扩充PL/0文法解读

以下，对扩充PL/0文法逐条解读,列出其作用、限定条件和句子示例。最后还给出了我自己总结的一个需要注意和小心的问题。

# <程序> ::= <分程序>.

## 作用

**<程序>**是该扩充PL/0文法识别的的开始符号，描述的是整个源程序文件的格式。该扩充PL/0文法所描述的Pascal-S语言是一种分程序结构语言，由**<程序>**的定义可知，整个源程序就是一个大的**<分程序>**。

## 限定条件

1. 与内部的一般**<过程>**或**<函数>**不同的是，**<程序>**的结尾**end**后面跟的是一个句点而不是分号。
2. 句点之后不再有其他非空白字符。
3. 上述“句点”指的是半角的英文句点。

## 例句

begin

...

end.

{**end**之后有一个句点}

# <分程序> ::= [<常量说明部分>][<变量说明部分>]{[<过程说明部分>]|[<函数说明部分>]}<复合语句>

## 作用

定义**<分程序>**的语法格式。由定义知，**<分程序>**由可选的一个**<常量说明部分>**、可选的一个**<变量说明部分>**、可选的任意多个**<过程说明部分>**或**<函数说明部分>**、必须有的一个**<复合语句>**构成。

这里，每一个**<过程说明部分>**里面都有一个或多个过程说明，**<函数说明部分>**同样，不过这不影响，完全可以理解成一个，这是等价的。

## 限定条件

需要注意以下几点：

1. **<常量说明部分>**、**<变量说明部分>**、**<过程说明部分>**或**<函数说明部分>**、**<复合语句>**这四个部分的的顺序是不可以调换的。所有的**<过程说明部分>**和**<函数说明部分>**合在一起是一个整体，只需要保证这个整体位于**<变量说明部分>**和**<复合语句>**之间即可。
2. **<常量说明部分>**和**<变量说明部分>**都可以没有，若有，分别最多只能有一个。
3. 在所有**<过程说明部分>**和**<函数说明部分>**构成的这个整体的内部，**<过程说明部分>**和**<函数说明部分>**都可以没有，也都可以有多个，而且它们的顺序是任意的，多个**<过程说明部分>**和多个**<函数说明部分>**之间也可以互相交叉，比如可以几个**<过程说明部分>**，几个**<函数说明部分>**，然后再几个**<过程说明部分>**等等。
4. 这里的**<分程序>**的结尾就是**<复合语句>**的结尾，也就是一个**end**，其后没有任何标点符号。

## 例句

var ...

function ...

procedure ...

function ...

function ...

procedure ...

procedure ...

begin

...

end

或

const ...

begin

...

end

{与**<程序>**的不同是，**end**之后没有句点}

# <常量说明部分> ::= const<常量定义>{,<常量定义>};

## 作用

描述**<常量说明部分>**的格式。可见，**<常量说明部分>**以一个**const**开关，以分号结尾。中间是一个或多个**<常量定义>**，如果有多个，它们之间以逗号连接。

## 限定条件

1. 一个**<常量说明部分>**的开始就是以一个**const**开始、一个分号结束为标志的。中间一个或多个**<常量定义>**是属于同一个**<常量说明部分>**。
2. 多个**<常量定义>**相邻两个之间有且仅有一个逗号，不可以有冗余，也不可以少。若只有一个**<常量定义>**，则没有逗号。
3. 这里的逗号和分号都是半角的英文标点。

## 例句

const lim = 100;

或

const i = 5, ch = 'x';

# <常量定义> ::= <标识符>＝<常量>

## 作用

描述**<常量定义>**的格式。这个格式非常简单，就是一个**<标识符>**加一个等号再加一个**<常量>**，没有其他可能。

## 限定条件

1. 前后没有标点符号。对标点符号的处理在上一层**<常量说明部分>**里。
2. 等号是半角的英语等号。

## 例句

i = 5

或

ch = '5'

{没有其他标点，注意这里不是赋值符}

# <标识符> ::= <字母>{<字母>|<数字>}

## 作用

描述**<标识符>**的格式。可见**<标识符>**是定义是：以字母开头，后面可以没有，也可以有若干个字母或数字。

## 限定条件

1. 长度至少为1，不能为空。
2. 文法中没有规定标识符的最大长度。
3. 开头第一个字符必须是**<字母>**。
4. 仅由**<数字>**或**<字母>**组成。这与C语言不同，C语言中下划线在标识符中被当作字母处理，而此处**<字母>**就是单纯的字母。

## 例句

IJlIJiiJJkHIUHihHIIIbi

或

SHGSD

或

J8748ug002039393hgIHH9

{不可以有下划线}

# <字母> ::= a|b|c|d…x|y|z |A|B…|Z

## 作用

规定**<字母>**的范围，全部26个英文字母，包括小写与大写，只52个字符。

## 限定条件

1. **<字母>**指的是单个英文字母。
2. 区分大小写。

## 例句

a

或

A

{区分大小写}

# <数字> ::= 0|1|2|3…8|9

## 作用

规定**<数字>**的范围：0, 1, 3, 4, 5, 6, 7, 8, 9，以上十个阿拉伯数字。

## 限定范围

这里**<数字>**指的是单个数字字符。

## 例句

9

或

0

{单个数字字符}

# <常量> ::= [+|-] <无符号整数>|<字符>

## 作用

定义**<常量>**的语法格式，有两种。一种是整数常量，一种是字符常量。整数常量前面可以有一个正负号，也可以没有，后面跟一个**<无符号整数>**；字符常量由**<字符>**定义。

## 限定条件

1. 整数常量前面可以没有符号，有符号时最多只能有一个符号。
2. 与C语言不同，这里整数常量没有进制、长短的区分，开关没有0x或0X，结尾没有L等。
3. 与C语言不同，这里的字符串不是常量。

## 例句

-294938423

或

22

或

'B'

{字符串不是常量}

# <无符号整数> ::= <数字>{<数字>}

## 作用

定义**<无符号整数>**的格式。定义为：**<数字>**开头，后面什么都没有或者再跟若干个**<数字>**。

## 限定条件

1. 长度至少是1.
2. 文法中没有规定**<无符号整数>**位数的上限。
3. 可以有前导**0**，甚至可以有多个前导**0**.

## 例句

00000928283791735617351978356197356

或

87654

或

0

{可以有前导0}

# <字符> ::= '<字母>'|'<数字>'

## 作用

定义**<字符>**的格式：一对单引号，里面为单个**<字母>**或者单个**<数字>**。

## 限定条件

1. 这里的字符范围非常窄，与C语言不同，这里的字符单纯指字母或数字字符，不包含其他字符。
2. 这里的单引号是半角的英文标点。

## 例句

'a'

或

'4'

{只能是字母或数字，不能是其他字符}

# <变量说明部分> ::= var<变量说明>;{<变量说明>;}

## 作用

描述**<变量说明部分>**的格式。由定义可知，**<变量说明部分>**与**<常量说明部分>**不完全一样：这里每个**<变量说明>**之后都是一个分号，多个变量说明之间没有逗号。

## 限定条件

1. 这部分以**var**作为开始，没有明显的结束标志。
2. 至少有一个**<变量说明>**，可以有多个。
3. 每一个**<变量说明>**后面都跟着一个分号。
4. 这里的分号是半角的英文标点。

## 例句

var i: integer;

或

var i: integer;

a, aa: array[1024] of integer;

c, cc, ccc: char;

{没有明显的结束标志}

# <变量说明> ::= <标识符>{,<标识符>}:<类型>

## 作用

描述**<变量说明>**的格式：一个或多个以逗号分隔的**<标识符>**，后接一个冒号，后接一个**<类型>**。

## 限定条件

1. 至少有一个**<标识符>**，如果有多个，则两两之间用一个逗号隔开，不能多不能少。
2. 文法中对**<标识符>**个数的上限没有要求。
3. **<变量说明>**结尾没有标点，对标点的处理位于上层**<变量说明部分>**中。
4. 这里的逗号、冒号都是半角的英文标点。

## 例句

i: integer

或

a: array[1024] of integer

或

c, cc, ccc: char

{结尾没有标点}

# <类型> ::= <基本类型>|array'['<无符号整数>']'of<基本类型>

## 作用

定义了**<类型>**的范围：**<基本类型>**或数组类型。其中数组类型格式为**array**加一对方括号，方括号内是**<无符号整数>**，方括号外再加一个**of**，再加一个基本类型。

## 限定范围

1. 方括号内是**<无符号整数>**，不能带正负号，不可以是常量**<标识符>**。
2. 文法中数组大小可以为**0**。
3. 此文法中数组类型声明时的格式只此一种，只规定数组的大小，没有规定上下界。在语义处理时，我们规定数组下标从0开始。
4. 数组里的元素类型只能是**<基本类型>**，不允许定义多维数组。
5. 这里的方括号为半角的英文标点。

## 例句

integer

或

char

或

array[0] of char

{数组的声明只规定数组的大小}

# <基本类型> ::= integer|char

## 作用

定义**<基本类型>**的范围，只有整型**integer**和字符型**char**。这个规定很强，大大缩小了语言的数据类型范围。

## 限定条件

无。

## 例句

integer

或

char

{仅此两种}

# <过程说明部分> ::= <过程首部><分程序>{; <过程首部><分程序>};

## 作用

描述**<过程说明部分>**的格式。仔细观察发现，事实上一个**<过程说明部分>**由一条或多条过程说明组成，而每一条过程说明都有如下格式：一个**<过程首部>**、一个**<分程序>**、一个分号。每一条过程说明都以分号结束，而整个**<过程说明部分>**没有明显的开始或结束标志。

## 限定条件

1. 由一条或多条过程说明组成，处理完一个过程说明之后，没有明显的标志是不是结束了这个**<过程说明部分>**的处理。
2. 这里的分号是半角的英文标点。

## 例句

procedure p1(arg1: integer);

begin

...

end;

procedure p2(arg1: char; var arg2: integer);

begin

...

end;

procedure p3;

begin

...

end;

{每一条过程说明都以分号结尾}

# <过程首部> ::= procedure<标识符>[<形式参数表>];

## 作用

描述**<过程首部>**的格式：以**procedure**开头，后接一个**<标识符>**，再接一个可选的**<形式参数表>**，最后一个分号。

## 限定条件

1. 每个**<过程首部>**都以一个**procedure**开始，以一个分号结束。
2. **<形式参数表>**可有可无。
3. 这里的分号是半角的英文标点。

## 例句

procedure p1(a1, a2: char; a3: integer);

或

procedure p2;

{形式参数表可有可无}

# <形式参数表> ::= '('<形式参数段>{;<形式参数段>}')'

## 作用

描述**<形式参数表>**的格式：开头结尾是一对圆括号，中间是一个或多个**<形式参数段>**。如果有多个，则两两之间用一个分号隔开，分号不能多也不能少。

## 限定条件

1. 以左圆括号开头，以右圆括号结尾。
2. 这里的圆括号、分号都是半角的英文标点。

## 例句

(a1: char; a2, a3: integer; a4, a5: char; a6: integer)

或

(a1, a2, a3: char)

# <形式参数段> ::= [var]<标识符>{,<标识符>}:<基本类型>

## 作用

描述**<形式参数段>**的格式：前面一个可选的**var**，然后一个或多个两两以逗号分隔的**<标识符>**，然后一个冒号，然后一个**<基本类型>**。其中标识符之前的逗号不能多也不能少。

## 限定条件

1. 前面的**var**可选。
2. 一个**<形式参数段>**中可以有一个或多个标识符，以逗号隔开。
3. 由定义知，形式参数只能为**<基本类型>**，不能为数组类型。
4. 这里的逗号为半角的英文标点。

## 例句

var v1: integer

或

var v1, v2: char

或

v1, v2: integer

{前面的**var**可选}

# <函数说明部分> ::= <函数首部><分程序>{;<函数首部><分程序>};

## 作用

描述**<函数说明部分>**的格式。一个**<函数说明部分>**由一个或多个函数说明组成，而每个函数说明格式如下：一个**<函数首部>**加一个**<分程序>**再加一个分号。每个函数说明都以分号结尾，而整个**<函数说明部分>**没有明显的结尾标志。

## 限定条件

1. 一个**<函数说明部分>**可以有一个或多个函数说明。多个函数说明之间没有分隔符。
2. 每个函数说明以分号结尾，而整个**<函数说明部分>**则没有明显的结尾标志。
3. 这里的分号是半角的英文标点。

## 例句

function f1: char;

begin

...

end;

function f2(...): integer;

begin

...

end;

# <函数首部> ::= function<标识符>[<形式参数表>]:<基本类型>;

## 作用

描述**<函数首部>**的格式。每个**<函数首部>**都以一个**function**开始，以分号结束。是一个**<标识符>**，一个可选的**<形式参数表>**，一个冒号，一个**<基本类型>**。

从这里可以看出function与procedure的区别：procedure没有返回值，而function是有返回值的，所以在**<函数首部>**中要说明返回值的类型。

## 限定条件

1. **<形式参数表>**可选。
2. 返回值类型只能是**<基本类型>**，不能是数组。
3. 这里的冒号、分号都是半角的英文标点。

## 例句

function f1: integer;

或

function f2(a1: char; var a2: integer): char;

{**<形式参数表>**可选，只能返回**<基本类型>**}

# <复合语句> ::= begin<语句>{;<语句>}end

## 作用

描述**<复合语句>**的格式，**<复合语句>**以**begin**开头，以**end**结尾，中间有一条或多条**<语句>**，如果是一条，则**<语句>**的结尾没有分号；如果是多条，则每条**<语句>**的结尾有一个分号，但是最后一条**<语句>**结尾没有分号。

## 限定条件

1. 与C语言不同的是，这里的**<语句>**结尾是没有分号的，分号只是为了分隔多条**<语句>**，所以无论如何，最后一条**<语句>**结尾都是没有分号的。
2. 空语句的表示方法与C语言不同。C语言中空语句是单独一个分号，而这里的空语句是什么都没有。但是这不影响，因为有空语句的存在，**<复合语句>**中，可以认为最后一条**<语句>**后面有一条空语句，这样真正的最后一条语句就可以有分号。同样，在**<复合语句>**中，空语句也可以用一个分号表示。综上，**<复合语句>**的最后一条**<语句>**结尾的分号是可有可无的。
3. 这里的分号是半角的英文标点。

## 例句

begin

i := j

end

或

begin

i := j;

j := k

end

{最后一条**<语句>**结尾分号可有可无}

# <语句> ::= <赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<for循环语句>|<空>

## 作用

规定**<语句>**的种类。**<空>**代表什么都没有。

## 限定条件

注意这里的**<语句>**与C语言不同，这里**<语句>**的结尾是不带分号的。对分号的处理在上层**<复合>**中。

## 例句

begin

end

或

j := 500

{结尾不带分号}

# <赋值语句> ::= <标识符>:=<表达式>|<函数标识符>:=<表达式>|<标识符>'['<表达式>']':=<表达式>

## 作用

规定**<赋值语句>**的格式。有以下几种：

1. 普通赋值：一个**<标识符>**，一个赋值运算符**:=**，一个**<表达式>**。
2. 函数返回值赋值：一个**<函数标识符>**，也就是函数名，一个赋值运算符**:=**，一个**<表达式>**。在文法上与普通赋值相同，但是在语义处理上不一样，所以单列出来。
3. 数组元素赋值：一个**<标识符>**，也就是数组名，一对方括号，方括号内部为一个**<表达式>**，方括号外面再跟一个**<表达式>**。

## 限定条件

1. 三种**<赋值语句>**的第一个符号看似一样，都是**<标识符>**，但是**<标识符>**的种类不同，也就决定了应该走哪一个分支。
2. 赋值运算符是一个双字节运算符，由一个冒号紧挨着一个等号构成，冒号等号之前不可以有任何字符，包括空白符也不能有。
3. 根据后文对**<表达式>**的定义，形式上不可以是**<字符>**，如果想表达一个字符，要用变量表示。
4. 这里的方括号、冒号、等号，都是半角的英文标点。

## 例句

variable1 := 5050

或

function1 := c1

或

array1[50\*3] := 5050 + 125

{冒号与等号必须是坚挨着的}

# <表达式> ::= [+|-]<项>{<加法运算符><项>}

## 作用

定义**<表达式>**的语法格式。第一个**<项>**之前可以有一个可选的正负号，这个正负号的作用范围只是第一**<项>**本身，而不是整个表达式。然后是一个**<项>**或者之个两两之间用一个**<加法运算符>**连接的**<项>**。与正常的书写习惯一致。这里的**<加法运算符>**不单纯是一个加号，后面有定义。

## 限定条件

1. 不管是一个**<项>**还是用**<加法运算符>**连接的多个**<项>**，**<表达式>**都没有明显的结束标志。
2. 这里的正负号是半角的英语标点。

## 例句

+i - j\*k

# <项> ::= <因子>{<乘法运算符><因子>}

## 作用

描述**<项>**的格式。一个**<项>**由一个或多个由**<乘法运算符>**连接起来的**<因子>**构成。这里的**<乘法运算符>**不单纯是乘号，后面有说明。

## 限定条件

1. 如果只有一个**<因子>**，就没有**<乘法运算符>**。
2. 如果有多个**<因子>**，那么每两个**<因子>**之间有且仅有一个**<乘法运算符>**，不能多也不能少。

## 例句

a[n] \* x / 50

# <因子> ::= <标识符>|<标识符>'['<表达式>']'|<无符号整数>|'('<表达式>')'|<函数调用语句>

## 作用

定义**<因子>**的格式。**<因子>**有以下几种：

1. **<标识符>**，表示一个变量名或者常量名。
2. **<标识符>**加一对方括号，方括号内为**<表达式>**，意为取数组元素。
3. **<无符号整数>**，表示整数常量，这里不带正负号，但是在表达式的定义中，第一个**<项>**之前是可以有正负号的，而一个**<项>**可以是一个**<因子>**，所以此处如果想要表达带正负号的**<因子>**在整个方法中是完全可以的。
4. 一对圆括号，括号内是一个**<表达式>**，由于表达式中是有加减的，为了保证优先级，把有加减号的**<表达式>**外面加一对括号，这样这个**<表达式>**就可以作为一个整体参与乘除法运算。与正常的书写习惯一致。**<表达式>**可以是**<项>**，**<项>**又可以是**<因子>**，故这个方法允许**<表达式>**中有冗余的括号。
5. **<函数调用语句>**，这里不可以是**<过程调用语句>**，前文说过，函数是有返回值的，而过程没有。**<因子>**一定是有值的。

## 限定条件

1. **<因子>**中不能出现直接用单引号表示的**<字符>**，如果想要表达**<字符>**，必须把它定义为**const**或者**var**，在**<因子>**中用相应的**<标识符>**来表示。
2. 由**<因子>**的定义可知，**<表达式>**前面的正负号默认是只作为于第一**<项>**的，如果想到作用于整个**<表达式>**，要将后面的部分用圆括号括起来，当作一个**<因子>**来处理。
3. 由**<表达式>**、**<项>**、**<因子>**的递归定义可知，**<表达式>**可允许有冗余的括号。
4. **<函数调用语句>**不可以是**<过程调用语句>**，语法上一样，但是语义上完全不同。
5. 这里的圆括号是半角的英文标点。

## 例句

(-c1 + i\*j - 5050)

或

func1(x1, x2, x3)

或

(((((a + b)))))

# <函数调用语句> ::= <标识符>[<实在参数表>]

## 作用

描述**<函数调用语句>**的格式。要么是只有一个**<标识符>**，即函数名，要么是一个**<标识符>**加一个**<实在参数表>**。当没有实在参数表时，只有一个标识符。

## 限定条件

与C语言不一样，当函数没有参数时，C语言在函数后要有一个空的括号，而这里只有一个函数名，只从形式上看，无法确定这是一个函数还是一个变量。

## 例句

func

或

func(arg1)

{没有参数时，没有括号}

# <实在参数表> ::= '('<实在参数>{,<实在参数>}')'

## 作用

定义**<实在参数表>**的格式：以左圆括号开头，以右圆括号结尾，中间是一个**<实在参数>**，或者多个用一个逗号分隔的**<实在参数>**。

## 限定条件

1. 圆括号必须有，里面的逗号不能多也不能少，相当于除第一个**<实在参数>**外，后面每个**<实在参数>**前都有且仅有一个逗号。
2. 这里的逗号是半角的英文标点。

## 例句

(a1)

或

(a1, a2, a3)

{参数表是带圆括号的}

# <实在参数> ::= <表达式>

## 作用

定义**<实在参数>**的形式，同**<表达式>**。

## 限定条件

无。

## 例句

+(i - j\*k) – function1(a1, a2, a3)

{同**<表达式>**}

# <乘法运算符> ::= \*|/

## 作用

**<乘法运算符>**和**<加法运算符>**分开定义，是为了限定它们的优先级。同类的运算符优先级是相同的，从左往后计算即可；而**<乘法运算符>**的优先级高于**<加法运算符>**的优先级。前面对**<项>**和**<因子>**的分开定义，就是为了在文法上对优先级加以区分。

## 限定条件

这里的乘号和除号必须是半角的英文标点，与小学算术的书写习惯不同，**\***代表乘号，**/** 代表除号。

## 例句

\*

或

/

{仅此两种}

# <加法运算符> ::= +|-

## 作用

规定**<加法运算符>**的范围：加号和减号。

## 限定条件

这里的加号和减号都是半角的英文标点。

## 例句

+

或

-

{仅此两种}

# <函数标识符> ::= <标识符>

## 作用

只是为了文法的完备，引入**<函数标识符>**的概念只是为了在变量名和函数名之间在语义上加以区分，使得它们走向不同的分支。

## 限定条件

无

## 例句

func

或

sdf335KWR928iwj

{只是一个**<标识符>**，没有特别之处}

# <条件语句> ::= if<条件>then<语句>|if<条件>then<语句>else<语句>

## 作用

规定**<条件语句>**的格式。有两种形式，一种带**else**，一种不带**else**。与C语言稍有区别，这里**if**后面的**<条件>**是不带括号的，直到遇到**then**，才算**<条件>**结束。后面跟的是**<语句>**，前文说过，**<语句>**是不带分号结尾的，注意。

## 限定条件

1. **<条件>**不带括号。
2. **<条件>**后面是**then**。
3. **<语句>**结尾不带分号。

## 例句

if i > 0 then

begin

num := num + 1;

i := i – 1

end

else

write(num)

{结尾不带分号}

# <条件> ::= <表达式><关系运算符><表达式>

## 作用

规定**<条件>**格式。这里**<条件>**的格式非常严格，没有且、或、非等逻辑连接词，必须是一个用**<关系运算符>**连接起来的两个**<表达式>**。

## 限定条件

一个**<条件>**只能有且仅有一个**<关系运算符>**。

## 例句

y >= 0

{只能提一个条件，没有且、或、非}

# <关系运算符> ::= <|<=|>|>=|=|<>

## 作用

规定**<关系运算符>**的格式。这里的不等号也C语言不同，这里用小于大于表示不等于，用一个等号表示等于（因为赋值符不是一个等号）。

## 限定条件

1. **<=**、**>=**、**<>**都是双字节运算符，中间不能插入包括空白符在内的任何字符。
2. 这里的小于号、大于号、等号都是半角的英文标点。

## 例句

=

或

<>

{双字节运算符中间没有空白符}

# <当循环语句> ::= do<语句>while<条件>

## 作用

描述**do-while**语句的格式，注意没有花括号、圆括号等，语句结尾没有分号，**<条件>**不带括号，结尾没有明显标志。

## 限定条件

1. **do**和**while**之间的部分为循环体，**while**之后的部分为循环条件。
2. **<条件>**不带括号，没有明显的结束标志。

## 例句

do

begin

j := j \* i;

i := i + 1

end

while i <= 123 \* (5 + j)

{没有明显结尾标志}

# <过程调用语句> ::= <标识符>[<实在参数表>]

## 作用

规定**<过程调用语句>**的格式。与**<函数调用语句>**形式上相同。

## 限定条件

没有参数时，只有一个**<标识符>**，没有括号。

## 例句

p1

或

p2(a1, i \* 5 + j)

# <读语句> ::= read'('<标识符>{,<标识符>}')'

## 作用

规定**<读语句>**的格式。相当于调用一个名为**read**的过程，但是对参数仅限制为变量名，即使是数组元素也不行！

## 限制条件

1. 参数只能是变量名。
2. 这里的圆括号是半角的英文标点。

## 例句

read(n)

或

read(i, j, k)

{参数只能是变量名}

# <写语句> ::= write'('<字符串>,<表达式>')'|write'('<字符串>')'|write'('<表达式>')'

## 作用

规定**<写语句>**的格式，这里**<写语句>**的表达能力非常有限，只能接受三种类型的参数：一个**<字符串>**加一个**<表达式>**、一个**<字符串>**、一个**<表达式>**。

## 限定条件

1. 只有这三种格式，不可以给多个参数。
2. 这里的圆括号和逗号都是半角的英文标点。

## 例句

write("My age is ", 18)

或

write("My name is Luice")

或

write(7+8)

{只有以上三种形式}

# <字符串> ::= "{十进制编码为32,33,35-126的ASCII字符}"

## 作用

定义了**<字符串>**的表示形式和其中有效的字符。有效的字符是空格，以及ASCII码表中空格之后一直到**~**的部分，去掉双引号。

## 限定条件

1. 这里的**<字符串>**不需要作有关转义字符的转换，写出来是什么就是什么。所以一定是键盘上能找到的字符，包括空格，不包括回车和双引号。
2. **<字符串>**开头和结尾的双引号是半角的英文标点。

## 例句

"~!@#$%^&\*()\_+QWERTYUIOP{}|ASDFGHJKL:ZXCVBNM<>?`1234567890-=qwertyuiop[]\asdfghjkl;'zxcvbnm,./ "

{以上是这里支持的所有字符}

# <for循环语句> ::= for<标识符>:=<表达式>(downto|to)<表达式>do<语句>

## 作用

规定了**<for循环语句>**的格式。从这个格式中可知这里的for语句的表达能力非常受限，循环条件只是规定了循环次数。格式如下：一个**for**，一个**<标识符>**即循环计数变量名，一个赋值运算符，一个**<表达式>**代表循环计数变量的初值，一个**downto**或者**to**，一个**<表达式>**代表循环计数变量的终值，一个**do**，一个**<语句>**。循环次数为从第一个**<表达式>**到第二个**<表达式>**（含），逐次加一或减一所经历的次数。其中**downto**代表减一，**to**代表加一。同上，**<语句>**是没有分号结尾的。

## 限定条件

1. 语义上，**<for循环语句>**只能表达循环计数变量逐次加一或减一的过程。
2. 如果循环计数变量的初始值在循环有效范围之外，则不招待循环。

## 例句

for i := j-1 to j\*k do

s := s + i

{没有分号结尾}

# 总结

对文法的分析过程完毕。最有感触的是有关**<语句>**结尾有没有分号的问题。无论在测试程序的构造中，还是在编译程序作语法分析的过程中，分号都将是一个值得小心又小心的问题。

~~与之前使用C语言等每条语句都有分号结尾的语言时的习惯明显不同。比如一个完整的程序可以写成下面这样：~~

~~var i, j, k: integer;~~

~~begin~~

~~j := 0; {有分号}~~

~~for i := 100 downto 0 do~~

~~j := j + i; {有分号}~~

~~write(j); {有分号}~~

~~for k := 0 to j do~~

~~begin~~

~~i = i – 1; {有分号}~~

~~j = j – 1 {没有分号}~~

~~end {没有分号}~~

~~end.~~

~~或者这样：~~

~~var i, j, k: integer;~~

~~begin~~

~~j := 0; {有分号}~~

~~for i := 100 downto 0 do~~

~~j := j + i; {有分号}~~

~~write(j); {有分号}~~

~~for k := 0 to j do~~

~~begin~~

~~i = i – 1; {有分号}~~

~~j = j – 1 {没有分号}~~

~~end; {有分号}~~

~~write(k) {没有分号}~~

~~end.~~

~~只在最后加了一条输出语句，最后两行都要注意。每一个~~**~~<复合语句>~~**~~内部都要加这种小心。~~

以上划掉的部分是我之前的直观理解，这里加以更正：

1. 当作为一条**<语句>**解释时，其结尾是没有分号的。
2. 在**<复合语句>**内部，最后一条**<语句>**的结尾分号可有可无，也可以用单独一个分号表示一条空语句。