实验四 对算术表达式的递归下降分析

一、实验内容

【任务性质】必做任务，分值10分

【任务介绍】根据给定的上下文无关文法，分析任意一个算术表达式的语法结构

【输入】任意的算术表达式

【输出】与输入对应的一颗语法树或者错误

【题目】设计一个程序，根据给定给的上下文无关文法，构造一颗语法树来表达任意一个算术表达式的语法结构。要求：

1. 基础文法：

```
<E> -> <T><E1>
<E1> -> <A><T><E1> | empty
<T> -> <F> <T1>
<T1> -> <M><F><T1> | empty
<F> -> id | number | (<E>)
<A> -> + | -
<M> -> * | /
```

2. 语法分析方法采用递归子程序法
3. 输入：形如

```
a+b*2/4-(b+c)*3
```

   的算术表达式，有+、-、*、/四种运算符，运算符的优先级、结合规则和括号的用法结合惯例，有变量、整数两种运算对象。为简化问题，变量和整数均为只含有1个字符的单词，忽略空格等非必要的字符。

4. 输出：输入正确时，输出其对应的语法树，树根标记为<E>；输入错误时，输出error。

二、实验思路

（一）递归下降分析

1. 递归下降分析过程

- 通过递归下降分析，判断输入的算术表达式是否符合文法，如果符合文法构建语法树，若不符合，直接报错，终止程序
- 为每个产生式写一个过程
- 因为文法中包含空产生式，所以需要求FIRST和FOLLOW，根据这两个集合编写各个产生式的处理过程

  FIRST如下：

```
FIRST(E)  = {id, number, (}
FIRST(E1) = {空串, +, -}
FIRST(T)  = {id, number, (}
FIRST(T1) = {空串, *, /}
FIRST(F)  = {id, number, (}
FIRST(A)  = {+, -}
FIRST(M)  = {*, /}
```

FOLLOW如下：

```
FOLLOW(E)  = {$, )}
FOLLOW(E1) = {$, )}
FOLLOW(T)  = {+, -, $, )}
FOLLOW(T1) = {+, -, $, )}
FOLLOW(F)  = {*, /, +, -, $, )}
FOLLOW(A)  = {id, number, (}
FOLLOW(M)  = {id, number, (}
```

2. 每个产生式伪代码

- 过程A

```
procedure A(token) {
    IF token='+' or '-' THEN RETURN;
    ELSE ERROR;
}
```

- 过程F

```
procedure F(token) {
    IF token=id or number THEN RETURN;
    ELSE IF token='(' THEN
        c <- getNext(); //字符串下一个字符
        E(C);
        c <- getNext();
        IF c!=')' ERROR;
    ELSE THEN ERROR;
}
```

- 过程T1

```
procedure T1(token) {
    IF token='+' or '-' or ')' or '$' THEN RETURN; //token在T1的FOLLOW中
    ELSE THEN
        M(token);
        c <- getNext();
        F(c);
        c <- getNext();
        1(c);
}
```

- 过程T、E1、E处理过程类似上述

3. 数据结构与算法

- 整个算术表达式用C++中的string s表示
- 用cnt全局变量和getNext函数，返回需要读取的s中下一个字符
- 处理空产生式时，一定要记得匹配上empty之后cnt减一，因为当前字符实际上没有被匹配，需要被进一步处理
- C++文件开头添加函数声明，可以在此后任意位置使用此函数

（二）建立语法树

思路：中序后序转层序

- 此实验的输入是一个中缀表达式就是一棵二叉语法树的中序遍历
- 给出一棵二叉树的中序和后序，这棵树就能确定下来
- (PAT甲级1020就是中序后序转层序，所以这个实验也想试一下)

1. 中缀表达式转后序表达式（借助栈）

- 从左到右遍历中缀表达式
- 运算数：输出
- 左括号：入栈（入栈后括号优先级降为最低，确保其他符号正确入栈）
- 右括号：意味一组括号结束，不断弹出栈顶运算符并输出，直到遇到左括号，左括号仅弹出不输出
- 运算符：
  - 优先级>栈顶运算符：入栈
  - 优先级<=栈顶运算符：将栈顶运算符弹出并输出，继续比较，直到优先级大于栈顶运算符或者栈空,再将该运算符入栈。（低于弹出意味着前面部分可以运算,先输出的一定是高优先级运算符,等于弹出是因为同等优先级,从左到右运算）
- 表达式处理完毕，若栈中仍有运算符，按出栈顺序输出

2. 中序后序转层序

- 与后序中序转换为前序的代码相仿（无须构造二叉树再进行广度优先搜索，此算法见下方ps），只不过加一个变量index，表示当前的根结点在二叉树中所对应的下标（从0开始），所以进行一次输出先序的递归过程中，就可以把根结点下标index及所对应的值存储在map<int, char> level中，map是有序的会根据index从小到大自动排序，这样递归完成后level中的值就是层序遍历的顺序
- 模板

```
void level(int root, int start, int end, int index) {
    if(start>end) return;
    int i = start;
    while(i<end && in[i]!=post[root]) i++;
    ans[index]=post[root];
    pre(root-1-end+i, start, i-1, 2*index+1);
    pre(root-1, i+1, end, 2*index+2);
}
```

- ps: 后序中序转前序模板

  因为后序的最后一个总是根结点，令i在中序中找到该根结点，则i把中序分为两部分，左边是左子树，右边是右子树。因为是输出先序（根左右），所以先打印出当前根结点，然后打印左子树，再打印右子树。左子树在后序中的根结点为root – (end – i + 1)，即为当前根结点-右子树的个数。左子树在中序中的起始点start为start，末尾end点为i – 1.右子树的根结点为当前根结点的前一个结点root – 1，右子树的起始点start为i+1，末尾end点为end。

```
void pre(int root, int start, int end) {//root为后序中的根，start与end为中序中子树的范围
    if(start>end) return;
    int i= start;
    while(i<end && in[i]!=post[root]) i++;//在中序中找到根结点
    printf("%d ", post[root]);
    pre(root-1-end+i, start, i-1);
    pre(root-1, i+1, end);
}
```

三、实验环境

- 语言：C++
- 编译环境：Code Blocks 17.12 自带MinGW中的g++，需要在编译器设置中勾选C++14
- 输入1：

```
a+b*2/4-(b+3)*3
```

- 输出1：

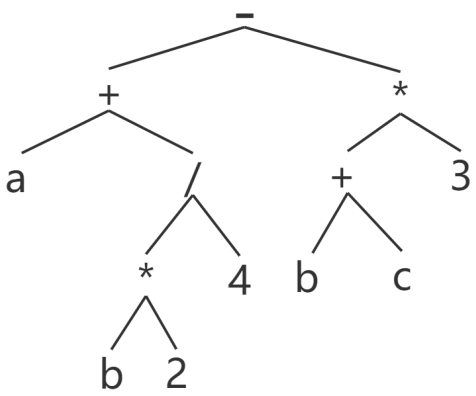
  后缀表达式

```
ab2*4/+bc+3*-
```

  层序输出

```
-+*a/+3*4bcb2
```

  语法树如下



- 输入2：

```
ab2*4/+bc+3*-
```

- 输出2：

```
Error!
```