

## PL/0 语言文法的 BNF 表示:

〈程序〉 → 〈分程序〉。  
〈分程序〉 → [**<常量说明部分>**][**<变量说明部分>**][**<过程说明部分>**] 〈语句〉  
〈常量说明部分〉 → **CONST****<常量定义>**{ , **<常量定义>**};  
〈常量定义〉 → **<标识符>**=**<无符号整数>**  
〈无符号整数〉 → **<数字>**{**<数字>**}  
〈变量说明部分〉 → **VAR****<标识符>**{ , **<标识符>**};  
〈标识符〉 → **<字母>**{**<字母>**|**<数字>**}  
〈过程说明部分〉 → **<过程首部>****<分程序>**; {**<过程说明部分>**}  
〈过程首部〉 → **procedure****<标识符>**;  
〈语句〉 → **<赋值语句>**|**<条件语句>**|**<当型循环语句>**|**<过程调用语句>**|**<读语句>**|**<写语句>**|**<复合语句>**|**<空>**  
〈赋值语句〉 → **<标识符>**:**<表达式>**  
〈复合语句〉 → **begin****<语句>**{ ; **<语句>**}**<end>**  
〈条件〉 → **<表达式>****<关系运算符>****<表达式>**|**odd****<表达式>**  
〈表达式〉 → **[+|-]****<项>**{**<加减运算符>****<项>**}  
〈项〉 → **<因子>**{**<乘除运算符>****<因子>**}  
〈因子〉 → **<标识符>**|**<无符号整数>**|(**<表达式>**)  
〈加减运算符〉 → **+**|**-**  
〈乘除运算符〉 → **\***|**/**  
〈关系运算符〉 → **=**|**#**|**<**|**=**|**>**|**>=**  
〈条件语句〉 → **if****<条件>****then****<语句>**  
〈过程调用语句〉 → **call****<标识符>**  
〈当型循环语句〉 → **while****<条件>****do****<语句>**  
〈读语句〉 → **read**(**<标识符>**{ , **<标识符>**})  
〈写语句〉 → **write**(**<标识符>**{ , **<标识符>**})  
〈字母〉 → **a|b|c...x|y|z**  
〈数字〉 → **0|1|2...7|8|9**

### 一. PL/0 语言建立一个词法分程序 GETSYM (函数)

把关键字、算符、界符称为语言固有的单词，标识符、常量称为用户自定义的单词。为此设置三个全程量：SYM, ID, NUM。

SYM: 存放每个单词的类别，为内部编码的表示形式。

ID: 存放用户所定义的标识符的值，即标识符字符串的机内表示。

NUM: 存放用户定义的数。

GETSYM 要完成的任务:

1. 滤掉单词间的空格。
2. 识别关键字，用查关键字表的方法识别。当单词是关键字时，将对应的类别放在 SYM 中。如 IF 的类别为 IFSYM，THEN 的类别为 THENSYM。
3. 识别标识符，标识符的类别为 IDENT，IDENT 放在 SYM 中，标识符本身的值放在 ID 中。关键字或标识符的最大长度是 10。
4. 拼数，将数的类别 NUMBER 放在 SYM 中，数本身的值放在 NUM 中。

5. 拼由两个字符组成的运算符，如：>=、<=等等，识别后将类别存放在 SYM 中。
  6. 打印源程序，边读入字符边打印。
- 由于一个单词是由一个或多个字符组成的，所以在词法分析程序 GETSYM 中定义一个读字符过程 GETCH。

## 二. PL/O 语言建立一个语法分析程序 BLOCK (函数)

PL/O 编译程序采用一遍扫描的方法，所以语法分析和代码生成都有在 BLOCK 中完成。BLOCK 的工作分为两步：

### a) 说明部分的处理

说明部分的处理任务就是对每个过程（包括主程序，可以看成是一个主过程）的说明对象造名字表。填写所在层次（主程序是 0 层，在主程序中定义的过程是 1 层，随着嵌套的深度增加而层次数增大。PL/O 最多允许 3 层），标识符的属性和分配的相对地址等。标识符的属性不同则填写的信息不同。

所造的表放在全程量一维数组 TABLE 中，TX 为指针，数组元素为结构体类型数据。LEV 给出层次，DX 给出每层的局部量的相对地址，每说明完一个变量后 DX 加 1。

例如：一个过程的说明部分为：

```
const a=35,b=49;
var c,d,e;
procedure p;
    var g;
```

对它的常量、变量和过程说明处理后，TABLE 表中的信息如下：

TX0 →

NAME: a	KIND: CONSTANT	VAL: 35	
NAME: b	KIND: CONSTANT	VAL: 49	
NAME: c	KIND: VARIABLE	LEVEL: LEV	ADR: DX
NAME: d	KIND: VARIABLE	LEVEL: LEV	ADR: DX+1
NAME: e	KIND: VARIABLE	LEVEL: LEV	ADR: DX+2
NAME: p	KIND: PROCEDURE	LEVEL: LEV	ADR:

TX1 →

NAME: g	KIND: VARIABLE	LEVEL: LEV+1	ADR: DX
o	o	o	o
o	o	o	o
o	o	o	o

对于过程名的 ADR 域，是在过程体的目标代码生成后返填过程体的入口地址。

TABLE 表的索引 TX 和层次单元 LEV 都是以 BLOCK 的参数形式出现，在主程序调用 BLOCK 时实参的值为 0。每个过程的相对起始位置在 BLOCK 内置初值 DX=3。

### 2. 语句处理和代码生成

对语句逐句分析，语法正确则生目标代码，当遇到标识符的引用则去查 TABLE 表，看是否有过正确的定义，若有则从表中取出相关的信息，供代码生成用。PL/O 语言的代码生成是由过程 GEN 完成。GEN 过程有三个参数，分别代表目

标代码的功能码、层差、和位移量。生成的目标代码放在数组 CODE 中。CODE 是一维数组，数组元素是结构体类型数据。

PL/0 语言的目标指令是一种假想的栈式计算机的汇编语言，其格式如下：

f	l	a
---	---	---

其中 f 代表功能码，l 代表层次差，a 代表位移量。

目标指令有 8 条：

- ① LIT：将常数放到运栈顶，a 域为常数。
- ② LOD：将变量放到栈顶。a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值。
- ③ STO：将栈顶的内容送到某变量单元中。a, l 域的含义与 LOD 的相同。
- ④ CAL：调用过程的指令。a 为被调用过程的目标程序的入中地址，l 为层差。
- ⑤ INT：为被调用的过程（或主程序）在运行栈中开辟数据区。a 域为开辟的个数。
- ⑥ JMP：无条件转移指令，a 为转向地址。
- ⑦ JPC：条件转移指令，当栈顶的布尔值为非真时，转向 a 域的地址，否则顺序执行。
- ⑧ OPR：关系和算术运算。具体操作由 a 域给出。运算对象为栈顶和次顶的内容进行运算，结果存放在次顶。a 域为 0 时是退出数据区。

### 三. 建立一个解释执行目标程序的函数

编译结束后，记录源程序中标识符的 TABLE 表已退出内存，内存中只剩下用于存放目标程序的 CODE 数组和运行时的数据区 S。S 是由解释程序定义的一维整型数组。解释执行时的数据空间 S 为栈式计算机的存储空间。遵循后进先出的规则，对每个过程（包括主程序）当被调用时，才分配数据空间，退出过程时，则所分配的数据空间被释放。

为解释程序定义四个寄存器：

1. I：指令寄存器，存放当前正在解释的一条目标指令。
2. P：程序地址寄存器，指向下一条要执行的目标指令（相当于 CODE 数组的下标）。
3. T：栈顶寄存器，每个过程运行时要为它分配数据区（或称为数据段），该数据区分为两部分。

静态部分：包括变量存放区和三个单元。

动态部分：作为临时工作单元和累加器用。需要时临时分配，用完立即释放。栈顶寄存器 T 指出了当前栈中最新分配的单元（T 也是数组 S 的下标）。

4. B：基地址寄存器，指出每个过程被调用时，在数据区 S 中给出它分配的数据段起始地址，也称为基地址。每个过程被调用时，在栈顶分配三个联系单元。这三个单元的内容分别是：

SL：静态链，它是指向定义该过程的直接外过程运行时数据段的基地址。

DL：动态链，它是指向调用该过程前正在运行过程的数据段的基地址。

址。

RA: 返回地址, 记录调用该过程时目标程序的断点, 即当时的程序地址寄存器 P 的值。

具体的过程调用和结束, 对上述寄存器及三个联系单元的填写和恢复由下列目标指令完成。

1. INT 0 a

a: 为局部量个数加 3

2. OPR 0 0

恢复调用该过程前正在运行过程 (或主程序) 的数据段的基地址寄存器的值, 恢复栈顶寄存器 T 的值, 并将返回地址送到指令寄存器 P 中。

3. CAL 1 a

a 为被调用过程的目标程序的入口, 送入指令地址寄存器 P 中。

CAL 指令还完成填写静态链, 动态链, 返回地址, 给出被调用过程的基地址值, 送入基址寄存器 B 中。

例: 一个 PI/O 源程序及生成的目标代码:

```
const a=10;
var b,c;
procedure p;
begin
  c:=b+a
end;
2 int 0 3
3 lod 1 3
4 lit 0 10
5 opr 0 2
6 sto 1 4
7 opr 0 0
  begin
    read(b);
    while b#0 do
      begin
        call p;
        write(2*c);
        read(b)
      end
    end .
8 int 0 5
9 opr 0 16
10 sto 0 3
11 lod 0 3
12 lit 0 0
13 opr 0 9
14 jpc 0 24
```

15	cal	0	2
16	lit	0	2
17	lod	0	4
18	opr	0	4
19	opr	0	14
20	opr	0	15
21	opr	0	16
22	sto	0	3
23	jmp	0	11
24	opr	0	0