学院：数据科学与计算机学院　　专业：计算机科学与技术
姓名：郑康泽　　学号：17341213

编译原理

实验二：递归算法

一. 题目

根据如下算术表达式的递归定义，编写C++递归程序，将算术表达式转化为逆波兰表达式（后缀表达式）。算术表达式E的递归定义如下：

E -> E+T | E-T | T　（表达式定义为：表达式+项 或者 表达式-项 或者 项）

T -> T*F | T/F | F　（项定义为：项乘以因子 或者 项除以因子 或者 因子）

F -> (E) | value　（因子定义为：（表达式）或者 数值）

二. 算法描述

根据算术表达式E的递归定义，我们可以知道E是由多个T通过+或者-串起来的，而T是由F通过*或者/串起来的，而F是由小括号括起来的一个子算术表达式或者一个数值。为了将算术表达式转化成逆波兰表达式，我们需要根据递归定义将算术表达式分解，并移动相应的运算符。转换函数的伪代码如下：

```
function Expr(E, Start, End)
Input: E is an arithmetic expression. Start and End are subsrcipts. And
       E(Start) and E(End) represents "+" or "-" or "*" or "/" or "(" or ")"
       or a numeric number.
Output: Print out the Reverse Polish notation of E.

find the subscript i (Start<=i<=End) of the last "+" or "-" that is not in any
parentheses;
```

```
if i is not found then  // E is a term
    T(E, Start, End);
else
    E(ps, Start, i-1);
    T(ps, i+1, End);
    print out E(i) and a blank space;


// auxiliary function
function Expr(E, Start, End)
Input: E is an arithmetic expression. Start and End are subsrcipts. And
       E(Start) and E(End) represents "+" or "-" or "*" or "/" or "(" or ")"
       or a numeric number.
Output: Print out the Reverse Polish notation of one part of E which is called
      Term.

find the subscript i (Start<=i<=End) of the last "*" or "/" that is not in any
parentheses;
if i is not found then  // E is a factor
    F(E, Start, End);
else
    T(ps, Start, i-1);
    F(ps, i+1, End);
    print out E(i) and a blank space;

function Expr(E, Start, End)
Input: E is an arithmetic expression. Start and End are subsrcipts. And
       E(Start) and E(End) represents "+" or "-" or "*" or "/" or "(" or ")"
       or a numeric number.
Output: Print out the Reverse Polish notation of one part of E which is called
      Factor.

if E(Start) = "(" then
    Expr(E, Start+1, End-1);
else
    print out E(Start) and a blank space;
```

Expr函数首先找到算术表达式E中最后一个项T，记去掉最后一项后的算术表达式为E1，那么当前的算术表达式就可以暂时转换为E1 T +|-，对于最后一项T，通过函数Term继续将其分解为项中最后一个因子F以及剩下的部分T1，那么当前的算术表达式就可以暂时转换为E1 T1 F *|/ +|-，接下来继续利用函数Factor处理F。如果F只是一个数字，那么将之输出即可，如果F是一个小括号括起来的子表达式，那么就要调用函数Expr处理该子表达式，这样就处理完项T中最后一个因子F，项T中其他因子的处理方法一样，同理表达式E中其他项的处理也一样，所以就可以通过递归不断调用自己来减小问题的规模。（实际上，E1比T早处理，T1也比F早处理，这里只是为了方便叙述）

三. 代码解释

1. 将输入的算术表达式处理成伪代码中E:

```
/*
根据此处预处理的方法，所有带符号的数字都需要带括号，例如-2需要写成(-2)
*/
string* preprocess(string s, int& len) {
    /*
    s：输入的算术表达式
    l：预处理后的算术表达式的长度
    */
    string* ps = new string [500];
    int l = s.length();
    bool last_is_num = 0;
    len = 0;
    for (int i=0; i<l; ++i) {
        // 操作数
        if ((s[i] >= '0' && s[i] <= '9') || s[i] == '.'){
            ps[len] += s[i];
            last_is_num = 1;
        }
        // 操作符
        else if (s[i] == '+' || s[i] == '-' || s[i] == '*' || s[i] == '/' ||
s[i] == '(' || s[i] == ')'){
            if (last_is_num) {
                len++;
                last_is_num = 0;
            }
            // 负数
            if (s[i-1] == '(' && (s[i] == '+' || s[i] == '-')) {
                int j;
                for (j=i; j<l && s[j] != ')'; ++j)
                    ps[len] += s[j];
                i = j-1;
                len++;
            }
            else
                ps[len++] = s[i];
        }
        // 无效字符
        else if (s[i] != ' ') return NULL;
    }
    return ps;
}
```

即将原来是一个字符串的算术表达式中的每一个操作数或者操作符拆出来，单独存为一个字符串，也就是原来的算术表达式用一个字符串数组存起来。

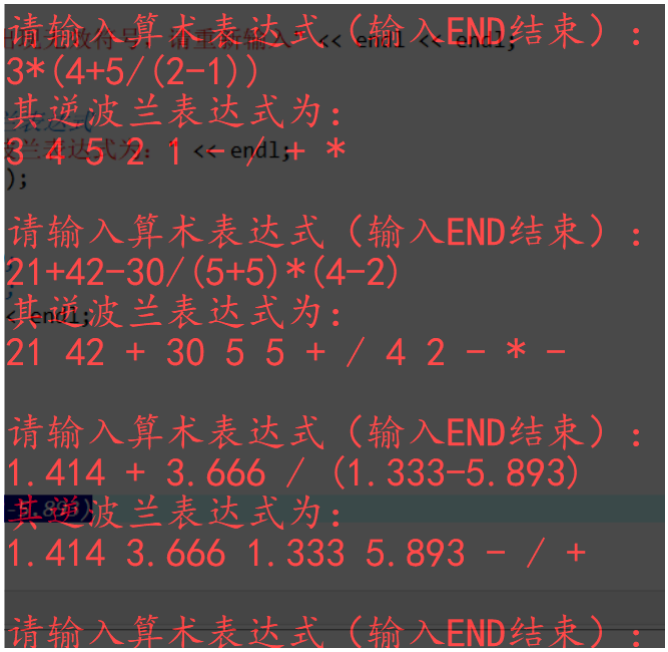
2. 将算术表达式转换为逆波兰表达式的函数：

```
void E(string*, int, int);
```

```
void T(string*, int, int);
void F(string*, int, int);

void E(string* ps, int start, int end) {
    int i;
    int parentheses = 0;
    // 不在括号中的最后一个加号或减号
    for (i=end; i>=start; --i) {
        if (ps[i] == "(") parentheses--;
        if (ps[i] == ")") parentheses++;
        if (!parentheses && (ps[i] == "+" || ps[i] == "-")) break;
    }
    if (i >=start) {
        E(ps, start, i-1);
        T(ps, i+1, end);
        cout << ps[i] << ' ';
    }
    else T(ps, start, end);
}

void T(string* ps, int start, int end) {
    int i;
    int parentheses = 0;
    // 不在括号中的最后一个乘号或除号
    for (i=end; i>=start; --i) {
        if (ps[i] == "(") parentheses--;
        if (ps[i] == ")") parentheses++;
        if (!parentheses && (ps[i] == "*" || ps[i] == "/")) break;
    }
    if (i >= start) {
        T(ps, start, i-1);
        F(ps, i+1, end);
        cout << ps[i] << ' ';
    }
    else F(ps, start, end);
}

void F(string* ps, int start, int end) {
    if (ps[start] == "(") E(ps, start+1, end-1);
    else cout << ps[start] << ' ';
}
```

具体解释可见[第二部分]。

四. 结果展示


请输入算术表达式（输入END结束）：
3*(4+5/(2-1))
其逆波兰表达式为：
3 4 5 2 1 - / + *
请输入算术表达式（输入END结束）：
21+42-30/(5+5)*(4-2)
其逆波兰表达式为：
21 42 + 30 5 5 + / 4 2 - * -
请输入算术表达式（输入END结束）：
1.414 + 3.666 / (1.333-5.893)
其逆波兰表达式为：
1.414 3.666 1.333 5.893 - / +
请输入算术表达式（输入END结束）：