项目说明文档

数据结构课程设计

——表达式转换

作 者 姓 名： 张喆

学 号： 1754060

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](#_Toc533503721)

[1.1 背景分析 1](#_Toc533503722)

[1.2 功能分析 1](#_Toc533503723)

[2 设计 2](#_Toc533503724)

[2.1 数据结构设计 2](#_Toc533503725)

[2.2 类结构设计 2](#_Toc533503726)

[2.3 成员与操作设计 3](#_Toc533503727)

[2.4 系统设计 3](#_Toc533503728)

[3 实现 4](#_Toc533503729)

[3.1 表达式转换功能的实现 4](#_Toc533503730)

[3.1.1 表达式转换功能流程图 4](#_Toc533503731)

[3.1.2 表达式转换功能核心代码 4](#_Toc533503732)

[3.1.3 表达式转换功能截屏示例 5](#_Toc533503733)

[3.2 输出后缀表达式功能的实现 6](#_Toc533503734)

[3.2.1输出后缀表达式功能流程图 6](#_Toc533503735)

[3.2.2 输出后缀表达式功能核心代码 6](#_Toc533503736)

[3.2.3 输出后缀表达式功能截屏示例 6](#_Toc533503737)

[3.3 取下一元素功能的实现 7](#_Toc533503738)

[3.3.1 取下一元素功能流程图 7](#_Toc533503739)

[3.3.2 取下一元素功能核心代码 7](#_Toc533503740)

[3.3.3 取下一元素功能截图示例 8](#_Toc533503741)

[3.4 判断某一元素是否为数字功能的实现 10](#_Toc533503742)

[3.4.1 判断某一元素是否为数字功能流程图 10](#_Toc533503743)

[3.4.2 判断某一元素是否为数字功能核心代码 10](#_Toc533503744)

[3.4.3 判断某一元素是否为数字功能截屏示例 11](#_Toc533503745)

[3.5 总体系统的实现 12](#_Toc533503746)

[3.5.1 总体系统核心代码 12](#_Toc533503747)

[3.5.2 总体系统截屏示例 12](#_Toc533503748)

[4 测试 13](#_Toc533503749)

[4.1 功能测试 13](#_Toc533503750)

[4.1.1 表达式转换功能测试 13](#_Toc533503751)

[4.1.2 输出后缀表达式功能测试 13](#_Toc533503752)

[4.1.3 取下一元素功能测试 13](#_Toc533503753)

[4.1.4 判断某一元素是否为数字功能测试 14](#_Toc533503754)

[4.2 边界测试 15](#_Toc533503755)

[4.2.1 嵌套括号 15](#_Toc533503756)

[4.2.2 运算数超过1位整数且有非整数出现 15](#_Toc533503757)

[4.2.3 运算数有正或负号 15](#_Toc533503758)

[4.2.4 只有一个数字 15](#_Toc533503759)

# 1 分析

## 1.1 背景分析

如何将表达式翻译成能够正确求值的指令序列，是语言处理程序要解决的基本问题。任何一个表达式都是由操作数、操作符和分界符组成的。

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达式，即二元运算符位于两个运算数中间。为了正确执行这种中缀表达式的计算，必须明确各个操作符的执行顺序。为此每个操作符都规定了一个优先级。

C++规定，一个表达式中相邻的两个操作符的计算次序为：优先级高的先计算；如果优先级相同，则自左向右计算；当使用括号时，从最内层的括号开始计算。

由于中缀表达式中有操作符的优先级问题，还有可加括号改变运算顺序的问题，所以对于编译程序来说，一般不适用中缀表示处理表达式。解决办法是用后缀表示。

## 1.2 功能分析

作为能实现将中缀表达式转换为后缀表达式的程序，首先应能正确读入用户想要转换的中缀表达式（该中缀表达式是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，表撒施不超过20个字符）。

之后可以根据相应的规则和各运算符的优先级正确无误的将中缀表达式转化为后缀表达式。

最后可以向用户输出转化好的后缀表达式（同时要求，在一行中输出转化后的后缀表达式，要求不同对象之间以空格分隔，但是结尾不得有多余空格）

# 2 设计

## 2.1 数据结构设计

使用一个操作符栈用于表达式转换。

同时设定两个优先级，isp是栈内优先数，icp是栈外优先数，取值如下

int isp(char ch)

//栈内优先级

{

switch (ch)

{

case '#':return 0;

case '(':return 1;

case '\*':case '/':return 5;

case '+':case '-':return 3;

case ')':return 6;

}

}

int icp(char ch)

//栈外优先级

{

switch (ch)

{

case '#':return 0;

case '(':return 6;

case '\*':case '/':return 4;

case '+':case '-':return 2;

case ')':return 1;

}

}

左括号的栈外优先数最高，它一来到立即进栈，但当它进入栈中后，其栈内优先数变得极低，以便括号内的其他操作符进栈。其他操作符进入栈中后优先数都升1，这样可体现在中缀表达式中相同优先级的操作符从左向右计算的要求，让位于栈顶的操作符先退栈听输出。操作符优先数相等的情况只出现在括号配对或栈顶的‘#’号与输入流最后的‘#’号配对时。前者将见徐退出位于栈顶的操作符，直到遇到左括号为止。然后将左括号退栈以对消括号，后者将结束算法。

## 2.2 类结构设计

设计表达式类，用于储存用户输入的中缀表达式和待输出的处理后的后缀表达式；同时维护操作符栈用于转换表达式。

## 2.3 成员与操作设计

**表达式类（Express）**

**私有成员：**

string \_express; //储存中序表达式

int \_current = 0; //表达式的定位变量

string \_next; //下一个内容

string \_result; //储存后缀表达式

stack<char> OpCode; //操作符栈

**表达式类的构造函数：**

Express() = default;

Express(string &buf);

Express::Express(string & buf)

//对表达式进行一些处理,在结尾添加下面内容用于判断终止

:\_express(buf)

{

\_express.push\_back(' ');

\_express.push\_back('#');

\_express.push\_back(' ');

}

**公有操作：**

void NextContent(); //寻找下一个对象

void Change(); //将中序表达式转化为后续表达式

void ShowResult(); //输出结果(结尾不能有空格)

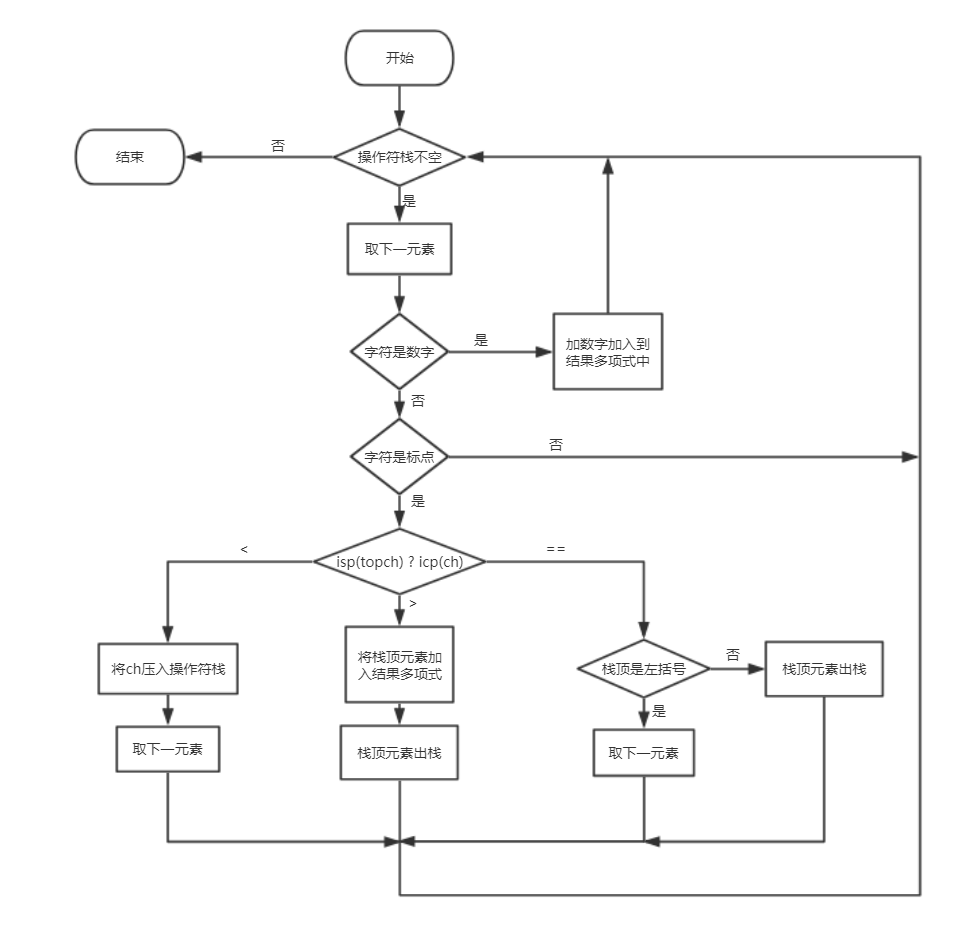
## 2.4 系统设计

系统首先读入用户输入的中缀表达式，并构建以该表达式为基础的表达式类，之后调用相应的算法将中缀表达式转换为后缀表达式。最后按照正确的格式输出后缀表达式。

# 3 实现

## 3.1 表达式转换功能的实现

### 3.1.1 表达式转换功能流程图



### 3.1.2 表达式转换功能核心代码

void Express::Change()

{

OpCode.push('#'); //操作符栈初始化，将结束符#进栈

NextContent(); //读入中缀表达式字符流的首字符

while (!OpCode.empty())

//循环做, 直到ch=='#', 同时栈顶的操作符也是'#', 停止循环

{

char ch = \_next[0];

if (MyisNum(\_next)) //如果是操作数, 直接读入下一个字符

{

\_result.append(\_next); //添加到结果当中

\_result.push\_back(' ');

NextContent();

}

else if (ispunct(ch)) //如果是操作符, 判断ch的优先级icp和当前栈顶操作符的优先级isp

{

char topch = OpCode.top();

if (isp(topch) < icp(ch)) //当前操作符优先级大,另ch进栈,读入下一个字符

{

OpCode.push(ch);

NextContent();

}

else if (isp(topch) > icp(ch)) //当前优先级小,退栈并输出到结果中

{

\_result.push\_back(OpCode.top());

\_result.push\_back(' ');

OpCode.pop();

}

else

{

if (OpCode.top() == '(') //如果退出的是左括号则读入下一个字符

{

NextContent();

}

OpCode.pop();

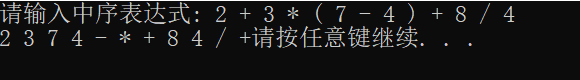
}

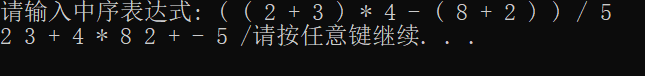
}

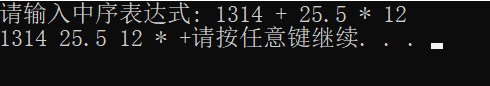
}

}

### 3.1.3 表达式转换功能截屏示例

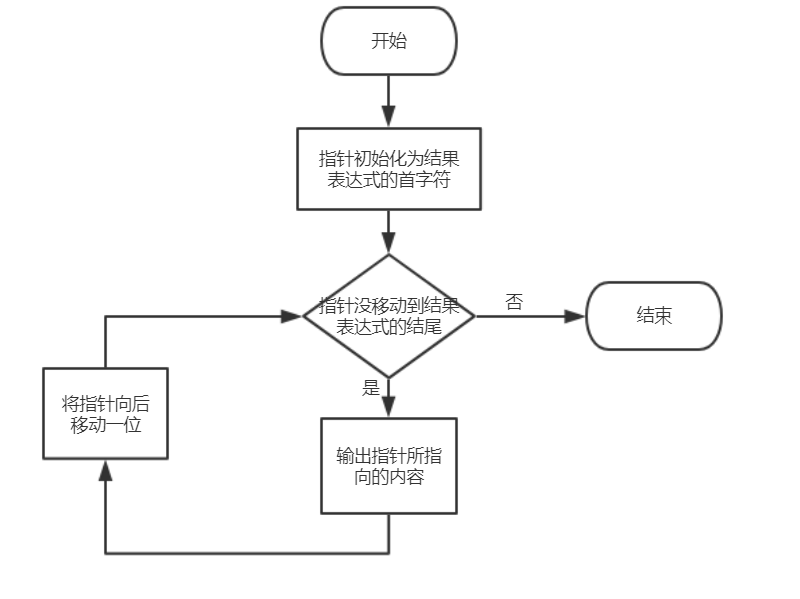






## 3.2 输出后缀表达式功能的实现

### 3.2.1输出后缀表达式功能流程图



### 3.2.2 输出后缀表达式功能核心代码

void Express::ShowResult()

//结尾空格不输出

{

for (char \*pch = &\_result[0]; pch < &\_result[0] + \_result.size() - 1; ++pch)

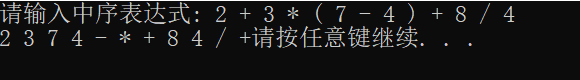
{

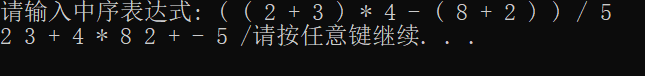
cout << \*pch;

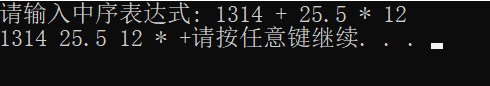
}

}

### 3.2.3 输出后缀表达式功能截屏示例

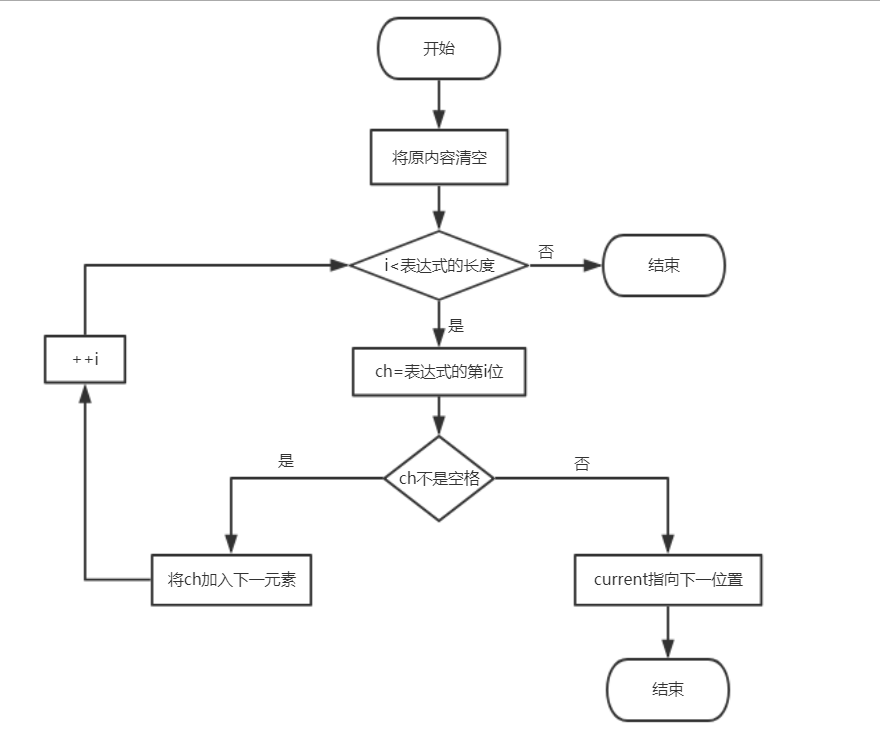






## 3.3 取下一元素功能的实现

### 3.3.1 取下一元素功能流程图



### 3.3.2 取下一元素功能核心代码

void Express::NextContent()

{

\_next.clear(); //每次找下一个内容先把\_next清空

char ch;

for (int i = \_current; i < this->\_express.size(); ++i)

{

ch = \_express[i];

if (!isspace(ch))

{

\_next.push\_back(ch);

//因为不同对象以空格隔开,所以只要不是空格就加到\_next

}

else

{

\_current = i + 1; //\_current指向下一个位置,结束当前对象的寻找

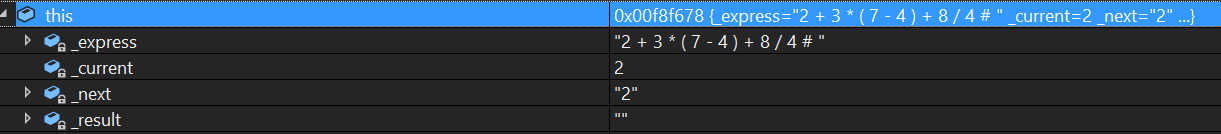
break;

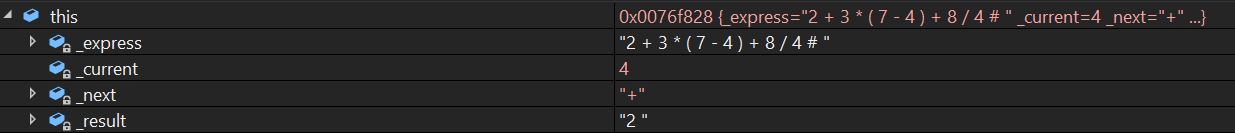
}

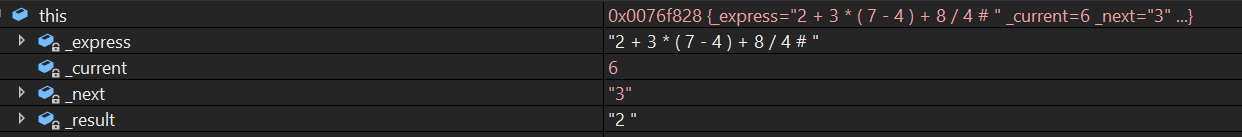
}

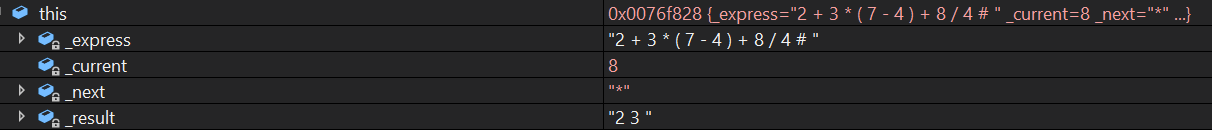
}

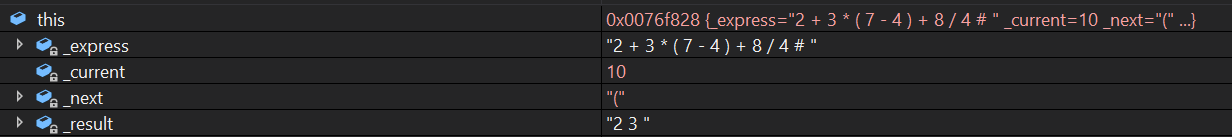
### 3.3.3 取下一元素功能截图示例

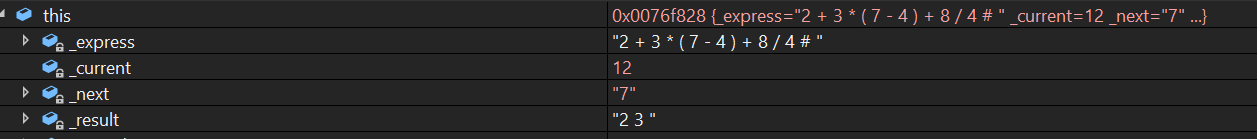


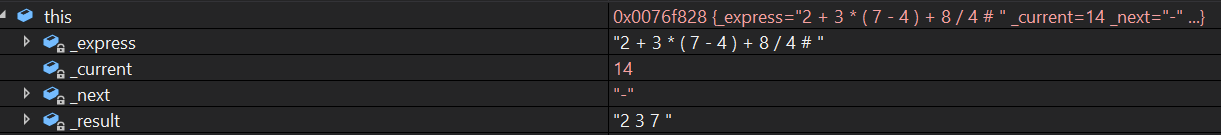


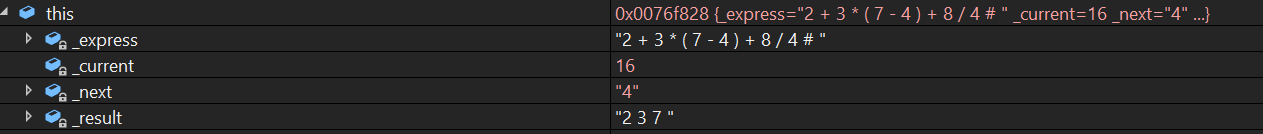


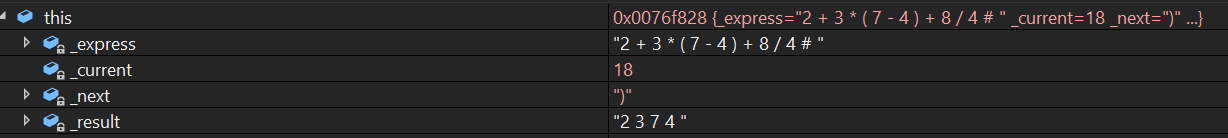


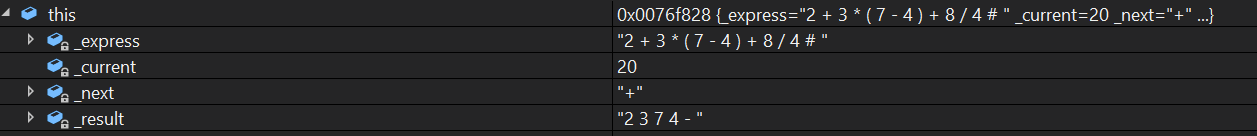


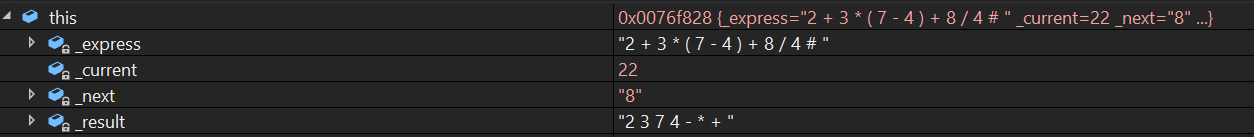


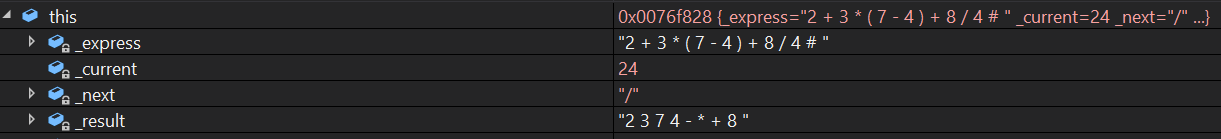


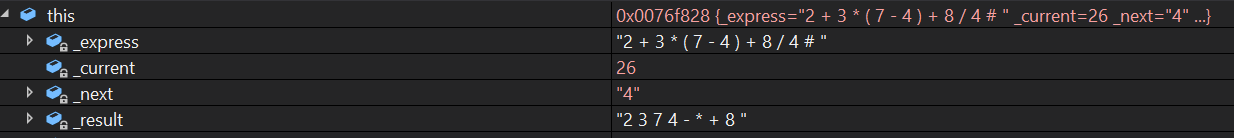


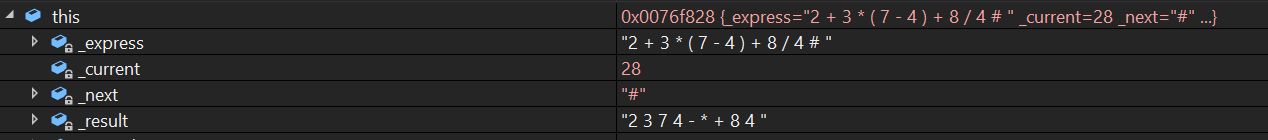






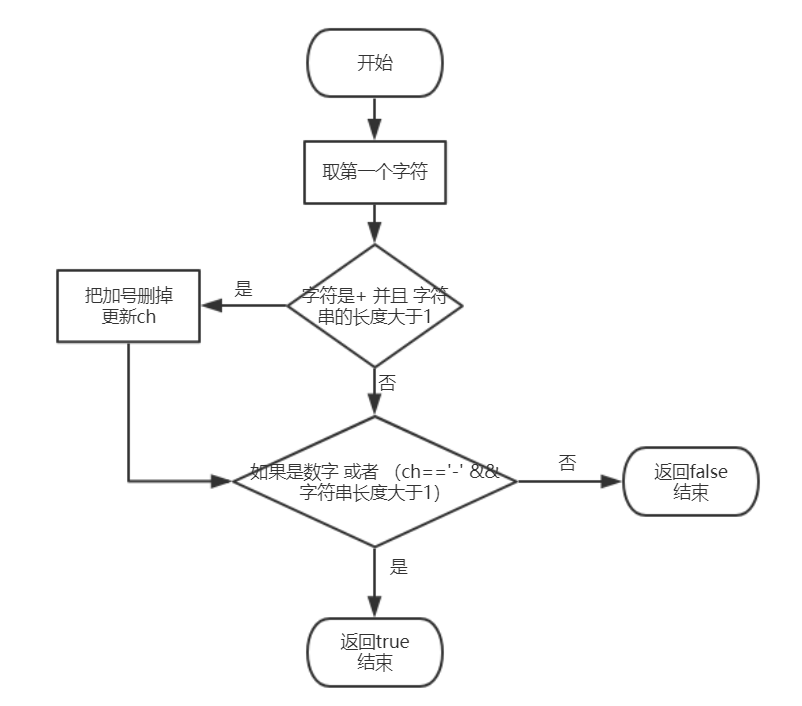






## 3.4 判断某一元素是否为数字功能的实现

### 3.4.1 判断某一元素是否为数字功能流程图



### 3.4.2 判断某一元素是否为数字功能核心代码

bool MyisNum(string &buf)

//用于判断当前对象是否为一个数字

//数字包括 正数 负数 浮点数 (其中+3要进行特殊处理)

{

char ch = buf[0];

if (ch == '+' && buf.size() > 1)

// 如 +3 就是储存 3

{

buf.erase(0, 1); //把+删除

ch = buf[0]; //更新一下ch

}

if (isdigit(ch) || (ch == '-' && buf.size()>1))

//储存各种数字, 包括正数,负数,浮点数

{

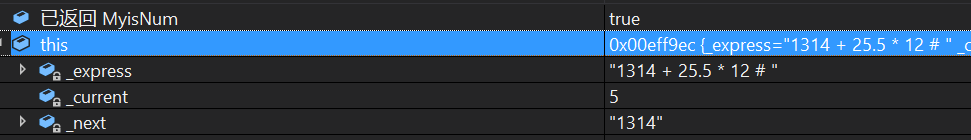
return true;

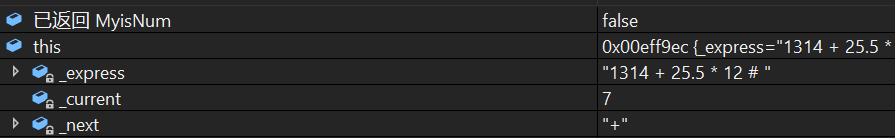
}

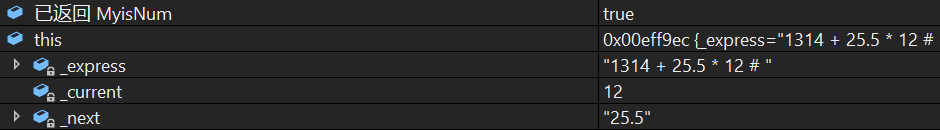
return false;

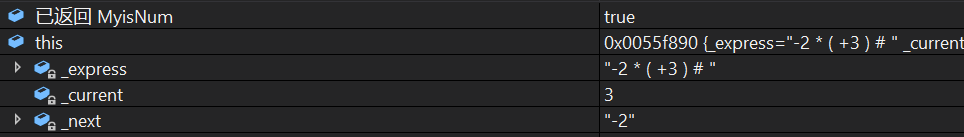
}

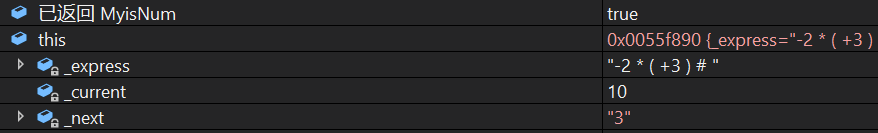
### 3.4.3 判断某一元素是否为数字功能截屏示例











## 3.5 总体系统的实现

### 3.5.1 总体系统核心代码

string buf;

cout << "请输入中序表达式: ";

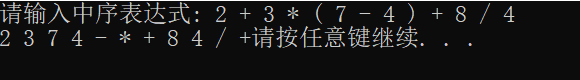
getline(cin, buf);

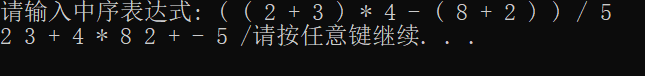
Express express(buf);

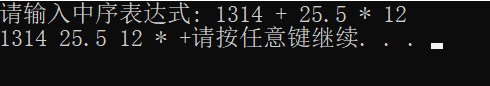
express.Change();

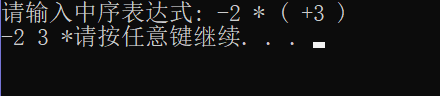
express.ShowResult();

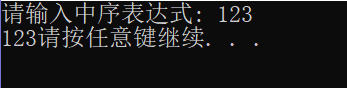
### 3.5.2 总体系统截屏示例











# 4 测试

## 4.1 功能测试

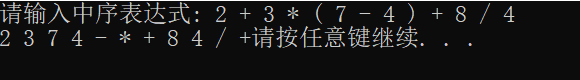
### 4.1.1 表达式转换功能测试

**测试用例**：2 + 3 \* ( 7 - 4 ) + 8 / 4

**预期结果**：

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

**实验结果：**

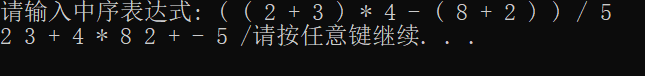


**测试用例**：( ( 2 + 3 ) \* 4 - ( 8 + 2 ) ) / 5

**预期结果**：

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

**实验结果：**



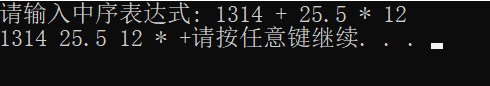
### 4.1.2 输出后缀表达式功能测试

**测试用例：**1314 + 25.5 \* 12

**预期结果：**

1314 25.5 12 \* +

**实验结果：**



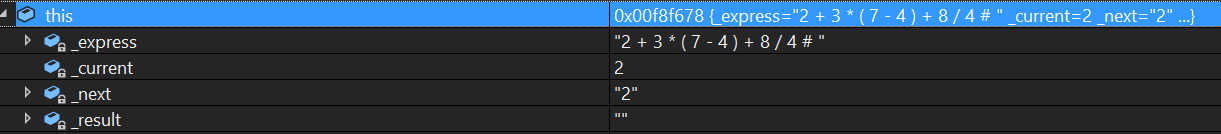
### 4.1.3 取下一元素功能测试

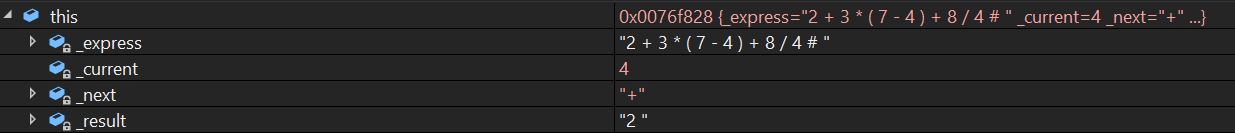
**测试用例：**2 + 3 \* ( 7 - 4 ) + 8 / 4

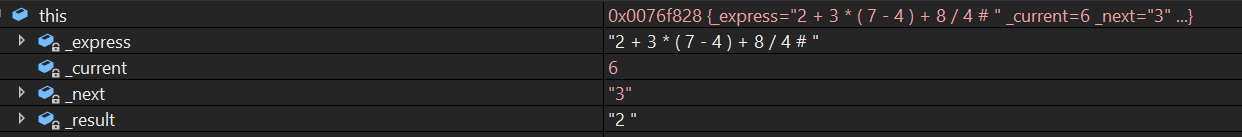
**预期结果：**

可以依次正确取出下一个元素，并且适用于浮点数、负数及各种运算符

**实验结果：**







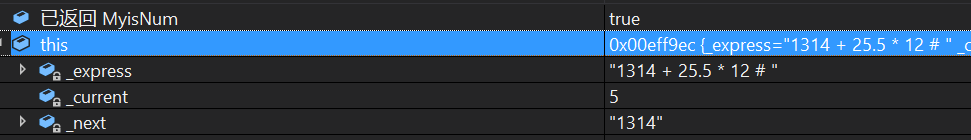
### 4.1.4 判断某一元素是否为数字功能测试

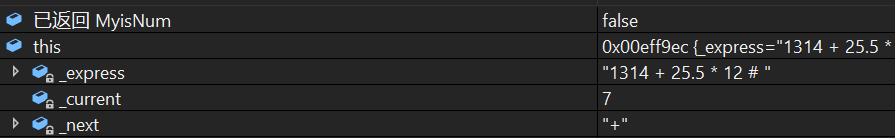
**测试用例：**1314 + 25.5 \* 12

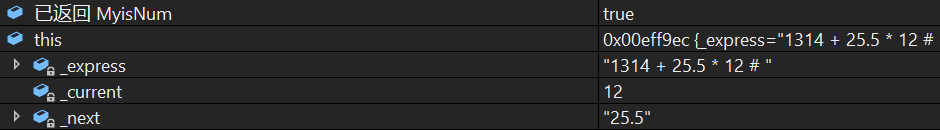
**预期结果：**

可以对浮点数、负数以及特殊的数字（如+3）做出正确的判断

**实验结果：**







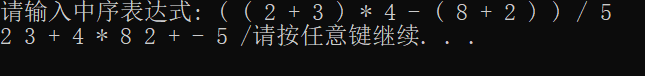
## 4.2 边界测试

### 4.2.1 嵌套括号

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

**预期结果：**正确处理好双重括号中各运算符的优先级

**实验结果：**

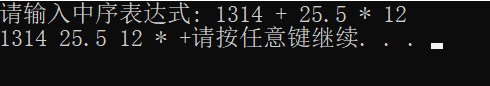


### 4.2.2 运算数超过1位整数且有非整数出现

**测试用例：**1314 + 25.5 \* 12

**预期结果：**正确处理多位整数以及浮点数的情况，并正确转换为后缀表达式

**实验结果：**

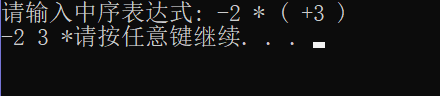


### 4.2.3 运算数有正或负号

**测试用例：**-2 \* ( +3 )

**预期结果：**正确处理负数以及带正号的正数的情况，并正确转换为后缀表达式

**实验结果：**



### 4.2.4 只有一个数字

**测试用例：**123

**预期结果：**只有一个数字的中缀表达式和后缀表达式相同

**实验结果：**

