# 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：刘雪迪
学号：1752985
指导教师：张颖
学院/专业：软件学院/软件工程

# 一、分析

## 1.1 项目背景分析

平时我们使用的运算表达式就是中缀表达式，例如**1+3*2**，中缀表达式的特点就是：**二元运算符总是置于与之相关的两个运算对象之间**。这种表达式人读起来比较好理解，但是计算机处理起来就很麻烦，运算顺序往往因表达式的内容而定，不具规律性。所以为了方便计算机求解中缀表达式的值，我们考虑将中缀表达式转换成计算机容易理解的格式，再进行表达式的求值运算。

## 1.2 项目功能要求

相比于中缀表达式，明显具有计算优势的一种表达式是后缀表达式（又名逆波兰表达式）。后缀表达式的特点就是：每一运算符都置于其运算对象之后，且表达式中不含有括号，运算符的顺序与计算顺序相关，以上面的中缀表达式**1+2*3**为例子，转为后缀表达式就是**123*+**。对于生成的后缀表达式，我们可以很容易的根据读取的运算符进行对运算符之前的数字的计算求值。

# 二、设计

## 2.1 整体系统设计

由分析得，我们主要需要进行的操作是

**1. 将中缀表达式转化为后缀表达式，**

**2. 求后缀表达式的值**

但因为不是所有的表达式都是由简单的个位数配合双目运算符组成，除了题中所给出的单目运算符存在，本题还需要处理一个步骤：

**0. 将中缀表达式进行预处理**

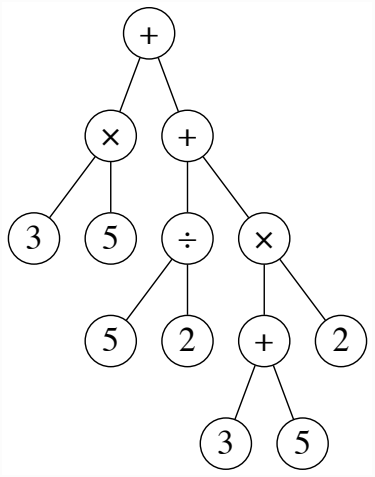
其中，需要特别处理的有以下部分：

**1. 前置单目运算符“+”、“-”的处理**

**2. 多位数的处理**

**3. 小数的处理(包含对小数做取余运算时的取整处理)**

对于预处理后的中缀表达式，将其转化成后缀表达式，有两种常见方法。我们在数据结构课程上有学到，对于一个二叉树，如果它的叶节点存放的是数字，其余节点存放操作符的话，那么这个二叉树的中序遍历序列会得到一个中缀表达式（需在某些地方手动添加括号使表达式逻辑正确），相应的，该二叉树的前序遍历序列对应该算术表达式的前缀表达式，后序遍历序列对应后缀表达式。以表达式**3×5+5÷2+(3+5)×2**为例，将其表示成二叉树形式，如下图所示：




将中缀表达式表示为二叉树形式的过程如下：

1. 找出表达式中最后进行运算的操作符，即优先级最低的操作符，作为根节点

2. 在步骤 1 找出的操作符的左侧部分找出最后进行运算的操作符，作为步骤 1 根节点的左子节点；右侧部分进行同样的操作
3. 若步骤 1 找出的操作符左侧或右侧部分只含有操作数，不含有操作符，则将该操作数作为对应的叶子节点。
4. 不断重复上述过程直到二叉树包含表达式的所有操作数和操作符

建立完毕表达式的二叉树，从叶子节点开始对二叉树进行后序遍历（左子树 -> 右子树 -> 根节点），即可得到中缀表达式对应的后缀表达式。

而二叉树的遍历分为递归法与非递归法，对于二叉树的中序遍历和后序遍历，我们都可以用**栈（Stack）**这种数据结构来实现二叉树的非递归遍历。同样的，为了方便表达式的转化，我们可以直接只用栈来实现中缀表达式转化为后缀表达式，从而避免二叉树的构建。该过程与计算中缀表达式的过程类似，使用一个栈保存操作符，一个队列结构用于保存转换得到的后缀表达式。其过程如下：

1. 从头到尾扫描表达式，若遇到操作符，则与操作符栈的栈顶元素比较优先级：如果当前操作符优先级高于栈顶元素的优先级，则压入操作符栈；否则不断弹出操作符栈顶元素压入结果队列，直到栈顶元素的优先级低于当前操作符的优先级，将当前操作符压入操作符栈
2. 若遇到操作数，则直接压入结果队列
3. 若遇到左括号，则压入操作符栈
4. 若遇到右括号，则依次弹出操作符栈顶元素压入结果队列，直到遇到左括号，将左括号弹出操作符栈

对于后缀表达式的计算，实现思路是：

1. 如果是操作数，那么将其压入"结果栈"中
2. 如果是操作符，从"结果栈"中取出两个元素，进行计算。（注意从栈中取元素的顺序和操作数的顺序是相反的） 遍历后缀表达式结束后，"结果栈"中的元素就是最终的结果。

## 2.2 数据结构设计

栈(stack)是存放数据对象的一种特殊容器，其中的数据元素按线性的逻辑次序排列，故也可定义首、末元素。不过，尽管栈结构也支持对象的插入和删除操作，但其操作的范围仅限于栈的某一特定端。也就是说，若约定新的元素只能从某一端插入其中，则反过来也只能从这一端删除已有的元素。禁止操作的另一端，称作盲端。

作为抽象数据类型，栈需要的常用操作有元素的入栈、栈顶元素的出栈以及返回栈顶元素的值。栈实现可以基于单向链表：设置一个哨兵头节点，每次入栈操作相当于在头节点后插入一个新节点，出栈则删除头节点后的节点。

由于本题需要用到多种数据类型的栈结构，所以数据结构由模板类实现。

## 2.3 成员与操作设计

### 1.节点结构体（Node）

```
template<typename T>
struct Node {
    T data;//节点数据
    Node<T> *next;//指向下一个指针
};
```

## 2. 栈类（Stack）

**私有成员：**

```
private:
    Node<T> *head;//头指针
    int _size;//栈的大小
```

**公有操作：**

```
public:
    Stack();

    int size();//返回栈的大小
    bool empty();//返回栈是否为空
    void push(T e);//插入一个元素
    //⬆踩坑：这里虽然理论上用指针传递更好，但是按照代码习惯，我们一般还是用值传递
    void pop();//弹出栈顶元素
    T top();//返回栈顶元素值
```

**核心函数：**

元素入栈：

```
template<typename T>
void Stack<T>::push(T e) {
    auto n = new Node<T>;
    n->data = e;
    n->next = this->head->next;
    this->head->next = n;
    this->_size++;
}
```

栈顶元素出栈：

```
template<typename T>
void Stack<T>::pop() {
    if (!this->empty()) {
        auto p = head->next;
        head->next = head->next->next;
        delete p;
    }
    this->_size--;
}
```

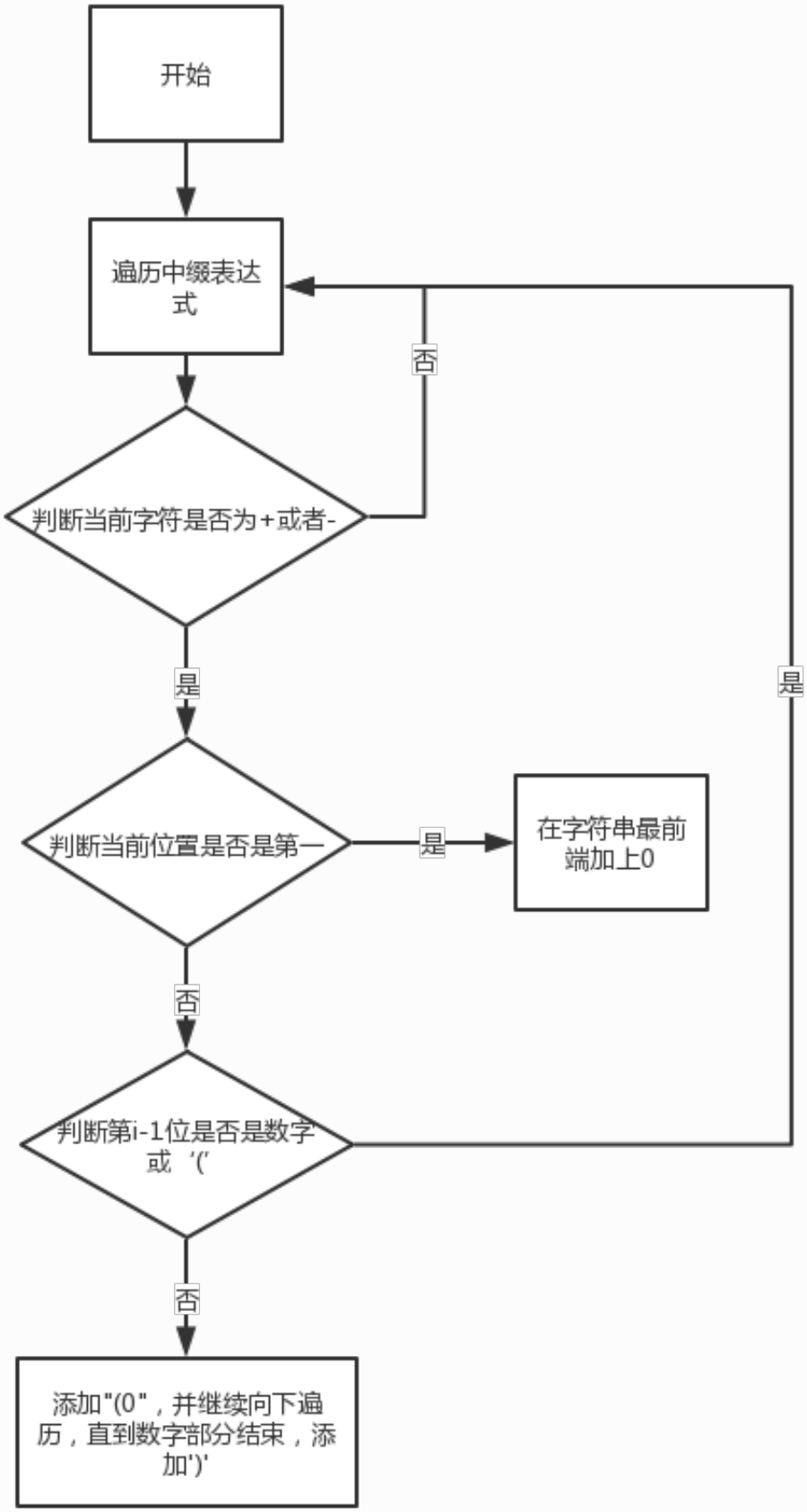
返回栈顶元素：

```
template<typename T>
T Stack<T>::top() {
    if (!this->empty())
        return this->head->next->data;
}
```

# 三、实现

## 3.1 单目运算符预处理

1. 预处理流程图流程图及说明


开始
遍历中缀表达式
判断当前字符是否为+或者-
否
是
判断当前位置是否是第一
是
在字符串最前端加上0
否
判断第i-1位是否是数字或‘(’
是
否
添加"(0"，并继续向下遍历，直到数字部分结束，添加')'

## 2. 预处理核心代码

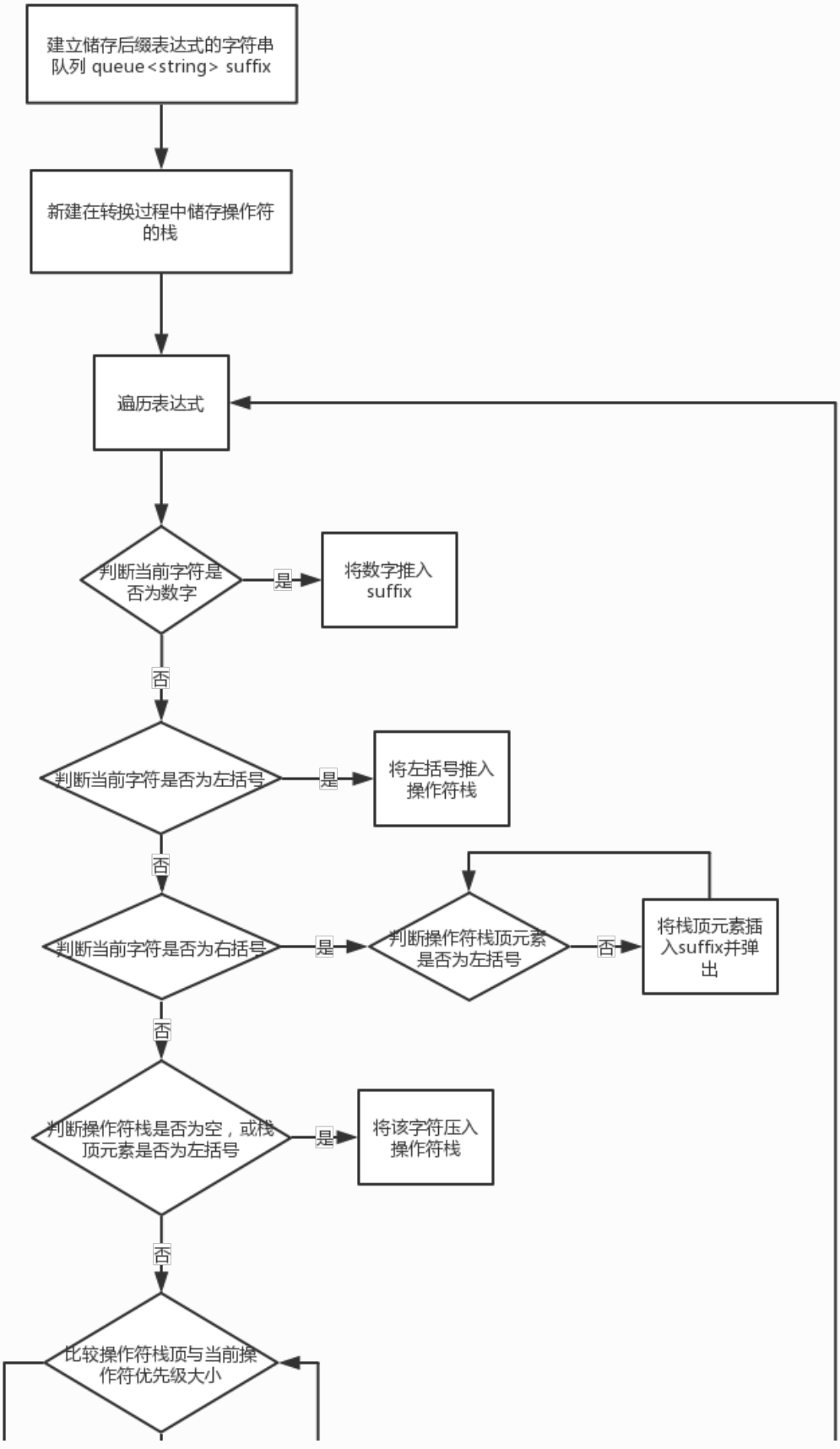
```
//处理单目运算符,返回处理后的字符串
string monoOpProcess(string s) {
    for (int i = 0; i < s.size(); ++i) {
        if (s[i] != '+' && s[i] != '-') {
            continue;
        }
        if (i == 0) {
            s.insert(i, 1, 0 + '0');
        } else if (!isNumber(s[i - 1]) && s[i - 1] != ')') {
            s.insert(i, "(0");
            for (i = i + 3; i <= s.size(); ++i) {
                if (isNumber(s[i]) || s[i] == '.') {
                    continue;
                } else {
                    s.insert(i, ")");
                    break;
                }
            }
        }
    }
    return s;
}
```

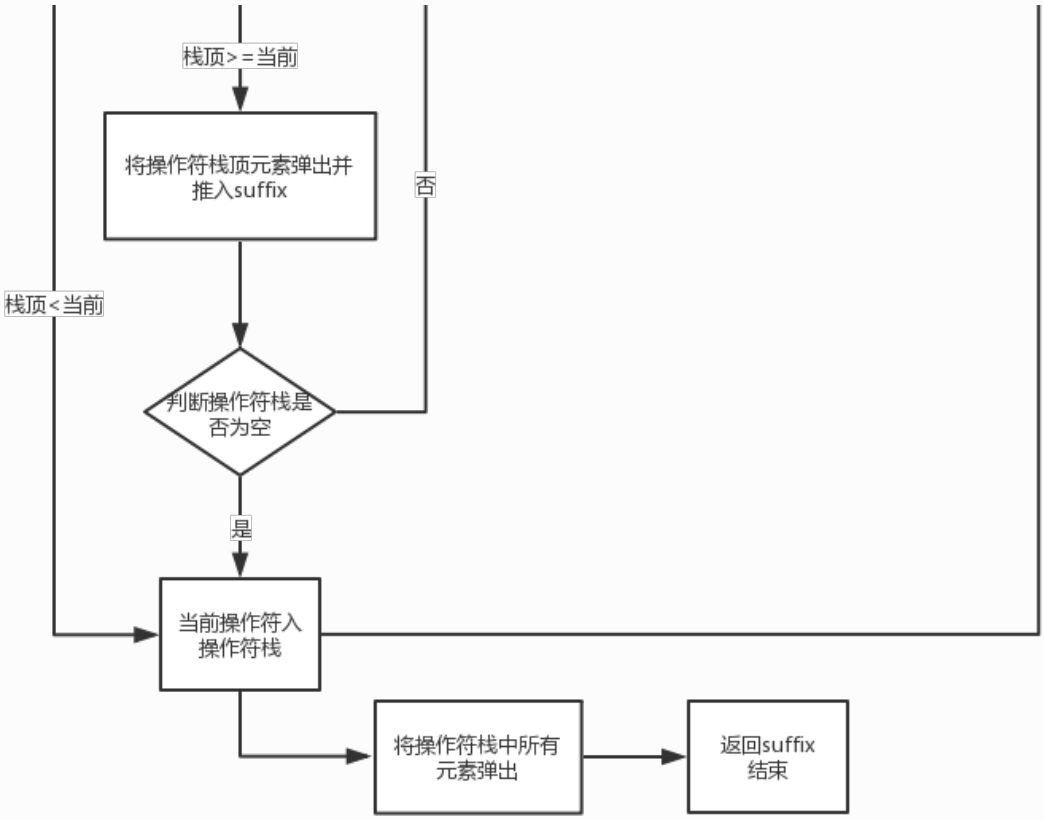
## 3. 预处理结果演示

```
输入表达式:
-2*(3+5)+2^3/4=
处理后的表达式为: 0-2*(3+5)+2^3/4
是否继续(y/n)?
y
输入表达式:
2^4/8-(+2+8)%3=
处理后的表达式为: 2^4/8-((0+2)+8)%3
```

## 3.2 中缀表达式转化为后缀表达式

### 1. 转化流程图及说明






## 2. 转化函数核心代码

```
queue<string> getSuffix(string s) {
    queue<string> suffix;//生成的后缀表达式
    auto op = new Stack<char>;
    for (int i = 0; i < s.size(); ++i) {
        if (isNumber(s[i])) {
            string num = string(1, s[i]);
            int j = i + 1;
            for (; j < s.size(); ++j) {
                if (!isNumber(s[j]) && s[j] != '.') {
                    break;
                }
                num = num + s[j];
                i = j;
            }
            suffix.push(num);
        } else if (s[i] == '(') {
            op->push(s[i]);
        } else if (s[i] == ')') {
            while (op->top() != '(') {
                suffix.push(string(1, op->top()));
                op->pop();
            }
            op->pop();//把左括号也出栈
        } else {
            if (op->empty() || op->top() == '(') {
```

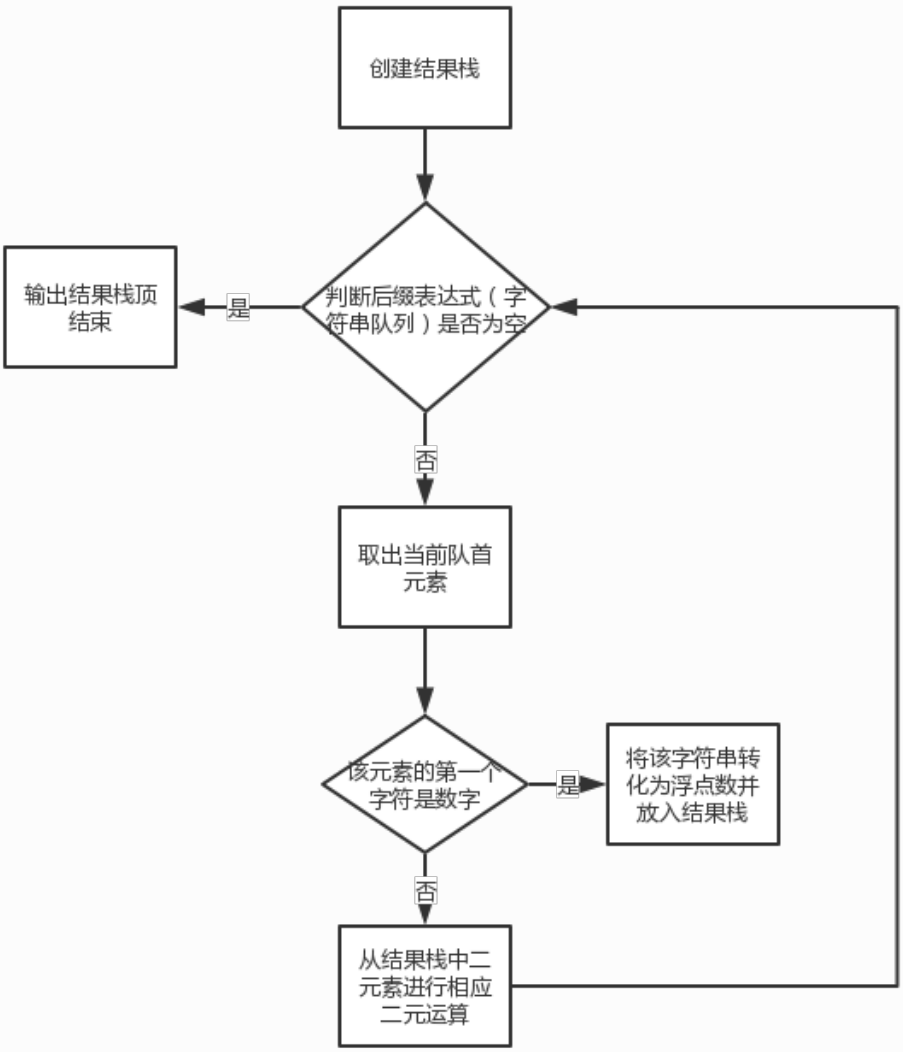
```
                op->push(s[i]);
            } else {
                while (cmp(op->top(), s[i])) {
                    suffix.push(string(1, op->top()));
                    op->pop();
                    if (op->empty()) {
                        break;
                    }
                }
                op->push(s[i]);
            }
        }
    }
    while (!op->empty()) {
        suffix.push(string(1, op->top()));
        op->pop();
    }
    return suffix;
}
```

## 3. 后缀表达式示意图

```
输入表达式:
-2*(3+5)+2^3/4=
处理后的表达式为: 0-2*(3+5)+2^3/4
转化后的后缀表达式: 0235+*-23^4/+
是否继续(y/n)?
2^4/8-(+2+8)%3
输入表达式:
处理后的表达式为: ^4/8-((0+2)+8)%
转化后的后缀表达式: 4^8/02+8+%-
```

## 3.3 后缀表达式求值的实现

### 1. 求值示意图及说明




特别需要注意的是，由于栈这种数据结构遵循**后进先出**的原则，所以对于从栈顶取出的两个数字进行二元运算时，应注意运算数的排列顺序。

### 2. 求值核心代码

```
double calculate(double a, double b, string op) {
    if (op == "+") {
        return a + b;
    } else if (op == "-") {
        return a - b;
    } else if (op == "*") {
        return a * b;
```

```
    } else if (op == "/") {
        if (b == 0) {
            cout << "除数不能为0，您输入的表达式有误" << endl;
            exit(-1);
        } else {
            return a / b;
        }
    } else if (op == "%") {
        return (int(a) % int(b));
    } else if (op == "^") {
        return pow(a, b);
    }
}
```

```
double getAns(queue<string> s) {
    Stack<double> ans;//结果栈
    while (!s.empty()) {
        string temp = s.front();
        s.pop();
        if (isnumber(temp[0])) {
            //当前取出是数字
            ans.push(atof(temp.c_str()));
        } else if(temp == "(" ||temp ==")"){
            cout << "您输入的表达式括号不匹配" << endl;
            exit(-1);
        } else{
            double b = ans.top();
            ans.pop();
            double a = ans.top();
            ans.pop();
            ans.push(calculate(a, b, temp));
        }
    }
    return ans.top();
}
```

## 3. 求值示意图

```
输入表达式:
2^4/8-(+2+8)%3=
1
是否继续(y/n)?
y
输入表达式:
-2*(3+5)+2^3/4=
-14
```

## 3.4 其他细节功能的实现及其代码实现

### 1. 操作符优先级比较

使用的数据结构：**map**

将运算符设为map的键，给定不同优先级的运算符不同的值。

核心代码：

```
map<char, int> optionWeight = {
        {'+', 1},
        {'-', 1},
        {'*', 2},
        {'/', 2},
        {'%', 3},
        {'^', 4}
};//操作符的优先级
```

```
//在当前操作符的优先级高于低于或等于栈顶操作符时，需要栈顶出栈
//使用时应cmp（栈顶，当前）
bool cmp(char a, char b) {
    return optionWeight[a] - optionWeight[b] >= 0;
}
```

### 2. 整体系统设计

```
 string s;//表达式
    while (1) {
        cout << "输入表达式：\n";
        cin >> s;
        s.pop_back();//删除等号做处理
        queue<string> suffix;//后缀表达式
        s = monoOpProcess(s);//处理单目运算符
        suffix = getSuffix(s);//转后缀表达式
        cout << getAns(suffix) << endl;
        cout << "是否继续(y/n)?" << endl;
        char op;
        cin >> op;
        if (op == 'y') {
            continue;
        } else if (op == 'n') {
            break;
        }
    }
```

# 四、测试

## 4.1 功能测试

### 1. 普通四则运算测试

**测试用例：**

12+3678*256-2670/2=

**预期结果：**

940245

**实验结果：**

```
输入表达式:
12+3678*256-2670/2=
940245
是否继续(y/n)?
```

### 2. 含括号的四则运算

**测试用例：**

278*((2567-2673)+111)=

**预期结果：**

1390

**实验结果：**

```
输入表达式:
278*((2567-2673)+111)=
1390
```

### 3. 含有乘方/取余的运算

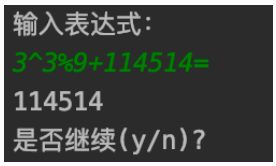
**测试用例：**

3^3%9+114514=

**预期结果：**

114514

**实验结果：**




## 4. 包含单目运算符的运算

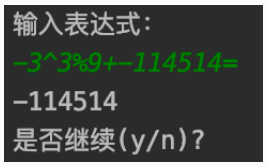
**测试用例：**

-3^3%9+-114514=

**预期结果：**

-114514

**实验结果：**




## 5. 包含浮点数的运算式

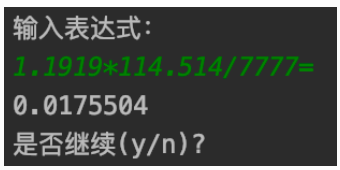
**测试用例：**

1.1919*114.514/7777=

**预期结果：**

0.0175504

**实验结果：**



## 4.2 边界测试

### 1. 输入的表达式不含运算符

**测试用例：**

114514=

**预期结果：**

114514

**实验结果：**


### 2. 取余运算含有浮点数需要去整

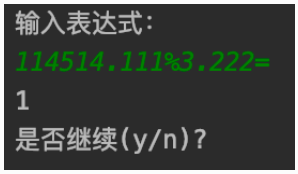
**测试用例：**

114514.111%3.222=

**预期结果：**

1

**实验结果：**



## 4.3 出错测试

### 1. 算术表达式中左右括号数量不匹配

**测试用例：**

278*(((2567-2673)+111)=

**预期结果：**

您输入的表达式括号不匹配

**实验结果：**


### 2. 运算式中除数为0

**测试用例：**

11111/0=

2678%0=

**预期结果：**

除数不能为0，您输入的表达式有误

**实验结果：**