项目说明文档

数据结构课程设计

——表达式转换

作 者 姓 名： 张诚睿

学 号： 2150998

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](#_Toc122644429)

[1.1 项目背景分析 1](#_Toc122644430)

[1.2 功能分析 1](#_Toc122644431)

[2 设计 1](#_Toc122644432)

[2.1 数据结构设计 1](#_Toc122644433)

[2.2 类结构设计 2](#_Toc122644434)

[2.3 类具体成员与操作设计 2](#_Toc122644435)

[2.4 系统设计 3](#_Toc122644436)

[3 实现 3](#_Toc122644437)

[3.1 链式栈-进栈功能的实现 3](#_Toc122644438)

[3.1.1 功能流程图 3](#_Toc122644439)

[3.1.2 核心代码 4](#_Toc122644440)

[3.2 链式栈-出栈功能实现 5](#_Toc122644441)

[3.2.1 功能流程图 5](#_Toc122644442)

[3.2.2 核心代码 5](#_Toc122644443)

[3.3 转换后缀表达式的实现 6](#_Toc122644444)

[3.3.1 功能流程图 6](#_Toc122644445)

[3.3.2 核心代码 6](#_Toc122644446)

[3.5 总体功能的实现 8](#_Toc122644447)

[3.5.1 总体流程 8](#_Toc122644448)

[3.5.2 总体功能核心代码 8](#_Toc122644449)

[3.6.3 总体功能截屏示例 8](#_Toc122644450)

[4 测试 8](#_Toc122644451)

[4.1 功能测试 8](#_Toc122644452)

[4.1.1 转换一般中缀表达式功能测试 8](#_Toc122644453)

[4.2 边界测试 9](#_Toc122644454)

[4.2.1嵌套括号 9](#_Toc122644455)

[4.2.2 运算数超过1位整数且有非整数出现 9](#_Toc122644456)

# 1 分析

## 1.1 项目背景分析

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，但在编译程序中通常使用后缀表达式求解表达式的值。因此，将中缀表达式转换为后缀表达式的操作很重要，需要设计程序来完成。

## 1.2 功能分析

本项的目的在于实现对给定中缀表达式（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）求解它的后缀表达式，并在一行中输出转换后的后缀表达式，不同对象（运算数，运算符号）之间以空格分隔，且结尾没有多余空格。

# 2 设计

## 2.1 数据结构设计



核心数据结构设计如上，通过类-LinkedStackForTrans实现中缀表达式转换后缀表达式功能，其内利用struct Node构成的链表模拟栈的操作，结点中用char[]来存放运算符或操作数，实际过程中，由于表达式不超过20个字符，char content[]实际大小为21\*sizeof(char)，足够处理数字很大的极端情况。

## 2.2 类结构设计

如上所述，本项目采用struct Node描述链表结点类（LNode），这样使得链表结点类LinkedStackForTrans（LinkList）可以访问链表结点。

所有计算表达式的操作均在LinkedStackForTrans类完成，它的私有成员还包括记录当前处理原字符串位置的变量、存放中缀表达式与后缀表达式的动态内存申请指针、记录字符串长度的变量以及部分函数。

## 2.3 类具体成员与操作设计

**链表结点类**

struct Node {

char content[21];

Node\* next;

};

**链式栈类**

class LinkedStackForTrans {

private:

Node\* top; // 头指针

char\* Origin; // 原中缀表达式

char\* Output; // 结果后缀表达式

int ptrOrigin; // 记录当前处理Origin的位置

int OriginLength, OutputLength; // 字符串长度

void Push(char\* in); // 进栈

bool Pop(char\* out); // 出栈

void makeEmpty(); // 释放内存

char\* GetTop();

bool GetOriginElement(char\* ch);

int Isp(const char ch);

int Icp(const char ch);

void WriteIn(char\* str);

public:

LinkedStackForTrans() :top(NULL), Origin(NULL),Output(NULL), ptrOrigin(0), OriginLength(0), OutputLength(0)

{

Output = new char[42];

if (Output == NULL)

{

cout << "Memory Wrong" << endl;

exit(EXIT\_FAILURE);

}

memset(Output, 0, 42); // 全部置\0

};

~LinkedStackForTrans() { makeEmpty(); };

void Transform(); // 中缀换后缀

friend istream& operator>>(istream& is, LinkedStackForTrans& target);

friend ostream& operator<<(ostream& os, LinkedStackForTrans& target);

};

## 2.4 系统设计

首先建立LinkedStackForTrans对象Expression，输入中缀表达式后调用Expression.Transform()函数计算结果，最后输出后缀表达式，主函数返回0。

# 3 实现

此处仅介绍核心功能的实现。

## 3.1 链式栈-进栈功能的实现

### 3.1.1 功能流程图

### 3.1.2 核心代码

if (top == NULL) // 空栈

{

top = new Node;

if (top == NULL)

{

cout << "Memory Wrong";

exit(EXIT\_FAILURE);

}

strcpy\_s(top->content, in);

top->next = NULL;

}

else // 非空栈

{

Node\* tmp = new Node;

if (tmp == NULL)

{

cout << "Memory Wrong";

exit(EXIT\_FAILURE);

}

strcpy\_s(tmp->content, in);

tmp->next = top;

top = tmp;

}

## 3.2 链式栈-出栈功能实现

### 3.2.1 功能流程图

### 3.2.2 核心代码

if (top == NULL)

{

return false;

}

else

{

out[0] = top->content[0];

Node\* tmp = top;

top = top->next;

delete tmp;

return true;

}

return false;

## 3.3 转换后缀表达式的实现

### 3.3.1 功能流程图

### 3.3.2 核心代码

Push(ch); // 结束符入栈

if (!GetOriginElement(ch)) // 取出第一个元素

{

return;

}

do

{

if ((ch[0] >= '1' && ch[0] <= '9') || ((ch[0] == '-' || ch[0] == '+') && ch[1] >= '1' && ch[1] <= '9')) // 是操作数

{

WriteIn(ch);

if (!GetOriginElement(ch))

{

break;

}

}

else // 是操作符

{

op[0] = top->content[0];

if (Icp(ch[0]) > Isp(op[0]))

{

Push(ch); // 应当进栈

if (!GetOriginElement(ch))

{

break;

}

}

else if (Icp(ch[0]) < Isp(op[0]))

{

Pop(out); // 应当出栈并输出

WriteIn(out);

}

else

{

Pop(out); // 退栈但不输出

if (!GetOriginElement(ch))

{

break;

}

}

}

} while (!(ch[0] == '#' && !strcmp(GetTop(), "#")));

## 3.5 总体功能的实现

### 3.5.1 总体流程

首先建立LinkedStackForTrans对象Expression，输入中缀表达式后调用Expression.Transform()函数计算结果，最后输出后缀表达式，主函数返回0。

### 3.5.2 总体功能核心代码

int main()

{

LinkedStackForTrans Expression;

cin >> Expression;

Expression.Transform(); // 生成结果

cout << Expression;

return 0;

}

### 3.6.3 总体功能截屏示例



# 4 测试

## 4.1 功能测试

### 4.1.1 转换一般中缀表达式功能测试

**测试用例**：

2 + 3 \* ( 7 – 4 ) + 8 / 4

**预期结果**：

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

**实验结果**



输出结果末尾没有多余空格

## 4.2 边界测试

### 4.2.1嵌套括号

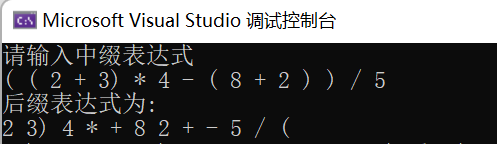
**测试用例：**

( ( 2 + 3) \* 4 – ( 8 + 2 ) ) / 5

**预期结果：**

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

**实验结果：**



输出结果末尾没有多余空格

### 4.2.2 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：**



**4.2.3 运算数有正或负号**

**测试用例：**

-2 \* ( +3 )

**预期结果：**

-2 3 \*

**实验结果：**



**4.2.4 只有1个数字**

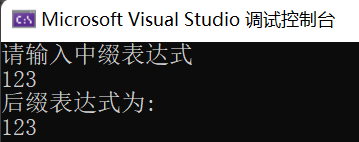
**测试用例：**

123

**预期结果：**

123

**实验结果：**



**4.2.5 中缀表达式为空**

**测试用例（输入为空，仅有一个回车）：**

\n

**预期结果（输出为空）：**

**实验结果：**

