项目说明文档

数据结构课程设计

——表达式转换

作 者 姓 名： 何慧琳

学 号： 2152343

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 背景分析

后缀表达式，又称逆波兰式，指的是不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行(不再考虑运算符的优先规则)。后缀表达式的特点就是计算机运算非常方便，需要用到栈；计算机处理过程只需要顺序读入，如果遇到数字，则放入栈中，如果是运算符，则将两个栈中数字取出进行运算。此程序虽不用将操作数存入栈中而是直接输出，但也有助于对计算机内部运算原理有个更深的理解。

## 1.2 功能分析

作为一个简易的表达式转换程序，以防用户只是想知道中缀表达式如何转换成后缀表达式，即看看后缀表达式的转换样例，该程序提供了样例版，用户只需按回车键就能看到5个表达式及其后缀表达式转换；如果用户想进行中缀表达式转换，则可选择另一版本自行输入表达式得到转换后的后缀表达式。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统在转换表达式时要求对运算符的优先级判断和存储，其取出符合先进后出的规则，所以采用栈结构进行存储，所以定义了栈类（Stack），为了避免开辟空间的不确定性，采用链式栈结构，所以还定义了一个栈的结点类（Stack Node），在其中将Stack声明为友元使Stack类能访问StackNode的私有成员。为了实现表达式读入、存储、转换的功能，创建了一个表达式转换类（expression\_in\_to\_post）。

## 2.2 类结构设计

**栈结点类（StackNode）**

和链表结点的定义相同，栈的结点也由两个要素组成：结点数值和结点链接指针。

**私有成员及操作：**

Type data; //结点数据

StackNode<Type>\* link; //结点链指针

StackNode(Type d = 0, StackNode<Type>\* l = NULL) : data(d), link(l) { }//结点构造函数

**栈类（Stack）**

栈为了实现先进后出的功能，其私有成员只需要一个StackNode类型的栈顶指针即可。作为一个存储结构，栈应当能够实现插入、删除、取栈顶元素、置空栈、判断栈是否为空的功能，所以有如下栈的具体结构。

**私有成员：**

StackNode<Type>\* top; //栈顶指针

**公有操作：**

Stack() : top(NULL) { }//栈构造函数

~Stack();//栈析构函数

void Push(const Type& item);//将data为的item结点入栈

Type Pop();//将栈顶结点删除

Type Top();//取栈顶元素

void MakeEmpty(); //将栈置空

int IsEmpty() const//判断栈是否为空

**表达式转换类（expression\_in\_to\_post）**

如前面功能分析所提到的，表达式转换首先需要读取并保存表达式，于是定义了函数read负责读表达式字符串，及成员char\* expression\_str保存读到的字符串。接下来就是主要功能中缀转后缀的实现，主要的实现由conversion()来完成，其中判断一个运算符是否入栈时要比较它和栈中运算符优先级的大小，所以另外定义了一个函数optr\_enStack(char opt)负责对运算符的处理，在conversion()函数中调用optr\_enStack(char opt)函数从而实现完整的表达式转换功能。

**私有成员：**

struct optr\_priority {

char optr;

int priority;

optr\_priority(char opt)//构造函数

{

optr = opt;

switch (opt) {

case '+':

priority = 2;

break;

case '-':

priority = 2;

break;

case '\*':

priority = 3;

break;

case '/':

priority = 3;

break;

case '(':

priority = 4;

break;

case ')':

priority = 4;

break;

default:

break;

}

}

};

char\* expression\_str;//算数表达式字符串

Stack<optr\_priority> optrs;//存运算符的栈

**公有操作：**

expression\_in\_to\_post(char\* str=NULL) :expression\_str(str) {};//表达式类构造函数

void read();//读入表达式字符串，初始化expression\_str

void conversion();//实现中缀转后缀功能

void optr\_enStack(char opt);//将读到的运算符入栈

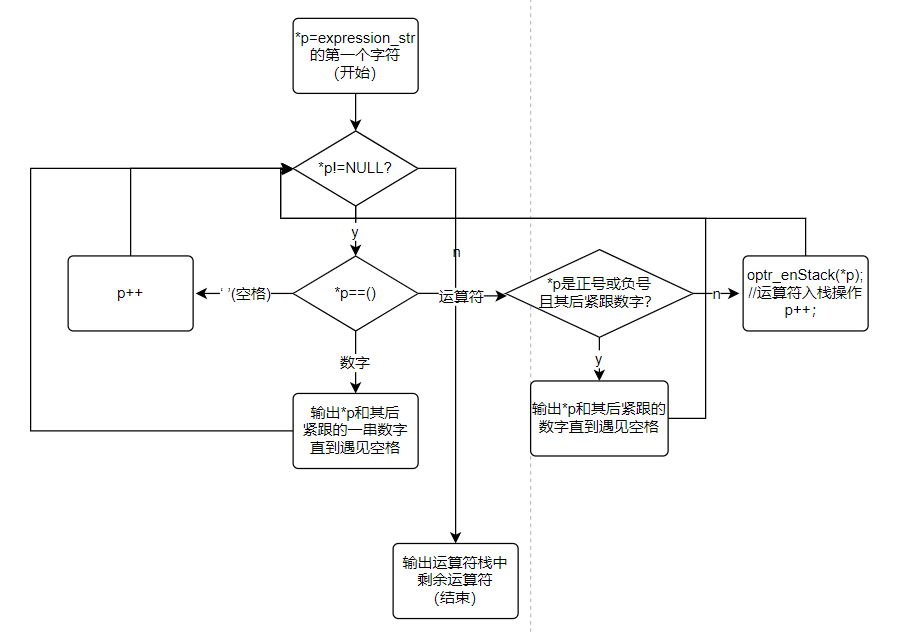
## 2.4 系统设计

系统首先调用main()函数实现对屏幕的初始化，由用户选择使用自动版本还是手动输入版本。在main()函数中调用expression\_in\_to\_post类中的函数从而实现表达式的输入、表达式转换等功能。

# 3 实现

## 表达式转换的实现

### 表达式转换功能流程图



### 表达式转换功能核心代码

char\* p = expression\_str;//扫描指针

int first=1;//是否是第一个算数的标记，1为是，0为不是，不是则在前面输出空格

while (\*p) {

if (\*p == ' ')

p++;

else if (\*p >= '0' && \*p <= '9') {

if (!first)//非第一个数的数前面都输出一个空格

cout << " ";

while (\*p&&\*p != ' ') {//遇到小数点也会输出

cout << \*p;

p++;

}

first = 0;//第一个数已访问过，first标记置零

}

else {

if ((\*p == '+' || \*p == '-') && (\*(p + 1) >= '0' && \*(p + 1) <= '9')) {

if (!first)//非第一个数的数前面都输出一个空格

cout << " ";

if (\*p == '-')//为负号则输出，为正好不输出

cout << "-";

p++;

while (\*p && \*p != ' ') {//输出数字

cout << \*p;

p++;

}

first = 0;//第一个数已访问过，first标记置零

}

else {

optr\_enStack(\*p);//运算符入栈操作

p++;

}

}

}

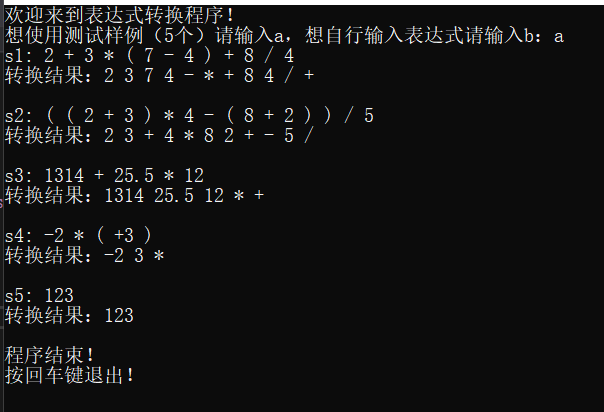
while (!optrs.IsEmpty()) {

cout << " "<<optrs.Pop().optr;

}

cout << endl;

### 表达式转换功能截屏示例



# 4 测试

## 4.1 功能测试

### 4.1.1 自动样例版本功能测试

**测试用例**：2 + 3 \* ( 7 - 4 ) + 8 / 4

**预期结果**：2 3 7 4 - \* + 8 4 / +

**实验结果**

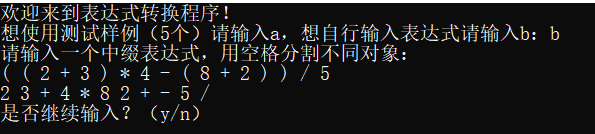
****

### 4.1.2 手动输入版本功能测试

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

**预期结果：**2 3 + 4 \* 8 2 + - 5 /

**实验结果：**



## 4.2 边界测试

### 4.2.1 初始化仅输入一个数

**测试用例：**123

**预期结果：**123

**实验结果：**

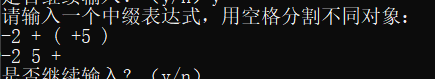


### 4.2.2 输入负数或带正号的正数

**测试用例：**-2 + （ +5 ）

**预期结果：**-2 5 +

**实验结果：**

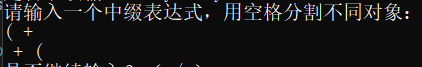


### 4.2.3 初始化仅输入运算符

**测试用例：**（ +

**预期结果：**程序不崩溃。

**实验结果：**



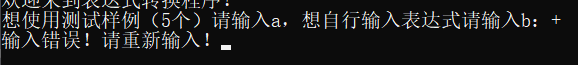
## 4.3 出错测试

### 4.3.1 选择版本和是否继续输入时输入错误

**测试用例：**输入不正确的字符

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

****

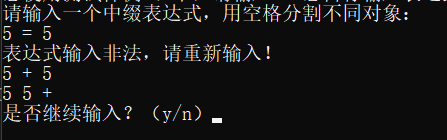
****

### 4.3.2 操作码错误

**测试用例：**输入表达式错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

****