1.

<程序>                ::=  <分程序>.

**范例：**

**const a=4; .**

**分析：**

**程序的主题是分程序， “.”为结束符号， 告诉编译器需要编译的程序到此为止；**

2.

<分程序>               ::=  [<变量说明部分>][<常量说明部分>][<过程说明部分>]<复合语句>  
<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>  
<标识符>                ::=  <字母>{<字母>|<数字>}  
<变量说明部分>   ::=  var<标识符>{, <标识符>};  
<过程说明部分>   ::=  <过程首部><分程序>{;<过程首部><分程序>};  
<过程首部>            ::=  procedure<标识符>;

<复合语句>           ::=  begin<语句>{;<语句>}end

**范例：**

**var x,y,z :integer; //变量说明部分**

**const a=6,b=5,c=3; //常量说明部分**

**procedure lab1; //过程声明部分**

**lab1 分程序（略）**

**begin**

**x:=a;**

**for y:=1 to 6**

**do**

**x:= x+1;**

**end**

**分析1：**

**根据分程序的文法可知，分程序各个组成成分的声明顺序已经限定好了，不能随便更改他的声明顺序，如：”procedure lab1;const a=6,b=5,c=3;var x,y,z :integer;”,“var x; var y;”，这些声明顺序不合法上述的文法；**

**分析2：**

**每一个分程序按顺序由变量说明部分，常量说明部分，过程说明部分，以及复合语句组成。其中变量，常量，过程说明，复合语句之间有严格的先后顺序，不能打乱。变量说明，常量说明，过程说明对于一个分程序来说均为可有可无的部分，只有最后的复合语句是必须存在的；**

3.

<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>|<空>   
<赋值语句>           ::=  <标识符> := <表达式>  
<条件>                    ::=  <表达式><关系运算符><表达式>|odd<表达式>  
<关系运算符>       ::=  =|<>|<|<=|>|>=  
<条件语句>           ::=  if<条件>then<语句>  
<当循环语句>       ::=  while<条件>do<语句>   
<过程调用语句>  ::=  call<标识符>   
<读语句>               ::=  read'('<标识符>{, <标识符>}')'   
<写语句>               ::=  write'('<表达式>{, <表达式>}')'

**范例：**

**var x,y,m :integer;**

**x := 5; //赋值语句**

**y := 0; //赋值语句**

**if x > 1 x := x – 1; //条件语句**

**if odd x -1 y := y + 1; //条件语句用odd**

**begin /\*复合语**

**while x > 0 /\*当循环语句**

**do write (x - 1); 写语句\*/**

**end; \*/**

**call lab1; //过程调用语句**

**read (m); //读语句**

**分析:**

**此部分主要说明了各种类型语句的结构组成，就是说明了能够接受的语句语法格式；某些语句会有多种形式，在进行分析时应考虑全面；在这部分我们文法有各种各样的语句语法格式；**

4.

<表达式>              ::=  [+|-]<项>{<加法运算符><项>}    //[+|-]只作用于第一个<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>       ::=  \*|/

**范例：**

**var x,y,z :integer;**

**x := 2**

**y := 3**

**z := x + y; /\* 表达式的多种形式**

**z := x – y;**

**z := x \* y;**

**z := x / y;**

**z := +2 + x;**

**z := -2 + x;**

**z := (x + 1) – y; \*/**

**分析：**

**此处表述了表达式由项组成的具体方式，包含多种情况，值得注意；第一个项前面的+、-是表示如果第一个项需要是负数或正数，在数字前加上符号；后面的+、-基本上为运算符的+、-，而在这文法也需要注意乘法运算和括号“（”“）”的优先级；**

**5.**

<无符号整数>       ::=  <非零数字>{<数字>}|0  
**范例：**

**var x,y :integer;**

**x := 1204;**

**y:= 0; //1204, 0 为无符号整数**

**分析：**

**此处说明了无符号整数的组成方法，即数字的重复，但需要注意以0开头的数字他无法接受，如：011，因为0为开头是无符号整数文法的终结符；**

6.

<字母>                    ::=  a|b|c|d...|x|y|z|A|B...|Z   
<数字>                    ::=  0|<非零数字>  
<非零数字>           ::=  1|2|3...|8|9

**分析：**

**此处说明了文法的所有终结符，而说明了字母，数字，非零数字的组成方式；**