1 Introduction

1.1 Discriptions

You are given an expression including: operators($+, -, \times, \div$), basic symbols(bracket and comma), and numbers. Calculate it.

1.2 Background of the Algorithms

Algorithms: Shunting yard algorithm.

Containers: string, vector in STL.

Other C++ Features: class, cpp style IO.

2 Algorithm Specifications

2.1 Translate the Expression into Suffix Expression

First, we should translate the given expression into suffix expression, which is easy to compute.

Define a stack of functions, and read the symbols one by one.

1. if we meet a number, push it to output.

2. if we meet an operator, keep poping the top operator, if the top is left-binded and its priority is bigger than current operator; **or the top is right-binded and its priority is smaller than current operator**, we pop the top and push it into output. Finally, push current operator into stack.

4. if we meet a left bracket, push it to the stack.

5. if we meet a right bracket, pop the top of the stack until meeting a left bracket, and pop the left bracket.

Finally, we pop the rest of the elements in the stack to output.

2.2 Build the Expression Tree and Calculate

Now we have a suffix expression, it is easy for us to build the expression tree.

Define a stack of argument, then check each symbol one by one, each time we meet a variable or number, put it into top of the stack.

Each time we meet a function, take one or two arguments from the stack, and push the result expression into the stack.

3 Testing Result

Let us show some of the test cases here. We will show the test purpose in comments.

3.1 sample input

```
10

1.43+2.323+3.2//simple pure-number expression
```

```
3*(-2.5)//negative number

1+2*3+4.4/3*5.2//test the priority

(1+2)*3-4/(3*5)//test the bracket

(2+(4/5))/(3*2)//multi bracket

((3+2)//wrong bracket

(2+3))//wrong bracket

3/0+5// divide by 0

1++++1//to many operator

+3+4//wrong arguments
```

3.2 sample output

```
Test Case #1
6.953
Test Case #2
7.5
Test Case #3
10.5455
Test Case #4
-5.25
Test Case #5
1.84615
Test Case #6
Invalid Expression
Test Case #7
Invalid Expression
Test Case #8
Invalid Expression
Test Case #9
Invalid Expression
Test Case #10
Invalid Expression
```

4 Analysis and Comments

4.1 Time : $O(n)$

Here n means the length of the input.

The shunting yard algorithm's time complexity is $O(n)$. We read each symbol once, and each symbol is push into and pop out of the stack once.

4.2 Space: $O(n^2)$

The space complexity is same as the expression tree, which is same as the length of the result, which is $O(n)$, similarly.

5 Source Code (in cpp)

5.1 expression_evaluator.h

```
#include<bits/stdc++.h>
//standard functions we need
//cpp style I/O
const double eps=1e-9;
using std::cerr;
using std::cin;//input
using std::cout;//output
using std::endl;//'\n'
//stl containers
using std::vector;//array with dynamic space
using std::map;//A standard container made up of (key,value) pairs
using std::stack;//A standard container giving FILO behavior.
using std::string;//A string of @c char
//random function
std::mt19937 rnd;
//hash for unsigned long long
using ull = unsigned long long;


//the Symbol in input expression
struct Sym{
    string x;// the symbol itself
    string type;//type: number,variable,function.
};
//Operators and its priority.
const map<string,int>mp={
    {"(",-1},{"+",0},{"-",0},{"*",1},{"/",1},{"^",2},
};
//Numbers of function's␣arguments.
const map<string,int>args_cnt={
    {"+",2},{"-",2},{"*",2},{"/",2},{"^",2}};


//this functions transform the input expression into a suffix expression.
//parameters:the input string
//return:the suffix expression, as a vector which contains type Sym.
vector<Sym> trans_Expr(string &s){
    for(auto i=s.begin();i!=s.end();i++){
        if(i==s.begin()){
            if(*i=='-'){
                s="0"+s;
            }
        }
        else{
            if(*i=='-'&&*prev(i)=='('){

s=s.substr(0,i-s.begin())+"0"+s.substr(i-s.begin(),s.end()-s.begin());
            }
        }
    }
    vector<Sym>suf;
```

```
    auto p=s.begin();
    stack<string>op;//the stack of functions
    while(p!=s.end()){
        string res="";
        if(*p<='9'&&*p>='0'){//if next symbol is a number:
            while(p!=s.end()&&((*p<='9'&&*p>='0')||*p=='.'))res+=*p,++p;
            suf.push_back({res,"number"});
        }
        else if(*p=='('){//if next symbol is a left bracket:
            res+=*p;++p;
            op.push(res);
        }
        else if(*p==')'){//if next symbol is a right bracket:
            res+=*p;++p;
            while(!op.empty()&&op.top().back()!='('){
                suf.push_back({op.top(),"func"});
                op.pop();
            }
            if(op.empty()){//check bracket
                cout<<"Invalid␣Expression"<<endl;
                return vector<Sym>();
            }
            op.pop();
        }
        else if(mp.find((string)""+*p)!=mp.end()){//if next symbol is +-*/^
            res+=*p;++p;
            while(!op.empty()&&

(mp.find(op.top())->second>=mp.find(res)->second)==(op.top()!="^")){
                suf.push_back({op.top(),"func"});
                op.pop();
            }
            op.push(res);
        }
    }
    while(!op.empty()){
        suf.push_back({op.top(),"func"});
        op.pop();
    }
    return suf;
}


//this function builds the expression tree and calculate at the same time.
//parameter: the input string of the expression
//return: ans
double calculate(string &a){
    vector<Sym>suf=trans_Expr(a);
    if(suf.size()==0)return 1e9;
    vector<double>args;
    for(auto v:suf){
        if(v.type=="number"){//Add numbers and variables to the stack of
parameters
            args.push_back(std::stof(v.x));//the function stoi(string)
translate a string to an double.
        }
```

```
        else{//the functions with two parameters.
            if(args.size()<2){
                cout<<"Invalid Expression"<<endl;
                return 1e9;
            }
            double a=args.back();args.pop_back();
            double b=args.back();args.pop_back();
            if(v.x=="+"){
                args.push_back(a+b);
            }
            else if(v.x=="-"){
                args.push_back(a-b);
            }
            else if(v.x=="*"){
                args.push_back(a*b);
            }
            else if(v.x=="/"){
                if(a<eps){//check /0
                    cout<<"Invalid Expression"<<endl;
                    return 1e9;
                }
                args.push_back(a/b);//translate / into *
            }
        }
    }
    return args.back();
}
    string type;//type: number,variable,function.
```