学 院：数据科学与计算机学院　　专 业：计算机类
姓 名：郑康泽　　学 号：17341213

编译原理

实验一：将算术表达式转化为逆波兰表达式

一. 算法描述（不支持负数，支持小数）

初始化逆波兰表达式为空串，从左到右扫描算术表达式，如果扫描到是数字或者是小数点，则将该字符添加到逆波兰表达式末尾；如果扫描到的是运算符，则分为以下情况：

- 该运算符为左括号"("，则直接存入堆栈中；
- 该运算符为右括号")"，则从上往下将堆栈中的运算符添加到逆波兰表达式末尾并弹出栈，直至栈顶为左括号"("，不将左括号"("添加进逆波兰表达式，直接弹出栈即可；
- 该运算符为加减乘除运算符，则从上往下将堆栈中优先级高于或等于该运算符优先级的运算符添加到逆波兰表达式末尾并弹出，直至堆栈为空或者栈顶为左括号"("或者栈顶的运算符优先级低于该运算符优先级，然后将该运算符存入栈中。

当算术表达式扫描完后，如果堆栈中还有运算符，则从上到下依次将运算符添加到逆波兰表达式末尾。最终就得到了该算术表达式的逆波兰表达式。

二. 代码解释

因为我们需要将数字与数字、数字与运算符分隔开，所以我们需要适时添加一些空格，添加空格的方法如下：

- 如果扫描到加减乘除运算符，则添加一个空格到逆波兰表达式，因为一旦扫描到有加减乘除运算符，说明有第二个操作数（第一个操作数显然就是运算符前面的操作数），所以就要有个空格隔开两个操作数。当然，根据第一部分的算法描述，我们知道可能会将栈中一些运算符添加到逆波兰表达式中，所以这个空格也可能是隔开第一个操作数和第一个被弹出的的运算符，并且栈中的运算符添加到逆波兰表达式后，也要加一个空格，以便隔开栈中弹出的相邻的运算符以及最后一个弹出的运算符和第二个操作数；
- 如果扫描到右括号")"，要先添加空格到逆波兰表达式，再添加弹出的运算符，以便隔开小括号中最后一个操作数和第一个被弹出的运算符以及栈中弹出的相邻的运算符，不用担心最后是不是少一个空格，因为右括号")"后面要么是加减乘除运算符（见上面加空格方法），要么是右括号")"（见当前加括号方法），要么就是表达式结束，不需要空格了；
- 当扫描完算术表达式，如果堆栈中还有运算符，添加空格的方法同扫描到右括号")"。

代码如下：

```
string RPN(string s) {
    stack<char> st;     // 存放运算符
    string ans = "";        // 最终的逆波兰表达式
    int l = s.length();     // 算数表达式字符串的长度

    for (int i=0; i<l; ++i) {
        // 操作数部分
        if ((s[i] >= '0' && s[i] <= '9') || s[i] == '.') ans += s[i];
        // 左括号直接进栈
        else if (s[i] == '(')   st.push(s[i]);
        // 右括号，出栈至 左括号
        else if (s[i] == ')') {
            while (!st.empty() && st.top() != '(') {
                ans += ' ';
                ans += st.top();
                st.pop();
            }
            if (st.empty()) return "表达式无效";     // 没有对应的左括号
            st.pop();       // 弹出左括号
        }
        // 乘除运算符
        else if (s[i] == '*' || s[i] == '/') {
            ans += ' ';
            // 出栈至 加减运算符 或者 左括号 或者 栈为空
            while (!st.empty() && st.top()!= '(' && st.top() != '+' && st.top()
!= '-') {
                ans += st.top();
                ans += ' ';
                st.pop();
```

```
            }
            st.push(s[i]);      // 当前运算符进栈
        }
        // 加减运算符
        else if (s[i] == '+' || s[i] == '-') {
            ans += ' ';
            // 栈至 左括号 或者 栈为空
            while (!st.empty() && st.top() != '(') {
                ans += st.top();
                ans += ' ';
                st.pop();
            }
            st.push(s[i]);      // 当前运算符进栈
        }
    }

    // 表达式扫描完，栈中还有运算符
    while (!st.empty() && st.top() != '(') {
        ans += ' ';
        ans += st.top();
        st.pop();
    }
    if (!st.empty()) return "表达式无效";        // 没有对应的右括号

    return ans;
}
```