**WEEK 3 Calculator**

郁思敏 10153901225

**Question:**

1. 实现一个double型栈类，然后用它完成表达式计算的程序, 支持加减乘除运算与括号。

例子：4\*(5+3\*2)

输出: 44

**Solution:**

**(1) op.h**

#ifndef OP\_H\_INCLUDED

#define OP\_H\_INCLUDED

#define MAX 100

enum Error\_code{overflow, underflow, success};

template <class T> //both operator and operand

class Stack

{

public:

Stack();

bool empty() const;

Error\_code push(const T &item);

Error\_code pop();

Error\_code top(T &item) const;

private:

int count;

T entry[MAX];

};

#endif // OP\_H\_INCLUDED

**(2) op.cpp**

#include <string>

#include "op.h"

template<class T>

Stack<T>::Stack()

{

count=0;

}

template<class T>

bool Stack<T>::empty() const

{

bool outcome=true;

if(count!=0)

outcome=false;

return outcome;

}

template<class T>

Error\_code Stack<T>::push(const T &item)

{

Error\_code outcome=success;

if(count>=MAX)

outcome=overflow;

else

entry[count++]=item;

return outcome;

}

template <class T>

Error\_code Stack<T>::pop()

{

Error\_code outcome=success;

if(count==0)

outcome=underflow;

else

count--;

return outcome;

}

template <class T>

Error\_code Stack<T>::top(T &item) const

{

Error\_code outcome=success;

if(count==0)

outcome=underflow;

else

item=entry[count-1];

return outcome;

}

**(3) main.cpp**

#include <iostream>

#include <string>

#include <cmath>

#include "op.cpp"

using namespace std;

int IsOperands(char s) //Judge whether char s is a digit.

{

if (s>='0' && s<='9')

return s-'0';

return -1;

}

double Calculate(double a, char ch, double b) //the calculation of two numbers.

{

double outcome;

switch (ch)

{

case '\*':

outcome=a\*b;

break;

case '/':

outcome=a/b;

break;

case '+':

outcome=a+b;

break;

case '-':

outcome=a-b;

break;

}

return outcome;

}

int OperatePrecedence(char s, int location) //location: in the stack:0; out of the stack:1

{

int precedence;

if(s=='+' || s=='-')

precedence=1;

else if(s=='\*' || s=='/')

precedence=2;

else if(s=='(' && location==0) //when '(' in the stack, it has the lower precedence.

precedence=0;

else if(s==')' && location==1) //when ')' is ready to push in the stack, it also has the lower precedence.

precedence=0;

else if(s=='(' && location==1) //when '(' is ready to push in the stack, it has the highest precedence.

precedence=3;

else if(s=='#') //'#' means end of the expression.

precedence=-1;

return precedence;

}

double calculate(string s)

{

int x, y; //x:precedence on the top of the stack; y:precedence of a new operator.

int cnt=-1; //record how many numbers after dot, -1 means it is not a decimal fraction.

unsigned i;

char ch;

double result=0, temp=0; //temp:for a number whose digit over 1.

double item1, item2; //the two numbers to calculate

Stack<double> operands;

Stack<char> operators;

operators.push(s[0]); //put start '#' in the stack.

for(i=1; i<s.size(); i++)

{

if(IsOperands(s[i])!=-1) //Judge whether it is a digit.

{

temp=temp\*10+s[i]-'0'; //10=1\*10+0

if(cnt>=0)

cnt++;

}

else if(s[i]=='.') //when encounter '.', change cnt to 0.

{

cnt=0;

}

else

{

if(cnt>=0) //10.1=101/pow(10,1) and push digit into stack.

{

temp/=pow(10,cnt);

operands.push(temp);

cnt=-1;

temp=0;

}

else if(cnt==-1 && temp!=0)

{

operands.push(temp);

temp=0;

}

operators.top(ch); //current top operator in operators.

x=OperatePrecedence(ch,0); //x:precedence on the top of the stack.

y=OperatePrecedence(s[i],1); //y:precedence of a new operator.

if(ch ==')') //when encounters ')' means we need to calculate the result of '()'.

{

operators.pop(); //pop ')'

operators.top(ch); //ch is the top of the operators in the stack.

operators.pop(); //when we use ch, pop ch.

result=0; //result=the sum of the number in ().

while (ch!='(') //calculate the number in () until '('.

{

operands.top(item1);

operands.pop();

result=Calculate(result,ch,item1);

operators.top(ch);

operators.pop();

}

operands.top(item1); //still one number left in (). add it to the result.

operands.pop();

result+=item1;

operands.push(result); //push the result of ().

operators.push(s[i]); //push the operator after ')'

}

else if(x<y)

{

operators.push(s[i]);

}

else

{

operators.pop(); //when '+' < '\*', pop \* and push +, then calculate +.

operators.push(s[i]);

operands.top(item2);

operands.pop();

operands.top(item1);

operands.pop();

result=Calculate(item1,ch,item2);

operands.push(result);

}

}

}

operators.top(ch); //when operators still has ops besides '#', then they must be '+' or '-'.

operators.pop();

result=0;

while (!operators.empty())

{

if(ch!='#')

{

operands.top(item1);

operands.pop();

result=Calculate(result,ch,item1);

}

operators.top(ch);

operators.pop();

}

operands.top(item1); //there is still one operand in operand.entry.

operands.pop();

result+=item1;

return result;

}

int main()

{

string s;

cout << "please enter your expression with '#' at first and last:" << endl;

getline(cin, s);

while(s[0]!='#' || s[s.size()-1]!='#')

{

cout << "input error! please input again." << endl;

cin >> s;

}

cout << calculate(s) << endl;

return 0;

}