**项目说明文档**

**数据结构课程设计**

**——表达式转换**

作 者 姓 名： 张梓瀚

学 号： 2051943

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 - 3 -](#_Toc121208998)

[1.1项目背景 - 3 -](#_Toc121208999)

[1.2 项目需求分析 - 3 -](#_Toc121209000)

[1.3 项目要求 - 3 -](#_Toc121209001)

[1.3.1 功能要求 - 3 -](#_Toc121209002)

[1.3.2 输入格式 - 3 -](#_Toc121209003)

[1.3.3 输出格式 - 3 -](#_Toc121209004)

[1.3.4 项目示例 - 3 -](#_Toc121209005)

[2 项目设计 - 4 -](#_Toc121209006)

[2.1 数据结构设计 - 4 -](#_Toc121209007)

[2.2 类设计 - 4 -](#_Toc121209008)

[2.2.1 链表类（List） - 4 -](#_Toc121209009)

[2.2.2 栈类（MyStack） - 4 -](#_Toc121209010)

[2.3 中缀表达式转后缀表达式算法 - 5 -](#_Toc121209011)

[2.3.1 主要思想 - 5 -](#_Toc121209012)

[2.3.2代码 - 5 -](#_Toc121209013)

[3 项目测试 - 7 -](#_Toc121209014)

[3.1 一般情况 - 7 -](#_Toc121209015)

[3.2 嵌套括号的情况 - 7 -](#_Toc121209016)

[3.3 多位数字且有小数出现的情况 - 7 -](#_Toc121209017)

[3.4 运算数有正负号的情况 - 7 -](#_Toc121209018)

[3.5 只有一个数字的情况 - 8 -](#_Toc121209019)

[3.6 有非法符号的情况 - 8 -](#_Toc121209020)

# 1 项目分析

## 1.1项目背景

栈是计算机中非常常用的一种数据结构，在数据存储、程序中断等场景中有着丰富的应用。计算机采用先进后出的方式处理表达式，因此后缀表达式相较于中缀表达式对于计算机而言更易理解，故中缀表达式转后缀表达式有着重要的意义。

## 1.2 项目需求分析

针对于中缀表达式转后缀表达式的系统，应当满足以下需求：

* 功能完善

系统应当考虑到表达式的各种情况，如带括号、带小数、带负数等。

* 执行效率高

系统采用的栈结构应当能高效增删结点，从而保证较低的时间复杂度。

* 健壮性

系统应当考虑当所输入的表达式为空或出现非法符号的情况。

## 1.3 项目要求

### 1.3.1 功能要求

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。请设计程序将中缀表达式转换成为后缀表达式。

### 1.3.2 输入格式

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）。

### 1.3.3 输出格式

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。

### 1.3.4 项目示例

|  |  |  |  |
| --- | --- | --- | --- |
| 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 项目设计

## 2.1 数据结构设计

如上述功能分析所述，该项目本质上是要实现一个栈数据结构，并在此结构上实现中缀表达式转后缀表达式算法。栈是计算机程序中常用的存储结构，根据存储方式有顺序栈、链式栈、堆栈等形式，在本程序中为实现高效的增删结点并提高代码复用，在前列项目的链表类基础上编写了一个链式栈。

## 2.2 类设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

为了实现代码的复用性，本系统利用了上题中实现过的**链表结点类**与**链表类**。

### 2.2.1 链表类（List）

链表的结点中存储的数据有链表数据data和后继结点next。为提高代码的泛用性，本题中ListNode类采用模板类实现。

### 2.2.2 栈类（MyStack）

//基于List类实现的链式栈

template<typename T>

class myStack : private List<T> {

public:

myStack();

//向链表头部（栈顶）插入结点

void push(const T& data);

//返回第一个结点的数据

const T top();

//删除第一个结点

void pop();

//判断栈是否为空

bool empty();

};

## 2.3 中缀表达式转后缀表达式算法

### 2.3.1 主要思想

初始化栈为空，依次读入原表达式字符并比较其与栈顶字符的优先级，若栈顶字符优先级低则将栈外字符压栈，否则不断将栈顶出栈直到栈顶字符优先级高于栈外字符。

### 2.3.2代码

const char\* infix\_to\_suffix(const char\* infix)

{

char\* suffix = new char[MAX\_LEN + 1];

char last = '1'; //last记录上个符号，简化判断

bool fir = false, is\_opt = false; // 判断首个符号和负号是否为运算符的标志

int cnt = 0; //记录在后缀表达式中的位置

myStack<char>stk;

for (int i = 0; infix[i] != '\0'; i++) {

char c = infix[i];

//扫描到句子末尾，结束循环

if (c == '\0') {

break;

}

else if (c == ' ') {

last = c;

continue;

}

//对于小数点和数字，直接输出

else if (isdigit(c) || c == '.') {

if (isdigit(c))

is\_opt = true;

//判断当前的数字是否和前几个符号同属一个运算数

if (!(!fir || last == '-' || isdigit(last) || last == '.'))

suffix[cnt++] = ' ';

suffix[cnt++] = c;

fir = true;

last = c;

continue;

}

//右括号，将栈顶符号出栈直到对应的左括号出栈

else if (c == ')') {

while (!stk.empty() && stk.top() != '(') {

suffix[cnt++] = ' ';

suffix[cnt++] = stk.top();

last = stk.top();

stk.pop();

}

stk.pop();

}

//左括号直接压栈

else if (c == '(') {

stk.push(c);

}

else {

if ((c == '+' || c == '-') && isdigit(infix[i + 1])) {

//对负数的特殊处理

if (c == '-') {

//若负号为运算符，输出空格

if (is\_opt)

suffix[cnt++] = ' ';

suffix[cnt++] = c;

last = c;

}

continue;

}

while (!stk.empty() && !cmp(c, stk.top())) {

suffix[cnt++] = ' ';

suffix[cnt++] = stk.top();

last = stk.top();

stk.pop();

}

stk.push(c);

}

}

//若栈内还有剩余运算符或数字则全部输出

while (!stk.empty()) {

suffix[cnt++] = ' ';

suffix[cnt++] = stk.top();

stk.pop();

}

suffix[cnt] = '\0';

return suffix;

}

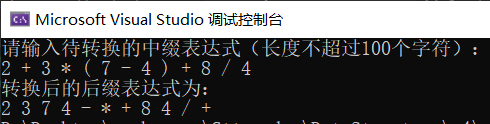
# 3 项目测试

## 3.1 一般情况

测试用例：2 + 3 \* ( 7 - 4 ) + 8 / 4

预期结果：2 3 7 4 - \* + 8 4 / +

实验结果：

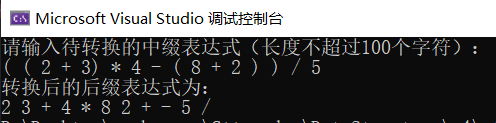


## 3.2 嵌套括号的情况

测试用例：( ( 2 + 3) \* 4 - ( 8 + 2 ) ) / 5

预期结果：2 3 + 4 \* 8 2 + - 5 /

实验结果：

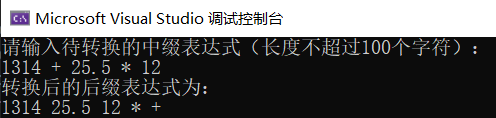


## 3.3 多位数字且有小数出现的情况

测试用例：1314 + 25.5 \* 12

预期结果：1314 25.5 12 \* +

实验结果：

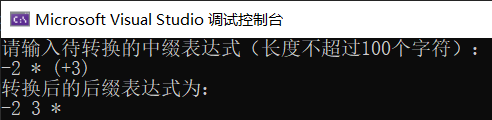


## 3.4 运算数有正负号的情况

测试用例：-2 \* ( +3 )

预期结果：-2 3 \*

实验结果：

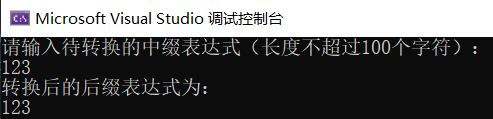


## 3.5 只有一个数字的情况

测试用例：123

预期结果：123

实验结果：



## 3.6 有非法符号的情况

测试用例：2 \* ( 3 – 4) **（减号为全角）**

预期结果：报错

实验结果：

