
语法分析

一、实验内容与要求

(1)实验目的

给出 PL/0 文法规范，要求编写 PL/0 语言的语法分析程序。

通过设计、编制、调试一个典型的语法分析程序，实现对词法分析程序所提供的单词序列进行语法检查和结构分析，进一步掌握常用的语法分析方法。

选择最有代表性的语法分析方法，如递归子程序法；选择对各种常见程序语言都具备的语法结构，如赋值语句，特别是表达式，作为分析对象。

(2)实验内容

已给 PL/0 语言文法，构造表达式部分的语法分析器。

分析对象〈算术表达式〉的 BNF 定义如下：

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

<项> ::= <因子>{<乘法运算符> <因子>}

<因子> ::= <标识符>|<无符号整数>| '(<表达式>')

<加法运算符> ::= +|-

<乘法运算符> ::= *|/

<关系运算符> ::= =|#|<|>|>=

(3)实验要求

将实验一“词法分析”的输出结果，作为表达式语法分析器的输入，进行语法解析，对于语法正确的表达式，报告“语法正确”；

对于语法错误的表达式，报告“语法错误”，指出错误原因。

把语法分析器设计成一个独立一遍的过程。

语法分析器的编写方法采用递归子程序法。

输入：

PL/0 表达式，用实验一的输出形式作为输入。例如：对于 PL/0 表达式，(a+15)

*b 用下列形式作为输入：

(lparen,()

(ident, a)
(plus, +)
(number, 15)
(rparen,))
(times, *)
(ident, b)

输出：

对于语法正确的表达式，报告“语法正确”；
对于语法错误的表达式，报告“语法错误”，指出错误原因。



二、实验分析与设计

递归下降分析法是一种自顶向下的分析方法，文法的每个非终结符对应一个递归过程（函数）。分析过程就是从文法开始符出发执行一组递归过程（函数），这样向下推导直到推出句子；或者说从根节点出发，自顶向下为输入串寻找一个最左匹配序列，建立一棵语法树。

在不含左递归和每个非终结符的所有候选终结首字符集都两两不相交条件下，我们就可能构造出一个不带回溯的自顶向下的分析程序，这个分析程序是由一组递归过程（或函数）组成的，每个过程（或函数）对应文法的一个非终结符。这样的分析程序称为递归下降分析器。

递归下降分析程序实现思想简单易懂。程序结构和语法产生式有直接的对应关系。因为每个过程表示一个非终结符号的处理，添加语义加工工作比较方便。

递归下降分析程序的实现思想是：识别程序由一组子程序组成。每个子程序对应于一个非终结符号。

每一个子程序的功能是：选择正确的右部，扫描完相应的字。在右部中有非终结符号时，调用该非终结符号对应的子程序来完成。

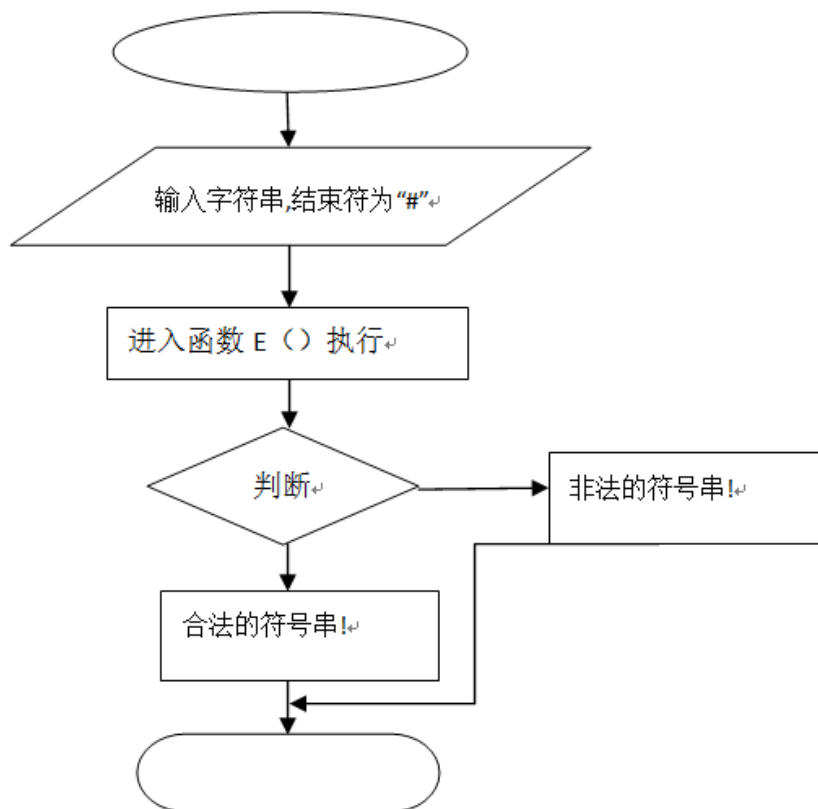
自上向下分析过程中，如果带回溯，则分析过程是穷举所有可能的推导，看是否能推导出待检查的符号串。分析速度慢。而无回溯的自上向下分析技术，当选择某非终结符的产生式时，可根据输入串的当前符号以及各产生式右部首符号而进行，效率高，且不易出错。

无回溯的自上向下分析技术可用的先决条件是：无左递归和无回溯。

无左递归：既没有直接左递归，也没有间接左递归。

无回溯：对于任一非终结符号 U 的产生式右部 $x_1|x_2|\dots|x_n$ ，其对应的字的首终结符号两两不相交。

如果一个文法不含回路（形如 $P \Rightarrow^+ P$ 的推导），也不含以 ϵ 为右部的产生式，那么可以通过执行消除文法左递归的算法消除文法的一切左递归（改写后的文法可能含有以 ϵ 为右部的产生式）。



本次实验使用词法分析器的分析结果，对输入的算术表达式进行语法判断：首先从文件中读入词法分析得到的结果，用以语法分析：

```

//读入词法分析得到的结果，将结果放入词汇表中
void loadFile()
{
    try {
        BufferedReader bufferedReader = new BufferedReader(new FileReader("Lex.txt"));

        String string = null;
        while ((string = bufferedReader.readLine()) != null) {
            String type = string.split(",")[0];
            String value = string.split(",")[1];
            LexTable.put(value, type);
            LexList.add(value);
        }
        bufferedReader.close();
    } catch (Exception e) {
        System.err.println("Lex.txt Not Found");
        System.exit(0);
        e.printStackTrace();
    }
}

```

对于读入的信息，用一个哈希表和一个 HashMap 存放单词名称和类型的映射关系：

```
HashMap<String, String> LexTable = new HashMap<String, String>();  
ArrayList<String> LexList = new ArrayList<String>();
```

对于已得到的词汇表整体，也应该设置全局变量：当前正在判断的终结符以及其类型：

```
String ch; //当前正判断的终结符  
String chType; //当前正判断的终结符类型  
int index = 0;  
boolean flag = false; //判断是否是完整表达式
```

采用递归下降分析法：

<表达式> ::= [+|-]<项>{<加法运算符> <项>}

表达式的识别：

```
Exp = ExpFlag Pro Exp1  
Exp1 = [+ | -] Pro Exp1 | 空  
ExpFlag = +|-|空
```

```
void expFlag()  
{ //表达式是否带符号  
  if(ch.equals("+") || ch.equals("-"))  
  {  
    Advance();  
    return;  
  }  
}  
  
void expression()  
{  
  expFlag();  
  pro();  
  expression1();  
}  
  
void expression1()  
{  
  if(addingOperator()) {  
    Advance();  
    pro();  
    expression1();  
  }  
}
```

<项> ::= <因子>{<乘法运算符> <因子>}

项的识别

Pro = Atom Pro1

Pro1 = [*|/] Atom Pro1 | 空

```
void pro()
{
    atom();
    pro1();
}
void pro1()
{
    if(multilingOperator())
    {
        Advance();
        atom();
        pro1();
    }
}
```

<因子> ::= <标识符>|<无符号整数>| '(<表达式>')'

因子的识别

Atom = ident | number | (Exp)

对于因子的识别，如果出现终结符，以为这要读入下一个单词：

并对读到的单词进行分析，得到单词名和单词类型：

```
//读取下一个终结符号
void Advance()
{
    if(index >= LexList.size())
    {
        ch = "输入串不符合规范";
        chType = "";
        return;
    }

    ch = LexList.get(index);
    chType = LexTable.get(ch);
    index++;
}
```

```

void atom()
{
    if(chType.equals("ident")) {
        Advance();
        return;
    }

    else if(chType.equals("number")) {
        Advance();
        return;
    }

    else if(chType.equals("Lparen"))
    {
        Advance();

        expression();

        if(chType.equals("Rparen"))
        {
            Advance();
            return;
        }
        else {
            error("缺少右括号  ) ");
        }
    }
    else {
        error(ch+": "+chType+" 无法识别");
    }
}

```

<加法运算符> ::= +|-

```

//加法运算符识别
boolean addingOperator()
{
    if(ch.equals("+") || ch.equals("-"))
    {
        return true;
    }

    else {
        return false;
    }
}

```

<乘法运算符> ::= */

```

//乘法运算符识别
boolean mutilingOperator()
{
    if(ch.equals("*") || ch.equals("/"))
    {
        return true;
    }
    else {
        return false;
    }
}

```

<关系运算符> ::= = | # | < | <= | > | >=

```

//关系运算符识别
boolean relationOperator()
{
    if(ch.equals("=") || ch.equals(":=") || ch.equals("<=")
        || ch.equals(">=") || ch.equals("<") || ch.equals(">") || ch.equals("<>"))
    {
        Advance();
        return true;
    }
    else {
        String mess = "关系运算符识别错误";
        error(mess);
        return false;
    }
}

```

同时要进行错误处理和成功提示:

```

void error(String mess)
{
    System.err.println("Error : -----语法错误-----");
    System.err.println("    "+mess);
    System.err.println();
    System.exit(0);
}

void success()
{
    System.out.println("语法正确");
    System.exit(0);
}

```

程序总流程如下:

```

void begin()
{
    loadFile();
    Advance();
    expression();
    success();
}

```

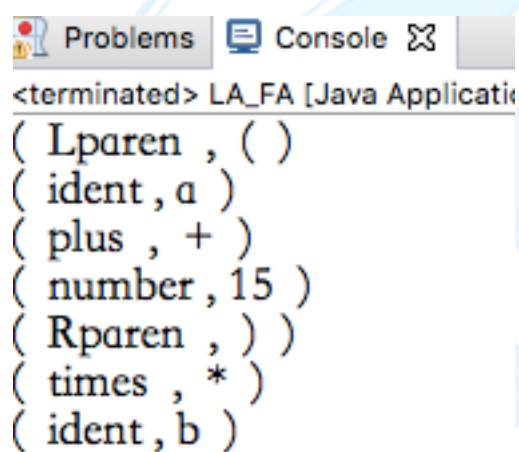

四，调试结果

输入一段程序：



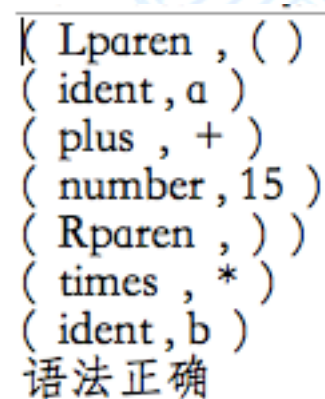
```
(a+15)*b
```

运行得到词法分析结果，可以看到结果正确无误：



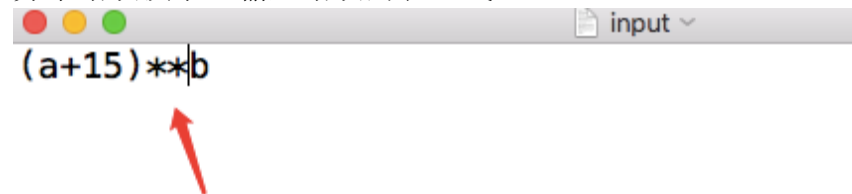
```
<terminated> LA_FA [Java Applicati  
( Lparen , ( )  
( ident , a )  
( plus , + )  
( number , 15 )  
( Rparen , ) )  
( times , * )  
( ident , b )
```

然后再运行语法分析：



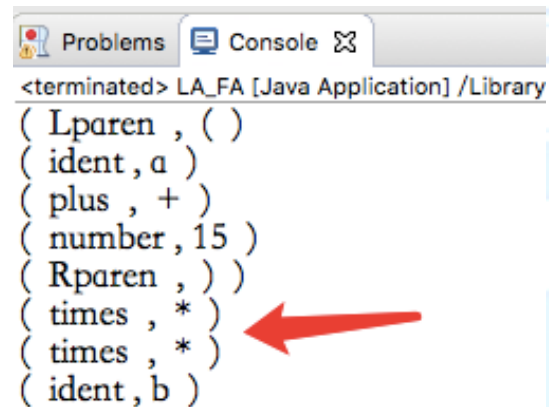
```
{ Lparen , ( )  
( ident , a )  
( plus , + )  
( number , 15 )  
( Rparen , ) )  
( times , * )  
( ident , b )  
语法正确
```

异常错误演示：输入错误的表达式：



(a+15)**b

词法分析器不会报错：



```
<terminated> LA_FA [Java Application] /Library
( Lparen , ( )
( ident , a )
( plus , + )
( number , 15 )
( Rparen , ) )
( times , * )
( times , * )
( ident , b )
```

但语法分析会报错：



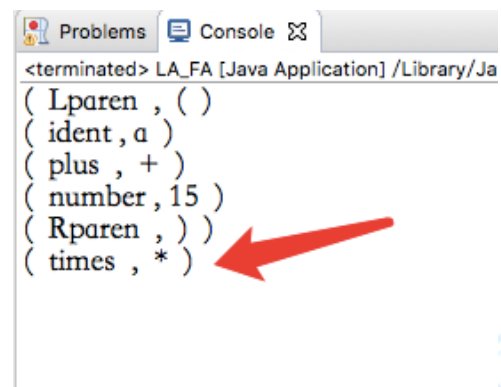
```
<terminated> LL1 [Java Application] /Library/Java/Javavirtua
( Lparen , ( )
( ident , a )
( plus , + )
( number , 15 )
( Rparen , ) )
( times , * )
( times , * )
( ident , b )
Error:      -----语法错误-----
          *: times 无法识别
```

异常错误演示：输入不完整的表达式：



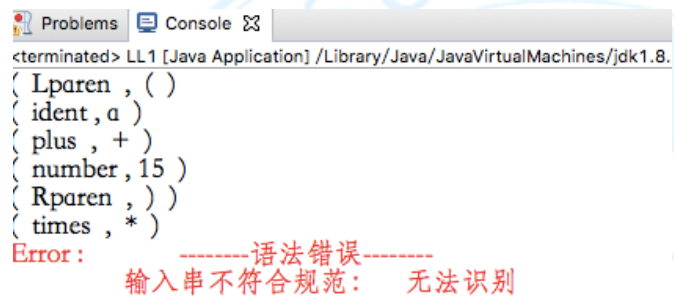
(a+15)*

词法分析器不会报错：



```
<terminated> LA_FA [Java Application] /Library/Ja
( Lparen , ( )
( ident , a )
( plus , + )
( number , 15 )
( Rparen , ) )
( times , * )
```

但语法分析会报错：



```
<terminated> LL1 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.
( Lparen , ( )
( ident , a )
( plus , + )
( number , 15 )
( Rparen , ) )
( times , * )
Error: -----语法错误-----
      输入串不符合规范: 无法识别
```