栈

1. 有效的括号

思路分析：括号匹配不成功共有三种情况

第一种情况：遍历完了字符串，但是栈不为空，说明左括号多了

第二种情况：访问字符串的过程中栈已经空了，说明右括号多了

第三种情况：访问过程中栈不为空，但是左右括号不匹配

 bool isValid(string s) {

        stack<char> st;

        for(int i=0;i<s.size();i++){

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

                st.push(s[i]);

            }

            else{ //说明遍历到的是右括号

第二种情况//如果遍历到右括号的时候栈为空 说明栈里面没有左括号 则直接返回否定

                if(st.empty()){

                    return false;

                }

第三种情况//

                char x=st.top();

                if((x=='('&&s[i]!=')')||(x=='{'&&s[i]!='}')||(x=='['&&s[i]!=']')){

                    return false;

                }

                st.pop();

            }

        }

第一种情况//

        return st.empty();

    }

1. 逆波兰表达式

注：字符串是双引号，字符是单引号

遇到操作数就压栈，遇到运算符就进行出两次栈，先出的为操作符的右数，后出的为操作符的左数，然后将运算的结果进行压栈，最后留在栈里的数据就是最终的结果

private:

    stack<int> st;

public:

    int evalRPN(vector<string>& tokens) {

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

//这里是一个逆向思维：不好判断是否为数字字符串，则判断是否为运算符

            if(tokens[i]!="+"&&tokens[i]!="-"&&tokens[i]!="\*"&&tokens[i]!="/"){

                st.push(stoll(tokens[i]));      //stoll将字符转化为数字

            }

            else{

                int right\_val=st.top();

                st.pop();

                int left\_val=st.top();

                st.pop();

                int res;

                if(tokens[i]=="+"){

                    res=left\_val+right\_val;

                }

                else if(tokens[i]=="-"){

                    res=left\_val-right\_val;

                }

                else if(tokens[i]=="\*"){

                    res=left\_val\*right\_val;

                }

                else if(tokens[i]=="/"){

                    res=left\_val/right\_val;

                }

                st.push(res);

            }

        }

        return st.top();

    }

1. 中缀表达式转后缀
2. 需要用两个栈s1和s2
3. 扫描到左括号直接入栈s2，如果栈顶元素是左括号，运算符无条件入栈（将左括号设置为0）
4. 扫描到操作数直接入栈s1（直接加到结果字符串里）
5. 扫描到右括号时将s2中从栈顶元素到左括号之间的依次出栈s2，入s1.。并将左括号舍弃
6. 扫描到的运算符元素的优先级大于s2栈顶元素的优先级，扫描到的运算符直接入s2；反之（小于等于）则将s2栈顶元素出栈，并入栈s1；然后再将扫描到的运算符元素与栈顶元素比较；

下面的代码本质上是两个栈，但是存放操作数的栈可以不用而直接输出

#include <iostream>

#include <stack>

#include <string>

using namespace std;

//判断是不是运算符

bool isOperator(char c) {

if (c == '+' || c == '-' || c == '\*' || c == '/')

return true;

return false;

}

//比较运算符优先级的函数

int precedence(char op) {

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

return 1;

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

return 2;

return 0; //左括号就用0代表 //将“如果是左括号直接压栈”巧妙处理了，0比1和2都小

}

//中缀转后缀表达式函数

string infixToPostfix(string infix) {

string postfix = "";

stack<char> s;

for (int i = 0; i < infix.length()-1; i++) {

char c = infix[i];

if (isalpha(c)) { //isalpha判断是不是字母的函数，无论大小写

postfix += c;

}

else if (c == '(') {

s.push(c);

}

else if (c == ')') {

while (!s.empty() && s.top() != '(') {

postfix += s.top();

s.pop();

}

s.pop(); //删除左括号

}

else if (isOperator(c)) {

while (!s.empty() && precedence(c) <= precedence(s.top())) {

postfix += s.top();

s.pop();

}

s.push(c);

}

}

while (!s.empty()) {

postfix += s.top();

s.pop();

}

return postfix;

}

int main() {

string infix;

cin >> infix;

string postfix = infixToPostfix(infix);

cout << postfix << endl;

return 0;

}

1. 中缀表达式转前缀

5.数字转换

坑比较多

    string toHex(int num) {

        if(num==0){

            return "0";

        }

        stack<char>st;

        //对负数的补码

        //将数转换成无符号整数

*//自动就是补码形式*

        unsigned int uint\_num=(unsigned int)num;

        while(uint\_num){       //num不为0说明没有转换完

            int mod=uint\_num%16;

            if(mod>=10)

                st.push(mod-10+'a'); //利用ASCLL码进行转换

            else

                st.push(mod+'0');   //利用ASCLL码进行字符转换

            uint\_num/=16;

        }

        string s="";

        while(!st.empty()){

            s+=st.top();

            st.pop();

        }

        return s;

    }