1953608\_WuYinghao\_Windows中，

其中类的定义以及其成员函数的定义在04\_1953608\_WuYinghao.h中;main函数以及相关的函数定义在04\_1953608\_WuYinghao.cpp中.04\_1953608\_WuYinghao.exe为Windows平台下的可执行文件。

**04\_1953608\_WuYinghao.cpp和04\_1953608\_WuYinghao.h的编码格式均为**

**简体中文(GB2312)-代码页 936**

(2)Linux 系统

Linux系统的相关文件在文件夹04\_1953608\_WuYinghao\_Linux中,

其中类的定义以及其成员函数的定义在04\_1953608\_WuYinghao\_Linux.h中;main函数以及相关的函数定义在04\_1953608\_WuYinghao\_Linux.cpp中.04\_1953608\_WuYinghao.out为Linux平台下的可执行文件。

**04\_1953608\_WuYinghao\_Linux.cpp和04\_1953608\_WuYinghao\_Linux.h的编码格式均为Unicode(UTF-8 无签名)-代码页65001**

3.实现

3.1整体功能

3.1.1 系统功能总体介绍以及算法介绍

本系统通过循环输入来反复计算中缀表达式的值。用户在进入界面之后首先需要输入一次表达式，系统在判断输入是否合乎标准之后计算表达式的值，然后提示用户是否继续输入，若输入y，则系统继续工作,若输入n，则终止运算，退出系统。

具体步骤:

①当用户输入中缀字符串之后，首先将字符串保存在中缀队列infix\_queue中（先进先出）；

②对于每一个可能的运算符，都有一个栈内权值Isp和一个栈外权值Icp。

具体权值为:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | = | ^ | \* | / | + | - | ( | ) | % |
| Icp | 0 | 6 | 4 | 4 | 2 | 2 | 8 | 10 | 4 |
| Isp | 0 | 7 | 5 | 5 | 3 | 3 | 1 | 8 | 5 |

借助栈char\_stack进行字符串转换。首先把结束符=入栈

③当infix\_queue不为空时,从队首元素出队。判断其类型，若是数字、小数点或正负号（非加减运算符），则入队postfix\_queue。若是字符，则弹出栈顶元素，比较栈顶元素栈站内权值Isp和队首元素的栈外权值Icp

④Icp>Isp队首元素，进栈，结束比较

Icp==Isp则只能是栈顶元素为(，队首元素为),删除栈顶元素，结束比较

Icp<Isp 循环退栈，将栈顶元素入队postfix\_queue，继续退栈直到栈空或是

Icp>=Isp

继续执行③操作

最终得到后缀字符串，将结果存在postfix\_queue中。

⑤借助栈infix\_queue将posifix\_queue后缀字符串转换为具体结果

--具体操作如下：

队首元素出队，若是正负号，小数点或者数字，则进行运算得到数值（通过运算将char类型的数计算为double类型）然后入队，若队首元素为运算符，则从栈顶连续退出两个元素，进行相应的计算，将计算结果入栈。

循环出队直到队列为空，此时的栈顶元素就是中缀表达式最终的结果。若在循环操作中队列还没有空就遇到了无法计算的问题（如栈无法连续退出两个元素，则说明一开始输入的中缀表达式存在问题，需要报错，给用户一定的提示）

**相关说明:**

（1）输入空格和回车不会影响最终的结果,以等号=为结束符

（2）当用户输入非法中缀字符串的时候，系统会进行检测给出相应的提示

（3）

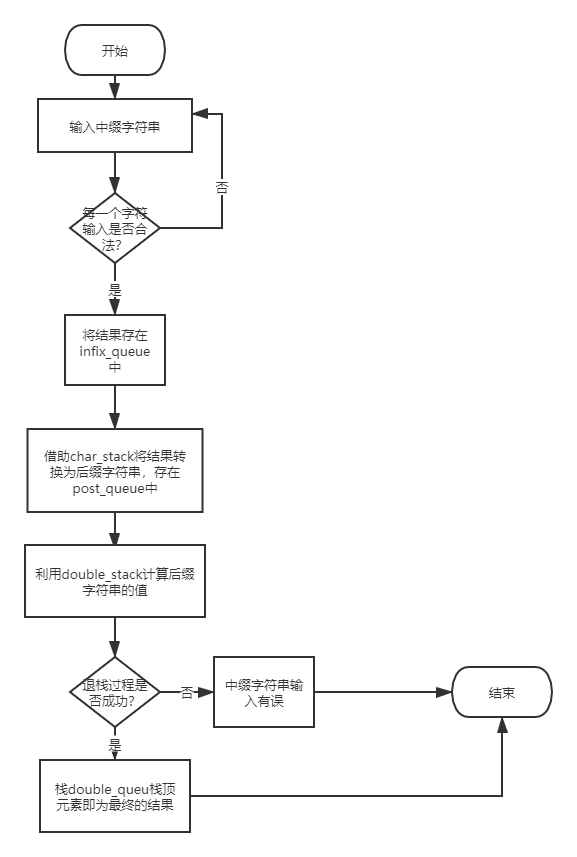
输入负数的时候，直接用正负号连接数字即可，不必额外加括号。如：

1+-2= 输入合法。

输入小数的时候,一定要输入整数部分.例如，表达式.1+2=不合法,但0.1+2=合法。同时小数输入只支持含小数点,整数部分和小数部分为多位数的输入，例如1234.56789,不支持科学计数法等含指数的小数形式。输出结果为小数的，精度为6位。

**输入相关字符(如括号或小数点),需要输入英文字符，输入中文字符会被认为不合法。**

3.1.2 系统整体功能流程图



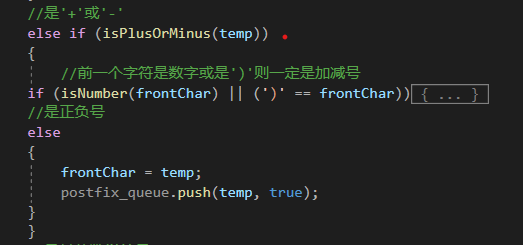
3.2 判断正负号功能的实现

3.2.1 功能说明

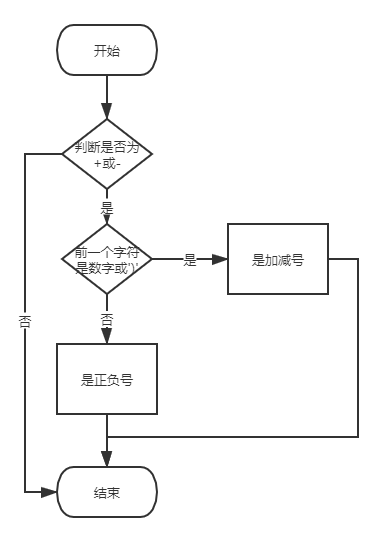
由于此系统需要支持单目运算符正负号的运算，所以在中缀字符串转换为后缀字符串的过程中需要判断+和-是普通的加减号还是正负号。若是加减号，则同乘除号等一起进行入栈出栈的操作；若是正负号，则把它当做一个数直接在post\_queue的队尾入队即可。在计算后缀表达式的值时，需要判断符号，若是加减号，则double\_stack中退栈两个元素进行加减法运算，若是正负号，则继续出队，计算这个数的值。

在具体实现中，引入bool类型的参数isSign，是正负号则取true，不是正负号则取false；当+或-前面一个元素存在且是数字，或前面一个元素的右括号’)’,那么这个符号一定是加减号，否则为正负号

3.2.2 核心代码展示



3.2.3流程图展示



3.3中缀字符串转换为后缀字符串

3.3.1 功能介绍

①当用户输入中缀字符串之后，首先将字符串保存在中缀队列infix\_queue中（先进先出）；

②对于每一个可能的运算符，都有一个栈内权值Isp和一个栈外权值Icp。

具体权值为:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | = | ^ | \* | / | + | - | ( | ) | % |
| Icp | 0 | 6 | 4 | 4 | 2 | 2 | 8 | 10 | 4 |
| Isp | 0 | 7 | 5 | 5 | 3 | 3 | 1 | 8 | 5 |

借助栈char\_stack进行字符串转换。首先把结束符=入栈

③当infix\_queue不为空时,从队首元素出队。判断其类型，若是数字、小数点或正负号（非加减运算符），则入队postfix\_queue。若是字符，则弹出栈顶元素，比较栈顶元素栈站内权值Isp和队首元素的栈外权值Icp

④Icp>Isp队首元素，进栈，结束比较

Icp==Isp则只能是栈顶元素为(，队首元素为),删除栈顶元素，结束比较

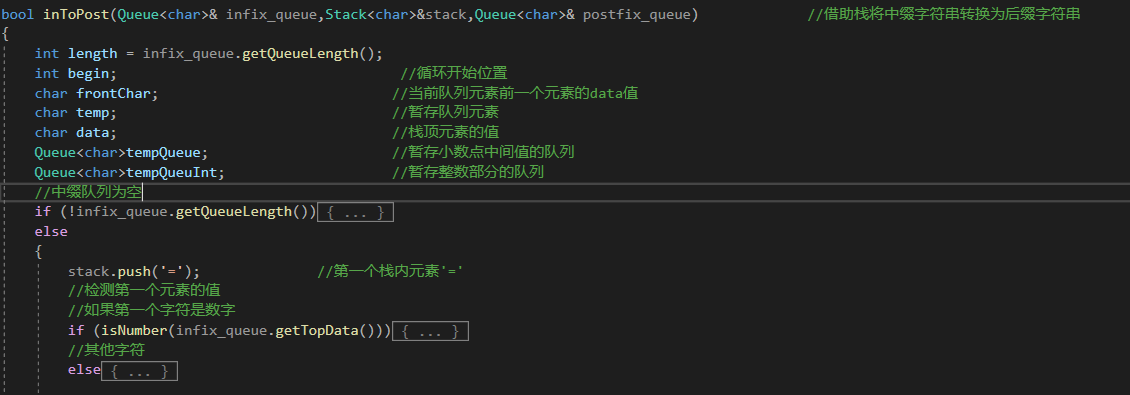
Icp<Isp 循环退栈，将栈顶元素入队postfix\_queue，继续退栈直到栈空或是

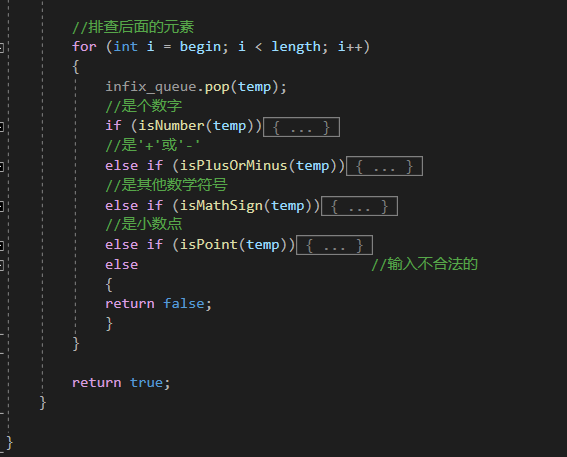
Icp>=Isp

继续执行③操作

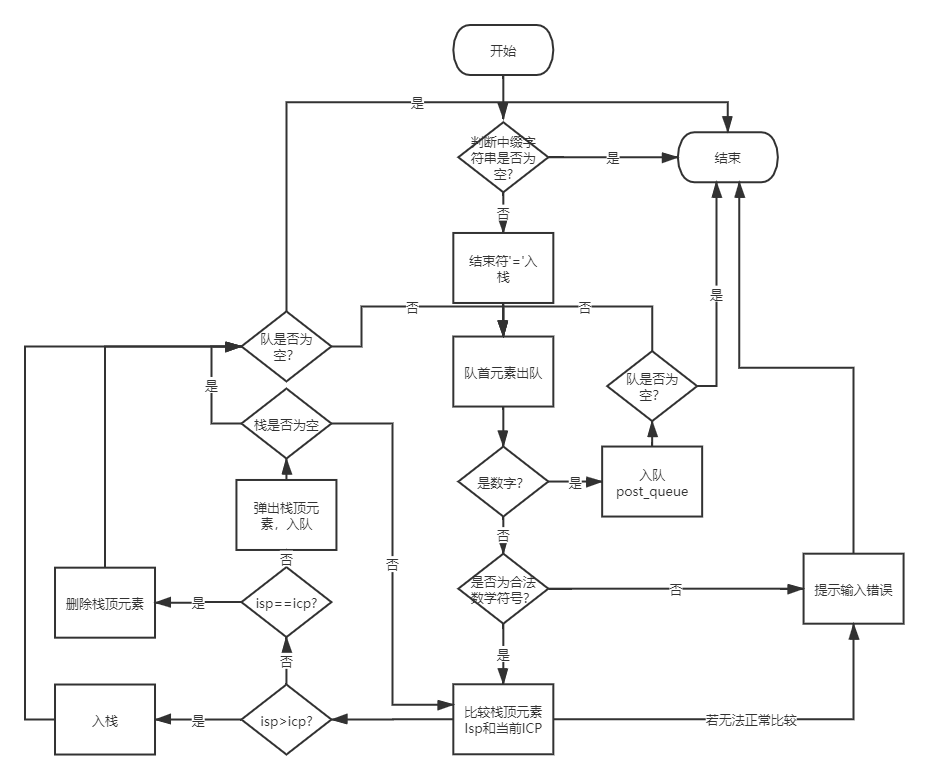
最终得到后缀字符串，将结果存在postfix\_queue中。

3.3.2 核心代码展示





3.3.3 功能流程图展示

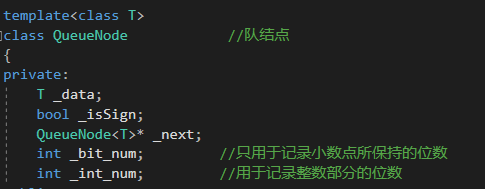


3.4 保存整数部分

3.4.1 功能介绍

输入的数据是中缀字符串，对于大于10的整数，需要考虑对应的字符串具体是多少位的。现引入\_int\_num成员变量来保存整数的具体位数,它是栈结点和队列结点的成员变量。中缀字符串转化为后缀字符串的时候，引入一个辅助变量num=0,当遇到一个数字，且这个数字前一个字符不为数字时开始计数num++;继续考察队的元素，每当遇到数字num++,当遇到非数字时将入栈的第一个元素的\_int\_num设置为num,num=0;

3.4.2 核心代码展示



infix\_queue.pop(temp);

//是个数字

if (isNumber(temp))

{

int num = 1; //记录这个数字的整数位数

char queue\_data; //暂存整数部分其他位的内容

frontChar = temp;

while (isNumber(infix\_queue.getTopData()))

{

infix\_queue.pop(queue\_data);

i++;

frontChar = queue\_data;

tempQueue.push(queue\_data);

num++;

}

postfix\_queue.push(temp, false, 0,num);

while (tempQueue.pop(queue\_data))

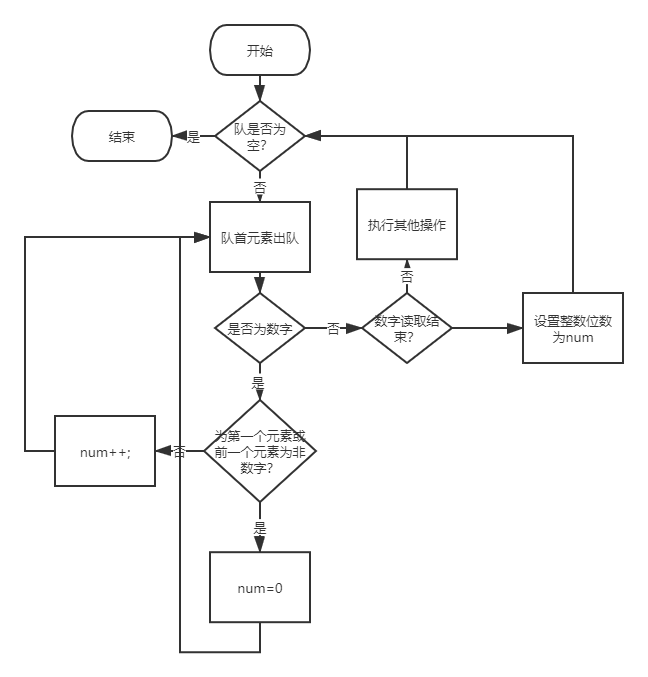
{

postfix\_queue.push(queue\_data);

}

}

3.4.3 功能流程图展示



3.5 保存小数部分

3.5.1 功能介绍

输入的数据是中缀字符串，对于含有小数部分的数，需要考虑对应的小数部分具体是多少位的。现引入\_bit\_num成员变量来保存小数的具体位数,它是栈结点和队列结点的成员变量。中缀字符串转化为后缀字符串的时候，引入一个辅助变量num=0,当遇到小数点时，开始计数，小数点入队post\_queue。继续排查队列元素，若是数字则num++，直到遇到非数字的字符时，将队列中小数点对应的\_bit\_num设置为num

3.5.2 核心代码展示

//是小数点

else if (isPoint(temp))

{

int num = 0; //用于计算小数点控制的位数

char queue\_data; //暂存小数点后的数字

while (isNumber(infix\_queue.getTopData()))

{

infix\_queue.pop(queue\_data);

i++;

frontChar = queue\_data;

tempQueue.push(queue\_data);

num++;

}

postfix\_queue.push(temp, false, num);

while (tempQueue.pop(queue\_data))

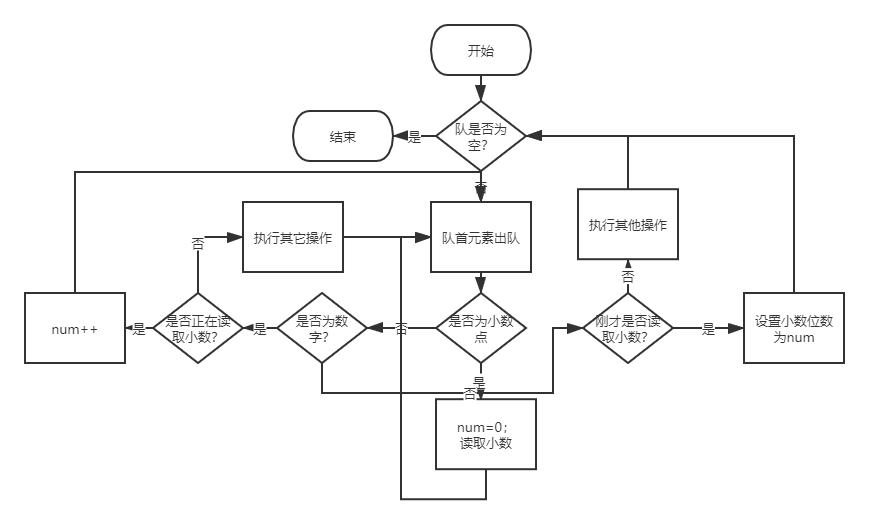
{

postfix\_queue.push(queue\_data);

}

}

3.5.3 功能流程图展示



3.6 字符数据转换为double类型

3.6.1 功能介绍

由于用户输入的是字符串，在计算的时候需要把字符类型的数据转换为double类型的数据(因为运算的数或运算结果会包含小数)。

在后缀字符串中,若某个数字的\_int\_num大于0，说明连续\_int\_num个数构成了整数部分，利用相关规则计算出整数部分。

若遇到小数点,则接下来\_bit\_num个数构成了小数部分，利用相关计算规则可以计算出小数部分。

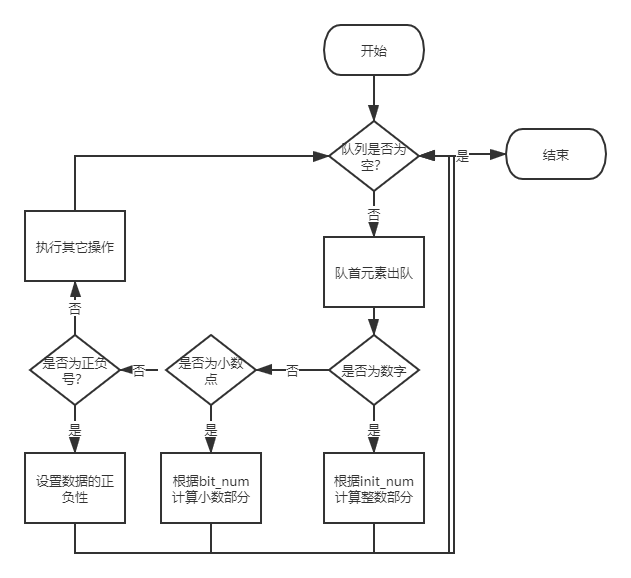
若遇到正负号，则根据正负号改变下一个数的正负性。

说明，若在计算整数部分或是小数部分的时候发现错误(如队提前为空，或遇到非数字的字符等)，说明用户输入的中缀字符串有误，进行相应的错误提示。

3.6.2 核心代码展示



3.6.3 功能流程图展示



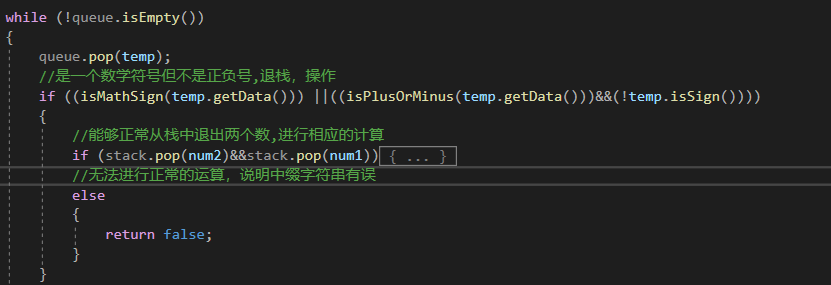
3.7 计算后缀字符串的值

3.7.1 功能介绍

得到后缀字符串之后，借助double\_stack进行计算,当队列post\_queue不为空的时候，队首元素出栈，若是数字，则入栈；若是运算符，则从栈顶连续弹出两个元素利用队首的运算符进行计算。若出栈失败或无法正常进行计算，说明用户输入的中缀表达式有误，进行一定的提示。

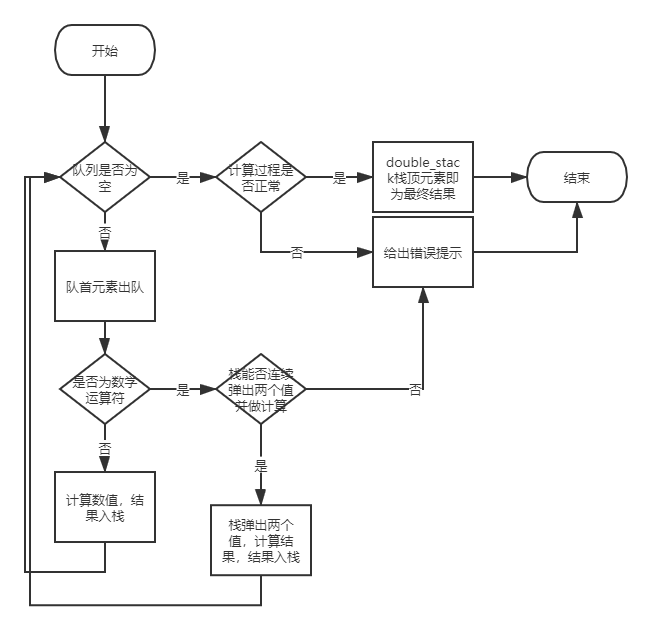
当队列为空时，栈只有一个元素，即为最终的运算结果，若栈元素大于1，说明中缀字符串输入错误，进行提示。

3.7.2 核心代码展示





3.7.3 功能流程图展示

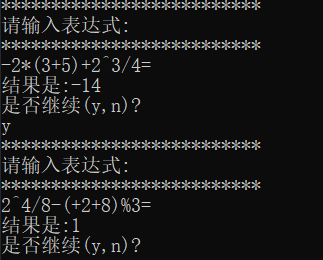


**4.测试**

**4.1功能测试**

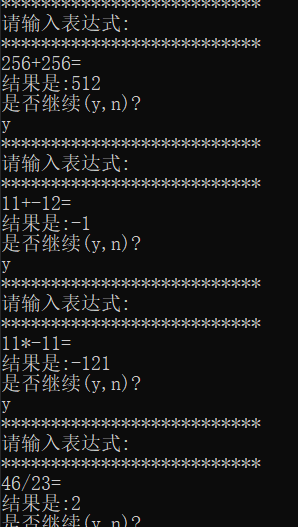
4.1.1 样例功能测试

测试题目中给出的样例



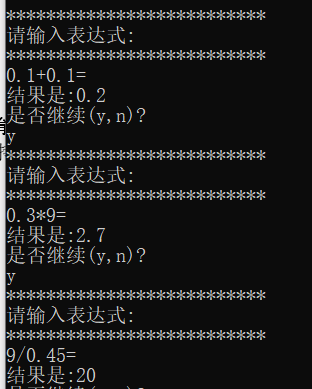
4.1.2 含10以及10以上数据的计算

由于样例没有给出含10以及10以上数据的测试数据，所以需要进行测试。



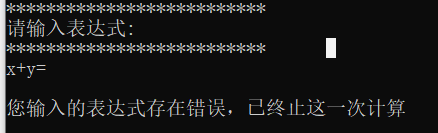
4.1.3 小数功能测试

由于此系统支持含小数点(不包括指数等其他形式)的输入，故作相应测试

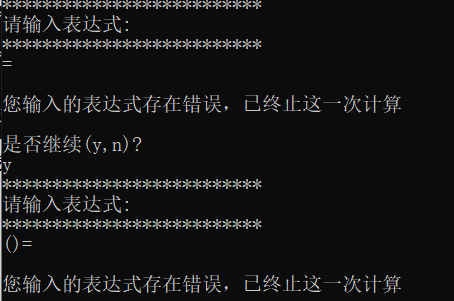


**4.2 非法表达式以及边界测试**

4.2.1 含有非法字符

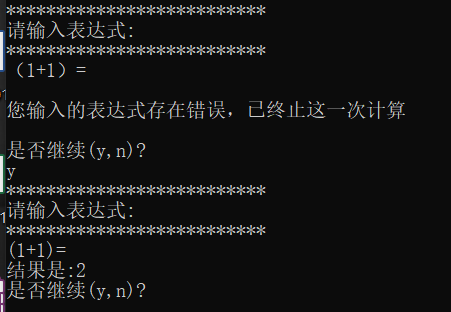


4.2.2 输入的表达式为空

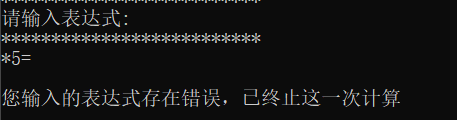


4.2.3 未按照要求输入

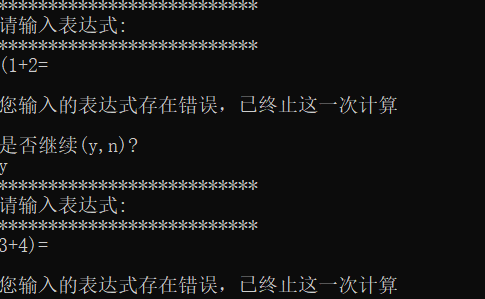
例如，输入中文的括号(注意此系统要求的括号输入格式为英文括号)



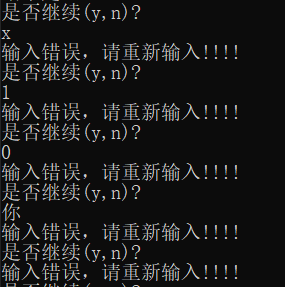
4.2.4 运算符的输入不合理



4.2.5 输入的括号个数不匹配



4.2.6 判断是否继续时输入其他数据



4.2.7 边界测试—只输入一个数据

