《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 18计算机类 | 姓名 |  | 学号 |  |
| 实验布置日期 | | 2019-9-20 | | 提交  日期 | 2019-9-27 | | 成绩 |  |

课程实践实验2：中缀表达式求值

## 一、问题描述及要求

中缀表达式是我们熟悉的表达式形式。为了能正确表示运算的先后顺序，中缀表达式中难免要出现括号。假设我们的表达式中只允许有圆括号。

读入一个浮点数为操作数的中缀表达式后，对该表达式进行运算。

要求中缀表达式以一个字符串的形式读入，可含有加、减、乘、除运算符和左、右括号，并假设该表达式以“#”作为输入结束符。

如输入“3.5\*(20+4)-1#”，则程序运行结果应为83。

要求可单步显示输入序列和栈的变化过程。并考虑算法的健壮性，当表达式错误时，要给出错误原因的提示。

## 二、概要设计

**可包含以下内容，可根据实际情况取舍。**

1. **对实验内容的理解和二次概括。**

**通过输入中缀表达式，对其字符串进行操作，转化为后缀表达式，并将其压入栈中，便于编程计算，最终可以得出结果。**

1. **给出系统的功能列表（可以用图示或清单的形式）**

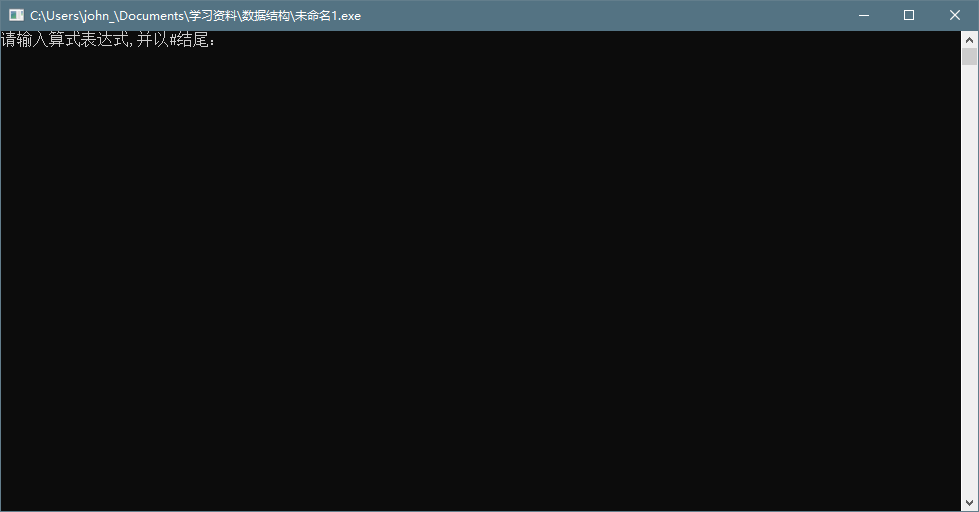
**1.用户可按照正确方法输入中缀表达式。**

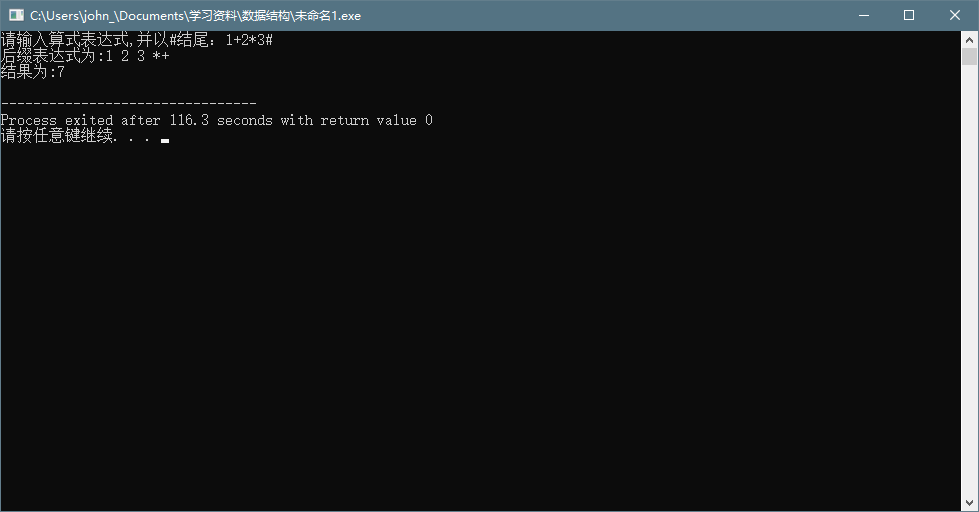
**2.系统能够识别中缀表达式。**

**3.系统能够计算正确的中缀表达式并输出结果。**

**4.如表达式含有错误，可识别并提示用户错误。**

**（3）程序运行的界面设计 （可以用图示等方式，如：首先出现屏幕提示，请用户选择输入配置的方式，1从键盘输入活单元坐标2……3….然后用户）**





**（4）确定总体设计思路，采用何种数据结构，设计哪些类，各类的作用 ，类方法的介绍，类之间的关系描述**

使用堆栈结构来处理表达式，并且设计三个函数分别负责转换表达式，通过后缀表达式计算值和检查算式是否错误。

**（5）程序结构设计，包括：对已有程序的使用，自己将设计哪些程序文件，各部分关系描述。**

这部分最核心的工作是：描述解决问题所需用到的数据结构和算法。——给出明确地确定该结构及对应算法的理由。

此外，这部分可以给出面向对象相关的设计，例如实现完整功能所涉及的类及对象间的总体关联关系等。

解决此问题，我采用了定义类的方式，将生命游戏中的多种操作定义为类中的函数，这样可以确保每个操作前后联系紧密，也使代码更具系统性，便于管理

## 三、详细设计

**主函数、关键算法的设计（可以用流程图等手段表示）**

开始

输入表达式

表达式是否合法？

否

是

输出错误

读入字符，

判断字符类型

输出最后结果

结束

数字直接输出到新的表达式字符串中

高优先级运算符（‘\*’，‘/’，‘（’）直接压入栈中

低优先级运算符（‘+‘，’-‘）

‘）’则依次弹栈输出直至遇到（为止

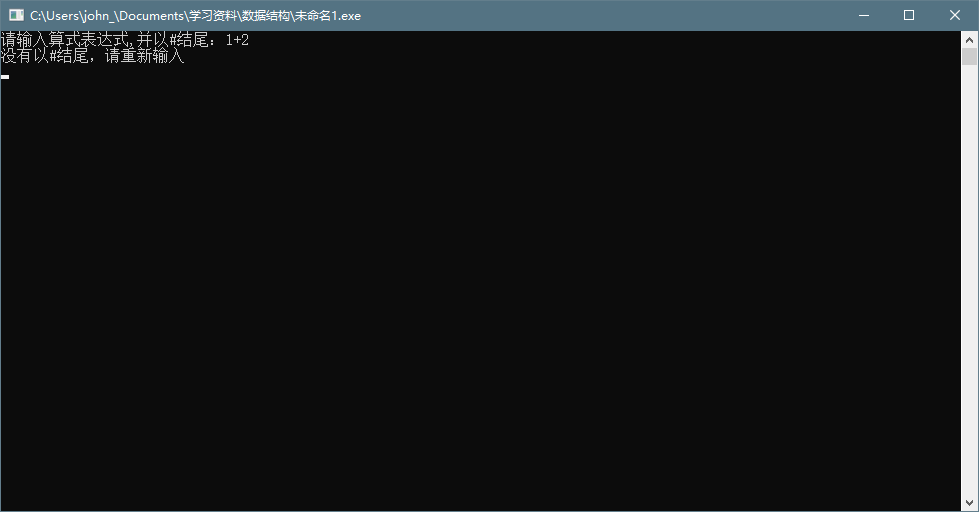
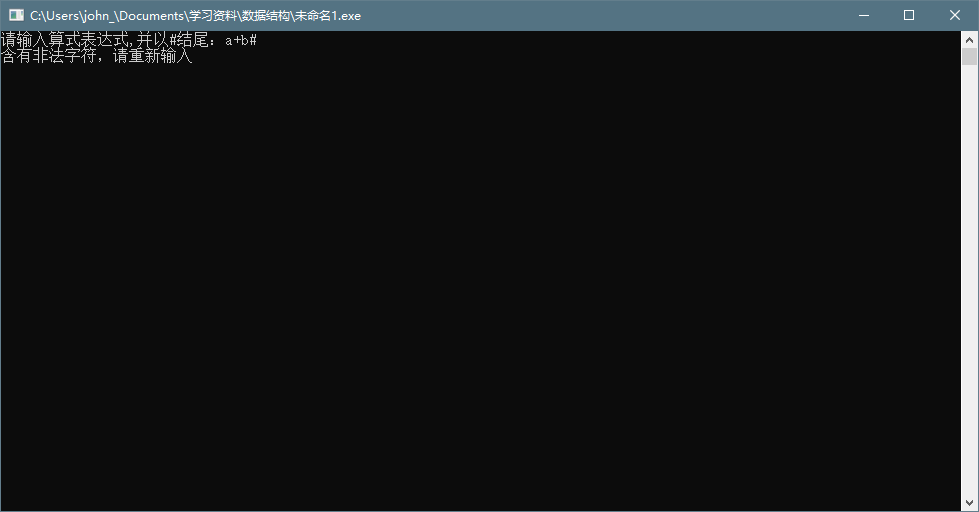
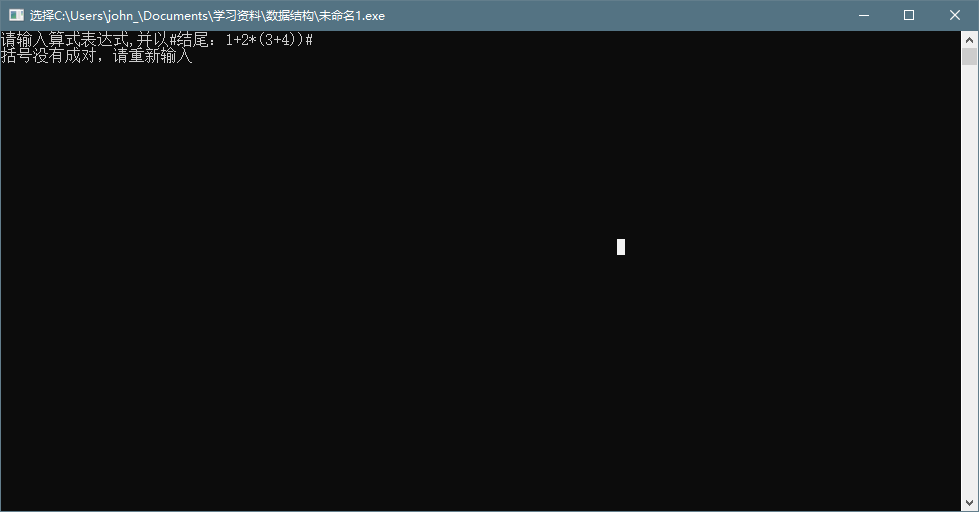
检查栈顶运算符，如果优先级较高则先弹栈输出在将低优先级运算符压入栈

用函数计算表达式的值

**着重对难点部分进行分析。**

## 四、实验结果

紧扣题目要求设计提供相应的测试方法和结果。可以给出具体的测试用例，每个测试用例一般可列出：

* 测试输入：1+2
* 测试目的：检查是否能检测出缺少#
* 正确输出：没有以#结尾，请重新输入
* 实际输出：
* 
* 测试结论：通过
* 测试输入：a+b#
* 测试目的：测试是否能检测出算式非法
* 正确输出：含有非法字符，请重新输入
* 实际输出：
* 
* 测试结论：通过
* 测试输入：1+2\*(3+4))#
* 测试目的：检查是否能检测出括号不匹配
* 正确输出：括号没有成对，请重新输入
* 实际输出：
* 
* 测试结论：通过

## 五、实验分析与探讨

这部分是实验报告最出彩的地方，主要应包含两方面：

测试结果的分析：解释测试策略，对得到的结果进行分析，总结算法的时空复杂度，得出对算法性能方面分析的结论

不局限于题目的一些限制条件和使用算法的要求，探讨更多解决问题的途径等。

**实验设计、实现过程中遇到的问题，将在分析问题、解决问题的整个过程中遇到的问题罗列出来，并写出解决问题的方法。**

**时间复杂度：**

**结果分析：在将中缀表达式转化为后缀表达式时操作过于繁琐，需要检测栈顶的符号优先级来做出判断，因此较为繁琐。在检测一个完整的数字时有多个循环嵌套，因此效率较为低下。**

## 六、小结

本程序主要运用了堆栈的特性来处理表达式，主要难点在于中缀表达式转后缀表达式中的数学优先级问题，具体要求过于繁琐，导致无法较为简洁的编写。

## 附录：源代码

#include <iostream>

#include <stack>

#include <string>

#include <sstream>

using namespace std;

string midfixtopostfix(string &s)

{

int i = 0;

stack<char> symbols;

string output = "";

while (s[i] != '#') //从整个表达式开头一个字一个字开始循环，直到#

{

if (isdigit(s[i]) || s[i] == '.') //如果是数字

{

output += s[i]; //把这个数字加进output的字符串里

if (!(isdigit(s[i + 1]) || s[i + 1] == '.')) output += ' '; //如果下一位不是数字或者小数点 就在输出字符串里加上一个空格 表示这个数字已经结尾了

}

else if (s[i] == '\*' || s[i] == '/') //如果是\*或/

{

while (!symbols.empty())

{

if (symbols.top() == '\*' || symbols.top() == '/')//检查栈顶是否也是\*或/

{

output += symbols.top();//如果是，把栈顶元素弹出，并加入输出字符串，再把新来的\*或/压入栈中

output += ' ';

symbols.pop();

}

else break;

}

symbols.push(s[i]);//如果不是，就直接压入栈

}

else if (s[i] == '(') symbols.push(s[i]);//如果是（，直接压入栈

else if (s[i] == '+' || s[i] == '-')//如果是+或-

{

while (!symbols.empty())

{

if (symbols.top() == '\*' || symbols.top() == '/' || symbols.top() == '+' || symbols.top() == '-')//检查前一位是否是\*或/或+或-，同理

{

output += symbols.top();

output += ' ';

symbols.pop();

}

else break;

}

symbols.push(s[i]);

}

else if (s[i] == ')')//如果是）

{

while (!symbols.empty())

{

if (symbols.top() != '(')//则弹栈直到碰到（

{

output += symbols.top();

output += ' ';

symbols.pop();

}

else break;

}

symbols.pop();

}

i++;

}

while (!symbols.empty())//最后检查栈是否空

{

output += symbols.top();//全部弹入字符串里

output += ' ';

symbols.pop();

}

return output;

}

double calculator(const string &s)

{

stack<double> calc;//定义一个栈

double x, y;

int n = s.size(), i = 0;//n为表达式长度

double tempdigit;

string tempstr;

while (i<n)//从表达式开头循环到结尾

{

tempstr = "";

while (isdigit(s[i]) || s[i] == '.') //判断当前这一位是不是数字或者.

{

tempstr += s[i];//如果是就暂时存储起来，直到把整个数字都识别出来，例如3.5就从3 . 5 这样的顺序识别下去直到下一位不是数字，这样就获得了一个完整的数字

i++;

if (s[i] == ' ')//识别到空格时，说明上一个数字已经完整识别

{

stringstream ss;

ss << tempstr;

ss >> tempdigit;//此处操作为把字符串格式的数字转为数字

calc.push(tempdigit);//并压入栈中

}

}

switch (s[i])//如果识别到操作符

{

case '+': x = calc.top(); calc.pop(); y = calc.top(); calc.pop(); calc.push(y + x); break;//拿出栈顶两个元素，分别赋值给x，y ，做加减乘除，再压入栈中

case '-': x = calc.top(); calc.pop(); y = calc.top(); calc.pop(); calc.push(y - x); break;

case '\*': x = calc.top(); calc.pop(); y = calc.top(); calc.pop(); calc.push(y\*x); break;

case '/': x = calc.top(); calc.pop(); y = calc.top(); calc.pop(); calc.push(y / x); break;

}

i++;

}

return calc.top();//最后栈中剩下的值为结果，返回结果

}

bool iscorrect(const string &s)

{

stack<char> brackets;

for (int i = 0; i<s.size(); i++)

{

if (s[i] == '(') brackets.push(s[i]);

if (!(isdigit(s[i]) || s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/' || s[i] == '#' || s[i] == '.' || s[i] == '(' || s[i] == ')'))

{

cout << "含有非法字符，请重新输入" << endl;

return false;

}

else if (s[i] == ')')

{

if (brackets.empty())

{

cout << "括号没有成对，请重新输入" << endl;

return false;

}

else brackets.pop();

}

}

if (s == "" || s == " " || s == "\n")

{

cout << "无输入，请重新输入" << endl;

}

if (s[s.size() - 1] != '#')

{

cout << "没有以#结尾，请重新输入" << endl;

return false;

}

if (brackets.empty()) return true;

else return false;

}

int main()

{

int flag = 1;

cout << "请输入算式表达式,并以#结尾：";

string s;

while (flag)

{

cin >> s;

if (iscorrect(s)) flag = 0;

}

string r = midfixtopostfix(s);

cout << "后缀表达式为:" << r << endl;

cout << "结果为:" << calculator(r) << endl;

system("pause");

}