对算术表达式的递归下降分析

17计二许红凯 320170941570

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

如果判断是正确，则输出表达式的后序遍历，否则输出错误

题目

设计一个程序，根据给定的上下文无关文法，能够判断。

> 1.基础文法：Expr -> Term Expr1
>
> Expr1 -> AddOp Term Expr1 | empty
>
> Term -> Factor Term1
>
> Term1 -> MulOp Factor Term1 | empty
>
> Factor -> id | number | (Expr)
>
> AddOp -> + | -
>
> MulOp -> * | /
>
> 2.语法分析方式采用递归子程序
>
> 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遵循惯例，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，输出表达式的后序遍历；输入错误时，输出error

代码思路

1. 根据给定的上下文无关文法，判断该表达式是否正确
2. 若正确，根据先序建立二叉树，在建树过程中，根据运算优先级决定括号的保留与去掉，再输出二叉树的后序遍历；若不正确，则输出错误

程序功能说明

对于任意算术表达式，输入后判断是否符合给定上下文无关文法，若符合则输出“正确”及其后序遍历结果；若不符合则输出“错误”

重点说明

1. 在先序建树过程中，对每一个括号（）需根据式子左右的符号判断是否要保留或去掉

```
/*

   local_r记录当前要转化的表达式生成二叉树的根节点操作符的位置
   flag记录是否当前搜索在括号里面
   m_m_p记录当前表达式中括号外面最右边的+、-位置
   a_s_p记录当前表达式中括号外面最右边的*、/位置
*/
int local_r = 0,flag = 0;
int m_m_p = 0,a_s_p = 0;
for(int i = ss; i < e; i++){
    if(afa[i] == '('){
        flag ++;
    }
    else if(afa[i] == ')'){
        flag --;
    }
    if(flag == 0){
        if(afa[i] == '*' || afa[i] =='/'){
            m_m_p=i;
        }
        else if(afa[i] =='+'||afa[i] =='-'){
            a_s_p = i;
        }
    }
}
if( (m_m_p == 0) && (a_s_p == 0) ){
     //如果式子整个有括号如(5-2*3+7)，即括号外面没有操作符，则去掉括号找二叉树
     afaToBtree(afa,ss + 1,e - 1);
}
else{
    //如果有+或者-，则根节点为最右边的+或-，否则是最右边的*或/
```

```
        if(a_s_p > 0){
            local_r = a_s_p;
        }
        else if(m_m_p > 0){
            local_r = m_m_p;
        }

  .....(其他代码)
  }
```

2. 对于给定文法中的Expr1()和Term1()，需注意判断Follow集防止死循环

```
void Expr1(){

    if(s[i] == '+' || s[i] == '-'){
        AddOp();
        Term();
        Expr1();
    }

    //Follow(Expr1)={)}
    else if( (s[i] != ')') && (s[i] != '#') ){

        error();
    }
}

void Term1(){

    if( (s[i] == '/') || (s[i] == '*') || (s[i] >= '0' && s[i] <= '9') || ((s[i] >=
        MulOp();
        Factor();
        Term1();
    }

    //Follow(Term1)={+,-,)}

    else if( ( (s[i] != '+') && (s[i] != '-') && (s[i] != ')') ) && (s[i] != '#') ){
        error();
    }

}
```

代码测试

1. 错误示例

```
Please input(以#结尾):a+b)*3#
Error!
```

2. 正确示例

```
Please input(以#结尾):a+b*2/4-(b+c)*3#
语句合法!
后序遍历为:
a
b
2
*
4
/
+
b
c
+
3
*
-
```