实验一：将算术表达式转化为逆波兰表达式

18340166　　王若琪

1. 算法描述

将中缀表达式转化为后缀表达式时需要用到栈结构，具体过程如下：

- 首先初始化一个空栈。
- 从左到右遍历中缀表达式中的每一个字符，如果遇到数，就将他们直接添加进要输出的后缀表达式中，在实现过程中，还需考虑几个字符组成一个数，根据每次遇到的数字字符的下一个是否还为数字字符来确定数字是否结束，输出的后缀表达式用空格分隔开数字和运算符。
- 如果遇到左括号 '(' ，就直接进栈。
- 如果遇到右括号 ')' ，就一直将栈里的符号出栈，并将这些符号添加进要输出的后缀表达式中，直到遇到左括号 '(' 为止，其中这个左括号也要出栈，但不添加进要输出的后缀表达式中。
- 如果遇到操作符 '+' , '-' , '*', '/'，那么继续进行如下判断：
  - 先判断栈是不是空的，如果是空栈，就直接进栈。
  - 如果不是空栈，就将此操作符与栈顶符号比较优先级：
    - 如果此操作符的优先级小于或等于栈顶操作符的优先级时，那么将该栈顶符号放入后缀表达式，并且弹出该栈顶符号。反复执行此过程直到当前的操作符的优先级大于栈顶操作符的优先级。
    - 如果此操作符优先级大于栈顶操作符，那么将此操作符进栈。
- 重复以上步骤直到中缀表达式遍历完成。
- 当遍历完中缀表达式后，如果栈中还有剩余的操作符，就依次弹出并放入后缀表达式中，直到栈空。

2. 结果展示

```
Enter the infix expression: 3*(4+5/(2-1))
Postfix expression: 3 4 5 2 1 - / + *

Enter the infix expression: 21+42-30/(5+5)*(4-2)
Postfix expression: 21 42 + 30 5 5 + / 4 2 - * -
```

3. 附录 (源代码)

```cpp
#include <iostream>
#include <stack>
using namespace std;

int get_priority(char ch) //判断符号的优先级
{
    if (ch == '*' || ch == '/')
        return 2;
    else if (ch == '+' || ch == '-')
        return 1;
    else
```

```
        return 0;
}

int main()
{
    while (1)
    {
        string infix, postfix; //中缀，后缀
        stack<char> char_stack;
        cout << "Enter the infix expression: ";
        cin >> infix;
        if (infix == "EXIT")
            break;
        int lenth = infix.size();
        for (int i = 0; i < lenth; i++) //从左往右遍历中缀表达式
        {
            if (isdigit(infix[i])) //如果是数字
            {
                postfix += infix[i];
                //如果这个数字结束了，后缀表达式中加空格
                if (i == lenth - 1)
                    postfix += ' ';
                else if (!isdigit(infix[i + 1]))
                    postfix += ' ';
            }
            else if (infix[i] == '(') //如果左括号直接进栈
            {
                char_stack.push(infix[i]);
            }
            else if (infix[i] == ')') //如果右括号，一直出栈直到遇到左括号
            {
                while (char_stack.top() != '(')
                {
                    postfix = postfix + char_stack.top() + ' ';
                    char_stack.pop();
                }
                char_stack.pop();
            }
            else if (infix[i] == '+' || infix[i] == '-' || infix[i] == '*'
|| infix[i] == '/')
            {
                if (char_stack.empty()) //如果空栈就push进栈
                {
                    char_stack.push(infix[i]);
                }
                else //如果不空，与栈顶符号比较优先级
                {
                    while (!char_stack.empty())
                    {
                        if (get_priority(infix[i]) <=
get_priority(char_stack.top())) //如果优先级低于等于栈顶的符号，栈顶元素就出栈直到不满足条件
                        {
                            postfix = postfix + char_stack.top() + ' ';
                            char_stack.pop();
                        }
                        else
                            break;
```

```
                    }
                    char_stack.push(infix[i]); //进栈
                }
            }
        }
        while (!char_stack.empty())
        {
            postfix = postfix + char_stack.top() + ' ';
            char_stack.pop();
        }
        cout << "Postfix expression: " << postfix << endl
             << endl;
    }
}
```