1. Basic Calculator

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ), the plus + or minus sign -, **non-negative** integers and empty spaces ``.

**Example 1:**

Input: "1 + 1"  
Output: 2

**Example 2:**

Input: " 2-1 + 2 "  
Output: 3

**Example 3:**

Input: "(1+(4+5+2)-3)+(6+8)"  
Output: 23

**Note:**

* You may assume that the given expression is always valid.
* **Do not** use the eval built-in library function.

**解** 栈的应用

注意设置优先级时，左括号最高，右括号最低。实现过程如下：

* 如果当前是数字，那么更新计算当前数字；
* 如果当前是操作符+或者-，那么需要更新计算当前计算的结果res，并把当前数字num设为0，sign设为正负，重新开始；
* 如果当前是(，那么说明后面的小括号里的内容需要优先计算，所以要把res，sign进栈，更新res和sign为新的开始；
* 如果当前是)，那么说明当前括号里的内容已经计算完毕，所以要把之前的结果出栈，然后计算整个式子的结果；
* 最后，当所有数字结束的时候，需要把结果进行计算，确保结果是正确的。

class Solution {  
public:  
 unordered\_map<char, int>prior{{'+', 1}, {'-', 1}, {'\*', 2}, {'/', 2}, {'(', 3}, {')', 0}};  
 int calculate(string s) {  
 s += "#";  
 stack<int>nums;  
 stack<char>op;  
 string tmp = "";  
 for(int i = 0; i < s.size(); ++i){  
 if(s[i] == ' ')continue;  
 if(isdigit(s[i])){  
 tmp += s[i];  
 }else{  
 if(tmp.size() > 0){  
 nums.push(stoi(tmp));  
 tmp = "";  
 }  
 if(s[i] == '#')continue;  
 while(!op.empty() && op.top() != '(' && prior[op.top()] >= prior[s[i]]){  
 int n1 = nums.top();  
 nums.pop();  
 int n2 = nums.top();  
 nums.pop();  
 char ch = op.top();  
 op.pop();  
 nums.push(compute(n2, n1, ch));  
 }  
 if(s[i] == ')')op.pop();  
 else op.push(s[i]);  
 }  
 }  
 while(!op.empty()){  
 int n1 = nums.top();  
 nums.pop();  
 int n2 = nums.top();  
 nums.pop();  
 char ch = op.top();  
 op.pop();  
 nums.push(compute(n2, n1, ch));  
 }  
 return nums.top();  
 }  
 int compute(int n1, int n2, char op){  
 if(op == '+')return n1 + n2;  
 if(op == '-')return n1 - n2;  
 if(op == '\*')return n1 \* n2;  
 if(op == '/')return n1 / n2;  
 return -1;  
 }  
};