### 第2题：算术表达式求值

#### 数据结构：

采用了栈（stack）来实现

#### 算法设计思想：

在依次读取字符串进行运算符优先发求算术表达式的值并同时检测错误较难实现。为了方便。

先使用iscorrect（）函数读取一遍输入进来的字符串进行检测错误，若检测出错误，提示错误信息，并使用户重新输入字符串直至字符串正确。

然后利用两个栈：存放数字的数栈和操作符的操作符栈，用运算符优先法进行中缀表达式求值计算。求值调用calculator（）函数，其中获取运算符优先级用getPriority（）函数，具体的四则运算用Calculate（）函数

信息提示和错误提示分别用INFO（）函数和ERROROUT（）函数。

为了实现用栈计算算数表达式的值，需设置两个工作栈：用于存储运算符的栈opter，以及用于存储操作数及中间结果的栈opnd。

这两个栈发生变动时将实时提示相关信息。

算法基本思想如下：

（1）首先将数栈设为空栈。

（2）依次读入表达式的每个字，读入字符若是数则入数栈，读入字符若是运算符，则将此运算符c与操作符栈的栈顶元素top比较优先级后执行相应的操作，具体操作如下：

(i)若top的优先级小于c，即top<c，则将c直接入栈，并读入下一字符赋值给c；

(ii)若top的优先级等于c，即top=c，则弹出opter的栈顶元素，并读入下一字符赋值给c，这一步目的是进行括号操作;

（iii）若top优先级高于c，即top>c，则表明可以计算，此时弹出数栈顶两个元素，并且弹出操作符栈顶的的运算符，计算后将结果放入数栈中。直至当前读入的字符为‘#’，此时求值结束。输出结果。

#### 源程序：

#include <iostream>

#include <Windows.h>

#include <stack>

#include <queue>

#include <string>

#include <iomanip>

#include <sstream>

using namespace std;

#define red SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED) //红色

#define yellow SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN) //黄色

#define green SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN) //绿色

#define white SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE) //白色

#define ching SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN |FOREGROUND\_BLUE) //青色

void INFO(string info) {

cout << "["; green; cout << "INFO"; white; cout << "]:" << info ;

}

void ERROROUT(string error) {

cout << "["; red; cout << "ERROR"; white; cout << "]:" << error << endl;

}

// 由于使用运算符优先法，故要获取优先级

int getPriority(char ch) {

int level = 0; // 优先级

switch(ch) {

case '(':

level = 1;

break;

case '+':

case '-':

level = 2;

break;

case '\*':

case '/':

level = 3;

break;

default:

break;

}

return level;

}

// 进行相应的计算

double Calculate(char ch, int a, int b) {

double result = 0;

switch(ch) {

case '+':

result = b + a;

break;

case '-':

result = b - a;

break;

case '\*':

result = b \* a;

break;

case '/':

if(a == 0) {

ERROROUT( "除数不能为0。\n");

exit(1);//结束程序

}

else

result = b / a;

break;

default:

break;

}

return result; // 返回计算得到的结果

}

void calculator(const string &s){

std::stack<double> m\_integer; // 数栈

std::stack<char> op; // 操作符栈

char c; // c存储操作符栈顶的元素

int num, a, b; // result存储计算的结果,a,b存储从数栈中取出的两个数

double result;

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

ching;cout<<"当前输入的字符为：";white;cout<<s[i]<<endl;

if (s[i] == '#') continue;

if(isdigit(s[i])) { // 如果是数字

num = 0;

do {

num = num \* 10 + (s[i] - '0'); // ch - '0'根据ASCAII码，字符与数字之间的转换关系

i++;

ching;cout<<"当前输入的字符为：";white;cout<<s[i]<<endl;

}while(isdigit(s[i]));

m\_integer.push(num); // 存到数栈中

INFO("当前数已存至数栈中,该数为：");cout<<num<<endl;

}

if(s[i] == '(') { // (:左括号

op.push(s[i]);

INFO("当前左括号已存至操作符栈中\n");

}

if(s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/') { // 操作符

if(op.empty()) { // 如果栈空，直接压入栈

op.push(s[i]);

INFO("当前操作符已存至数栈中,该操作符为：");cout<<s[i]<<endl;

}

else {

// 比较栈op顶的操作符与当前操作符的优先级

// 如果当前优先级高，入栈，否则，弹出栈中的操作符，除非1.弹出操作符小于当前优先级 2.或出来( 3.或栈空

while(!op.empty()) {

c = op.top();

if(getPriority(s[i]) <= getPriority(c)) {

// 优先级低或等于

// 取出栈中操作符和数栈中两个数进行运算，再将结果放回数栈

result = 0;

a = m\_integer.top(); // 第二个操作数,因为栈是后进先出

m\_integer.pop();

INFO("有数出数栈，该数为：");cout<< a <<endl;

b = m\_integer.top(); // 第一个操作数

INFO("有数出数栈，该数为：");cout<< b <<endl;

m\_integer.pop();

result = Calculate(c, a, b); // 计算

//输出结果为两位小数

INFO("有数入数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<result<<endl;

m\_integer.push(result); // 把计算结果压入栈中

op.pop(); // 操作符出栈

INFO("有操作符出操作符栈，该操作符为");cout<< c <<endl;

} else // 当前操作符优先级高于栈中操作符

break;

} // while结束

op.push(s[i]); // 防止不断的推出操作符，最后空栈了;或者当前操作符优先级高了

INFO("当前操作符已存至数栈中,该操作符为：");cout<<s[i]<<endl;

} // else

}

if(s[i] == ')') { // 如果是右括号，一直推出栈中操作符，直到遇到左括号(

while(op.top() != '(') {

c = op.top(); // 取出栈顶操作符

// 取出栈中操作符和数栈中两个数进行运算，再将结果放回数栈

result = 0;

a = m\_integer.top(); // 第二个操作数,因为栈是后进先出

m\_integer.pop();

INFO("有数出数栈，该数为：");cout<< a <<endl;

b = m\_integer.top(); // 第一个操作数

INFO("有数出数栈，该数为：");cout<< b <<endl;

m\_integer.pop();

result = Calculate(c, a, b); // 计算

//输出结果为两位小数

INFO("有数入数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<result<<endl;

m\_integer.push(result); // 把计算结果压入栈中

INFO("有操作符出操作符栈，该操作符为");cout<< c <<endl;

op.pop(); // 把操作符推出栈

}

op.pop(); // 把左括号(推出栈

INFO("当前左括号已弹出操作符栈\n");

}

}

while(!op.empty()) { // 当栈不空，继续取出操作符进行计算

c = op.top(); // 取出栈顶操作符

// 取出栈中操作符和数栈中两个数进行运算，再将结果放回数栈

result = 0;

a = m\_integer.top(); // 第二个操作数,因为栈是后进先出

m\_integer.pop();

INFO("有数出数栈，该数为：");cout<< a <<endl;

b = m\_integer.top(); // 第一个操作数

INFO("有数出数栈，该数为：");cout<< b <<endl;

m\_integer.pop();

result = Calculate(c, a, b); // 计算

//输出结果为两位小数

INFO("有数入数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<result<<endl;

m\_integer.push(result); // 把计算结果压入栈中

INFO("有操作符出操作符栈，该操作符为");cout<< c <<endl;

op.pop(); // 把操作符推出栈

}

INFO("有数出数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<m\_integer.top()<<endl;

INFO("结果为");

cout<<setiosflags(ios::fixed)<<setprecision(2)<<m\_integer.top()<<endl;

m\_integer.pop();

}

bool iscorrect(const string &s)

{

int zuokuohao = 0;

char ch;

int onlyOPND = 0;

int isColiedOPND = 0;

stack<char> brackets;

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

{

if (i!=0 && i != s.size()-1 && s[i] == '#'){

ERROROUT( "表达式中间含有#，请重新输入" );cout << endl;

return false;

}

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

//if (s[i])

brackets.push(s[i]);

isColiedOPND == 0;

}

if (isdigit(s[i])) isColiedOPND = 1;

if (s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/'){

if (isColiedOPND == 0) {

ERROROUT("运算符位置错误使得为一目计算符或者没有任何与该运算符匹配的操作数");cout<< endl;

return false;

}

else isColiedOPND = 0;

}

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

ERROROUT( "含有非法字符，请重新输入" );cout << endl;

return false;

}

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

if (brackets.empty())

{

ERROROUT("括号没有成对，请重新输入");cout<< endl;

return false;

}

if (isColiedOPND == 0){

ERROROUT( "右括号左边无数字" );cout << endl;

Return false;

}

else brackets.pop();

}

}

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

ERROROUT("无输入，请重新输入" );cout << endl;

return false;

}

if (s[0] != '#'){

ERROROUT( "没有以#开头，请重新输入"); cout<< endl;

return false;

}

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

ERROROUT( "没有以#结尾，请重新输入"); cout<< endl;

return false;

}

if (brackets.empty()) return true;

else {

ERROROUT("括号没有成对，请重新输入");cout<< endl;

return false;

}

}

int main()

{

int flag = 1;

INFO("请输入算式表达式,以#开头，并以#结尾：\n");

string s;

while (flag)

{

cin >> s;

if (iscorrect(s)) flag = 0;

}

INFO("输出序列如下：");

cout << s <<endl;

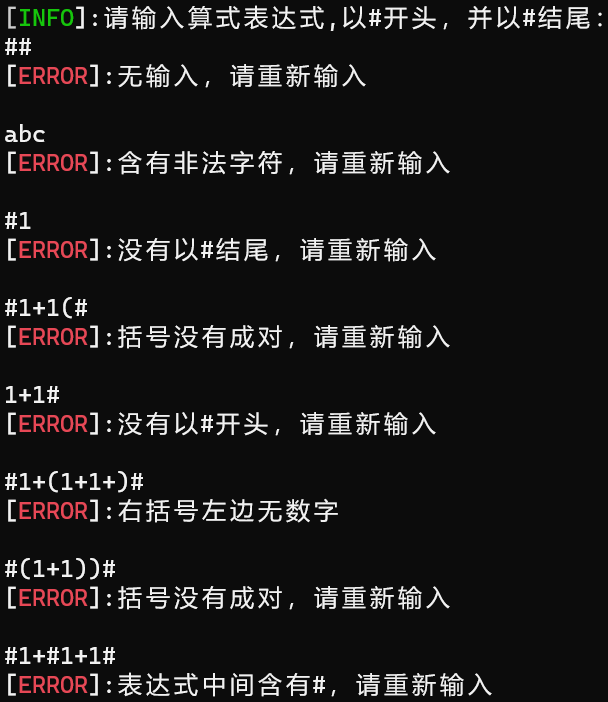
calculator(s);

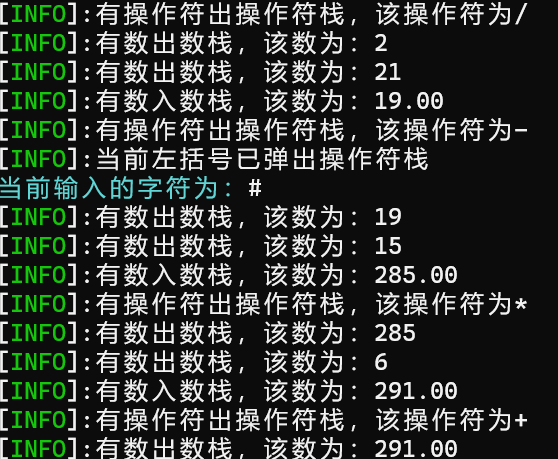
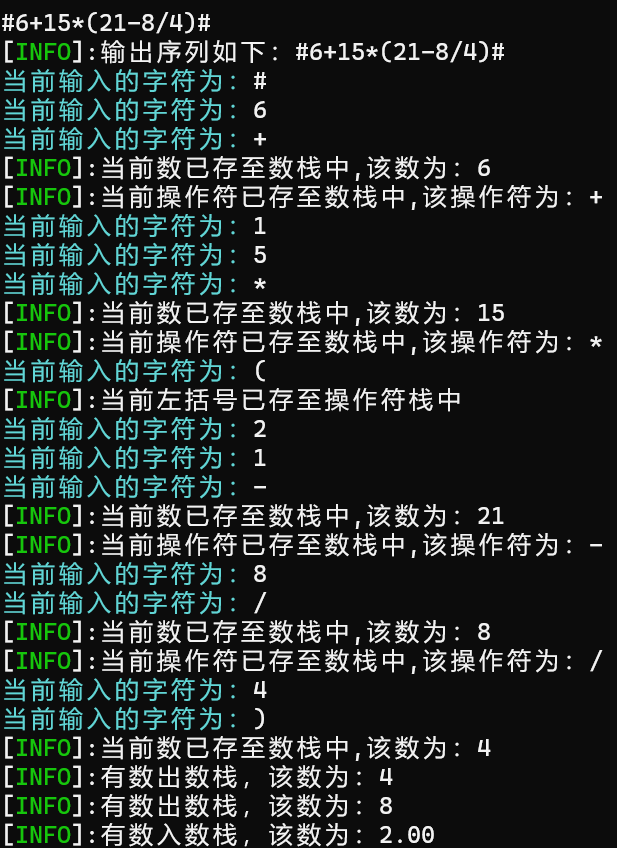
system("pause");

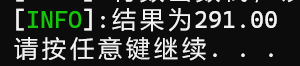
return 0;

}

#### 测试数据和结果







#### 时间复杂度：

O（N+KM+N+M），N为数字个数，M为符号个数，K为一常数，遍历N次即可获得数字，根据符号个数进行出入栈，理论上为一个线性变化的函数，故为KM，检测错误需要（N+M）。

#### 反思：

一些错误情况未细分提示。

未对小数的情况进行操作。

#### 改进：

该题代码行：

252行