实验一 逆波兰表达式

郑戈涵 17338233

算法描述

后缀表达式的特点是运算符在运算数的后面，题目要求的算术表达式示例是中缀表达式。因此目标是将中缀表达式转换为后缀表达式。

算法使用栈实现，利用到下面这个运算符优先级的表，表中第i行第j列的结果用prior(i,j)表示。#代表终止符。

lhs / rhs	+	-	*	/	#	(	)
+	>	>	<	<	>	<	>
-	>	>	<	<	>	<	>
*	>	>	>	>	>	<	>
/	>	>	>	>	>	<	>
#	<	<	<	<	=	<	=
(	<	<	<	<	=	<	=
)	=	=	=	=	=	=	=

伪代码

```
function in2post(expression expr)
    init 2 stacks,numStack for expression and opStack for operator
    add endmark '^' as input's tail
    scan input expr
        if expr[curr] is space
            continue
        else if expr[curr] is digit
            num<-getCurrNum(expr,curr)
            numStack.push(num)
        else if opStack is not empty
            op<-expr[curr]
            res<-prior(opStack.top(),op)
            if res='<'
                opStack.push(op)
            else if res='='
                opStack.pop(op)
            else
                do
                    rhs<-numStack.pop()
                    lhs<-numStack.pop()
                    numStack.push(lhs+' '+rhs+' '+opStack.pop())
                while(opStack is not empty and prior(opStack.top(),op)='>')
                if opStack is not empty and opStack.top()='(' and expr[curr]=')'
                        opStack.pop()
                if op!=')'
                    opStack.push(op)
        else if expr[curr] is operator
            opStack.push(expr[curr])
        else
            return error
end
```

C++代码

```
#include <stack>
#include <string>
#include <map>
#include <cctype>
#include <exception>
#include <iostream>
using namespace std;
const string oper="+-*/()#";
char opOrder[][8] = {">><<><>", ">><<><>", ">>>>><>", ">>>>><>", "<<<<=<=","
<<<<=<=", "======="};

map<char,int> op2id={{'+',0},{'-',1},{'*',2},{'/',3},{'#',4},{'(',5},{')',6}};

string splitNum(string s,int &i){
    string intStr;
    for (i=0; i < s.size(); i++)
    {
        if(oper.find(s[i])==string::npos)
            intStr.push_back(s[i]);
        else
            break;
    }
    if(intStr.size()>0){
        return intStr;
    }
    else return "";
}
char lt(char lhs,char rhs){
    try
    {
        return opOrder[op2id[lhs]][op2id[rhs]];
    }
    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
        return 0;
    }
}

string convert(string &s){
    s+="#";
    string res;
    stack<string>numStack;
    stack<char>opStack;
    int strSize=s.size();
    int index=0;
    string tmpInt;
```

```cpp
    int subIndex;
    while(index<strSize){
        subIndex=0;
        char op=s[index];
        if(isspace(op)){
            index++;
            continue;
        }
        else if ((tmpInt=splitNum(s.substr(index),subIndex))!=""){
            numStack.push(tmpInt);
            index+=subIndex;
        }
        else if(!opStack.empty()){
            char cmpRes=lt(opStack.top(),op);
            if(cmpRes=='<'){
                opStack.push(op);
            }
            else if(cmpRes=='='){
                opStack.pop();
            }
            else{
                do
                {
                    string lhs=numStack.top();
                    numStack.pop();
                    string rhs=numStack.top();
                    numStack.pop();
                    res=rhs+' '+lhs+' '+opStack.top();
                    opStack.pop();
                    numStack.push(res);
                } while (!opStack.empty()&&lt(opStack.top(),op)=='>');
                if(!opStack.empty()&&opStack.top()=='('&&op==')')opStack.pop();
                if(op!=')')
                    opStack.push(op);
            }
            index++;
        }
        else if(oper.find(op)!=string::npos){
            opStack.push(op);
            index++;
        }
        else
            throw invalid_argument("bad input!");
    }

    return numStack.top();
}

int main(){
    string s;
    while(getline(cin,s))
        cout<<convert(s)<<endl;
}
```