项目说明文档

数据结构课程设计

——表达式转换

作 者 姓 名： 祝新元

学 号： 1751629

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 分析 1](#_Toc531347429)

[1.1 背景分析 1](#_Toc531347430)

[1.2 功能分析 1](#_Toc531347431)

[2 设计 2](#_Toc531347432)

[2.1 数据结构设计 2](#_Toc531347433)

[2.2 类 2](#_Toc531347434)

[2.3 算法设计 3](#_Toc531347435)

[3 实现 3](#_Toc531347436)

[3.1 判断是否是运算符 3](#_Toc531347437)

[3.1.1 代码实现 3](#_Toc531347438)

[3.2 计算栈内优先数 3](#_Toc531347439)

[3.2.1 代码实现 3](#_Toc531347440)

[3.3 计算栈外优先数 4](#_Toc531347441)

[3.3.1 代码实现 4](#_Toc531347442)

[3.4 核心算法 4](#_Toc531347443)

[3.4.1 核心算法 4](#_Toc531347444)

[4 测试 6](#_Toc531347445)

[4.1 测试用例测试 6](#_Toc531347446)

[4.1.1正常测试6种运算符 6](#_Toc531347447)

[4.1.2嵌套括号 7](#_Toc531347448)

[4.1.3运算数超过1位整数且有非整数出现 8](#_Toc531347449)

[4.1.4运算数有正或负号 9](#_Toc531347450)

[4.1.5只有1个数字 9](#_Toc531347451)

[4.2 其他测试 10](#_Toc531347452)

[4.2.1 只有运算符 10](#_Toc531347453)

# 1 分析

## 1.1 背景分析

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.输入说明：在一行中输入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）。

2.输出说明：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 功能分析

要将中缀表达式转换为后缀表达式，首先要明白后缀表达式的形式。后缀表达式即为二元运算符位于运算数后面。例如中缀表达式为2 + 3 \* ( 7 – 4 ) + 8 / 4，表示成后缀形式就变成了2 3 7 4 - \* + 8 4 / +。此外还要考虑有括号的形式，因为括号也会改变运算顺序。

# 2 设计

## 2.1 数据结构设计

在中缀表达式中操作符的优先级和括号使得求值过程复杂化，把它转化为后缀表达式，可化简求值过程。为了实现这种转换，需要考虑各种操作符的优先级。不同操作符应有不同的栈内优先级数和栈外优先级数。

为了实现对于输出的控制，采用栈这种数据结构来储存，通过进出栈能够轻松实现转换。

## 2.2 类

类采用简单的栈SeqStack。

template<class T>

class SeqStack

{

public:

SeqStack(int sz=50);

~SeqStack(){delete[]elements;}

void Push(const T& x); //入栈

bool Pop(T& x); //通过x得到退出的元素的值

bool getTop(T& x)const; //得到栈顶元素

bool IsEmpty()const{return(top==-1)?true:false;}//栈是否为空

bool IsFull()const{return(top==maxSize-1)?true:false;}

int getSize()const{return top+1;}

void MakeEmpty(){top=-1;}

private:

T \*elements; //保存栈内元素的数组

int top; //头指针

int maxSize; //最大大小

void overflowProvess(); //栈的溢出处理

};

## 2.3 算法设计

#的栈内栈外优先级都为0，左括号的栈外优先级最高，他一来到立即进栈，但当他进入栈中后，其栈内优先级变得极低，以便括号内的其他操作数进栈。其他操作符进栈后优先数都上升1，这样可体现在中缀表达式中相同优先级的操作符自左向右计算的要求，让位于栈顶的操作符先退栈并输出。操作符优先数相等的情况只出现在括号配对时。此时连续退出位于栈顶的操作符。

# 3 实现

## 3.1 判断是否是运算符

### 3.1.1 代码实现

//判断是否是运算符

bool isdigit(char ch)

{

if(ch=='+' ||ch=='-' ||ch=='\*'||ch=='/'){return false;}

if(ch=='(' ||ch==')'){return false;}

return true;

}

## 3.2 计算栈内优先数

### 3.2.1 代码实现

//栈内优先数

int isp(char ch)

{

if(ch=='#'){return 0;}

if(ch=='('){return 1;}

if(ch=='+' || ch=='-' ){return 3;}

if(ch=='\*' || ch=='/' || ch== '%'){return 5;}

if(ch==')'){return 6;}

}

## 3.3 计算栈外优先数

### 3.3.1 代码实现

//栈外优先数

int icp(char ch)

{

if(ch=='#'){return 0;}

if(ch=='('){return 6;}

if(ch=='+' || ch=='-' ){return 2;}

if(ch=='\*' || ch=='/' || ch== '%'){return 4;}

if(ch==')'){return 1;}

}

## 3.4 核心算法

### 3.4.1 核心算法

//将栈底标志#先放入栈中

SeqStack<char> s;

char ch='#',ch1,op;

s.Push(ch);

cin>>ch;

if(ch=='-')

{

cout<<'-';

cin>>ch;

}

char a;

while(s.IsEmpty()==false)

{

//如果输入结束，则将栈内的元素逐个退出

if(ch=='\n')

{

cout<<' ';

while(s.getTop(a) &&a!='#')

{

s.Pop(a);

cout<<a<<' ';

}

}

//如果是操作数，输出它

if(isdigit(ch))

{

cout<<ch;

cin.get(ch);

}

else

{

s.getTop(ch1);

//新输入操作符ch的优先级较高，进栈，读入下一个字符

if(isp(ch1)<icp(ch))

{

s.Push(ch);

cin>>ch;

}

//新输入操作符ch的优先级较低，出栈

else if(isp(ch1)>icp(ch))

{

s.Pop(op);

cout<<op<<' ';

}

//新输入操作符ch的优先级等于栈顶优先级

else

{

s.Pop(op);

if(op=='('){cin>>ch;}

}

}

}

# 4 测试

## 4.1 测试用例测试

|  |  |  |  |
| --- | --- | --- | --- |
| 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 \* + | 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 |

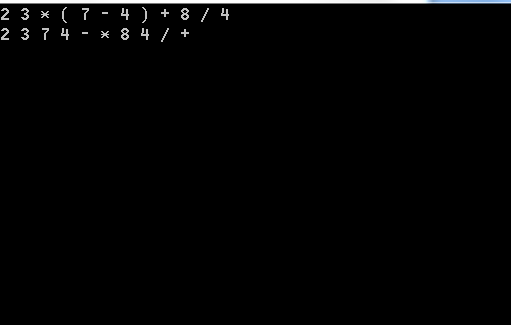
### 4.1.1正常测试6种运算符

**测试用例**：2 + 3 \* ( 7 – 4 ) + 8 / 4

**预期结果**：

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

**实验结果**



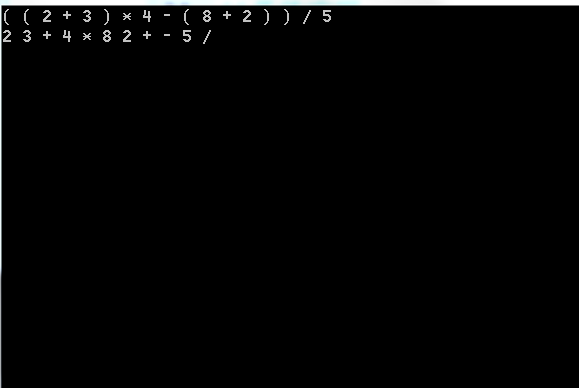
### 4.1.2嵌套括号

**测试用例：**1314 + 25.5 \* 12

**预期结果：**

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

**实验结果：**



### 4.1.3运算数超过1位整数且有非整数出现

**测试用例：**1314 + 25.5 \* 12

**预期结果：**

1314 25.5 12 \* +

**实验结果：**

****

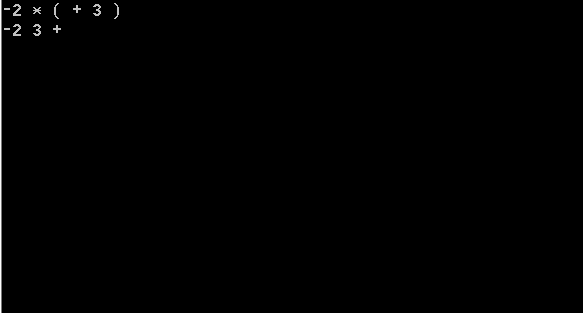
### 4.1.4运算数有正或负号

**测试用例：**-2 \* ( +3 )

**预期结果：**

-2 3 +

**实验结果：**



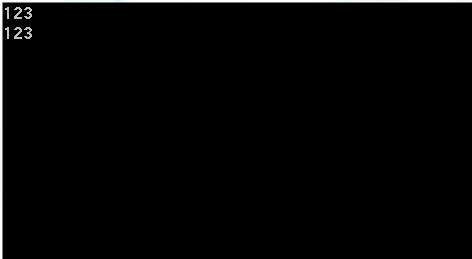
### 4.1.5只有1个数字

**测试用例：**123

**预期结果：**

123

**实验结果：**



## 4.2 其他测试

### 4.2.1 只有运算符

**测试用例：**-

**预期结果：**-

**实验结果：**

