练习 2.1.1：考虑下面的上下文无关文法：

S=>+SS|*SS|a

1) 试说明如何使用文法生成串 *a + a a

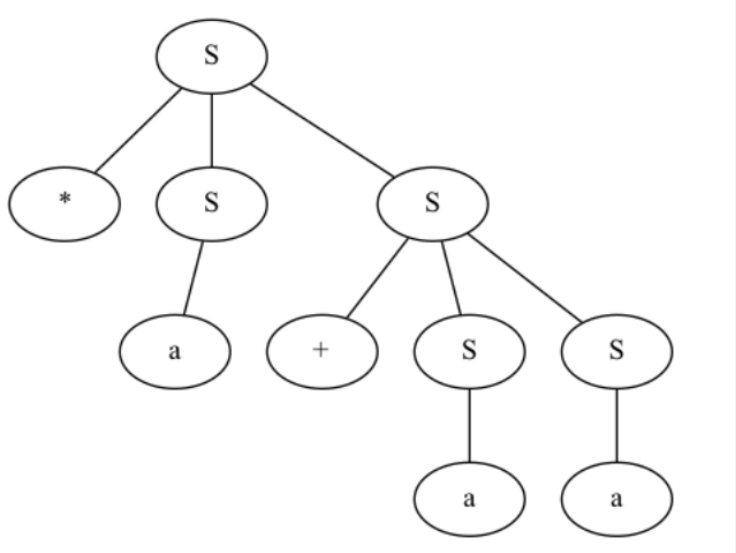
2) 试为这个串构造一颗语法分析树

3) 该文法生成的语言是什么？为什么？

1）答案不唯一

S=>*SS=>*aS=>*a+SS=>*a+aS=>*a+aa

2）语法分析树如下图所示：



3）由字符 a 与运算符+、*构成的前缀表达式。

（证明言之有理即可，用数学归纳法就行。）

练习 2.1.2：考虑文法：

num =>101|1111|num 0|num num

1) 证明：用该文法生成的所有二进制串的值都能被 5 整除（提示：对语法分析树的结点数目，即推导步数，使用数学归纳法）

2) 上面的文法是否能够生成所有能被 5 整除的二进制串？

1）（其他证明言之有理也可。）

使用数学归纳法，对该文法最终获得句子进行的推导步数（语法分析树节点）m 进行归纳。

当 m=1 时，仅进行 1 次推导，仅有 n=>101 和 n=>1111 两种可能，

两个句子分别表示数值 5 和 15，均能被 5 整除。

设 m <= k (k >= 1)$时，得到句子的数值都能被 5 整除，

对于 m = k + 1：第 1 步推导必然为 n => $n_1$0 或 n=>n_1n_2（下标仅为区分 n 的多次出现）。

对于 n=>n_1 0，设 n_1 由 k 步推导得到终结符号串 x，则 n 最终推导得到终结符号串 x0，

由归纳假设知 x 可被 5 整除，故 x0 必能被 5 整除；

对于 n=> n_1n_2 : n_1 和 n_2 分别可经过不超过 k 步推导获得终结符号串 x、y。

因此 n 经过 k+1 步推导获得终结符号串 xy；

由归纳假设知 x、y 均可被 5 整除；故 xy 可被 5 整除。

综上，该文法获得的句子表示的数值都可被 5 整除。

2）不能

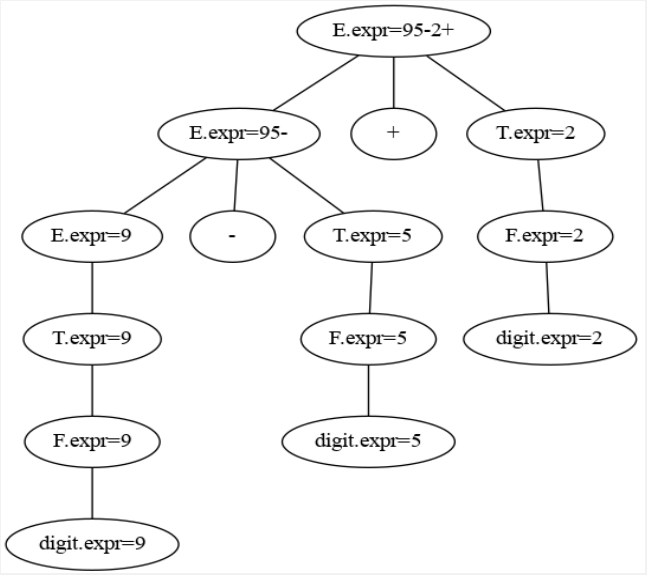
可以举反例说明，例如该文法无法表示 0（0）、25（11001）、35（100011）等。

练习 2.1.3：构建一个语法制导翻译方案，该方案把算术表达式从中缀表示方式翻译为运算符在运算分量之后的后缀表示方式。例如，xy−是表达式 x−y 的后缀表示。给出输入 9-5+2 和 9-5*2 的注释分析树

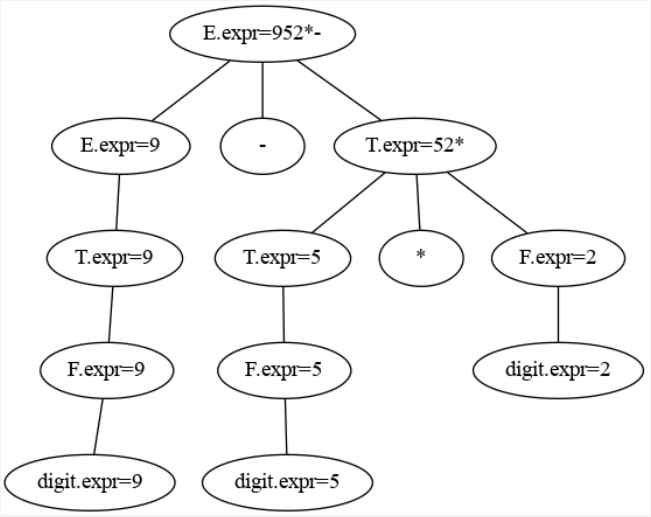
1) 语法制导翻译方案

```
E -> E1 + T { print('+') }
     | E1 - T { print('-') }
     | T
T -> T1 * F { print('*') }
     | T1 / F { print('/') }
     | F
F -> digit { print(digit) }
     | ( E )
```

2) 9-5+2 的注释语法分析树



3) 9-5*2 的注释语法分析树



练习 2.1.4：从 C99 标准开始，C 语言的 for 语句中初始化子句可以是声明，例如：

```
for ( int i = 0; i < n; i++ )
```

第一在 C++ 中允许类似的语法，但区别是：C++ 中 *初始化语句* 的作用域与 *循环语句* 的作用域一致，而在 C 中 *循环语句* 的作用域嵌套于 *初始化语句* 的作用域中。例如

```
for (int i = 0; ; ) { long i = 1;     // 在 C 中合法，在 C++ 中非法}
```

参考龙书图 2-37 中的实现，试讨论在两种 for 语句实现中分别应该如何实现符号表来正确区分变量和避免重复声明。

在 C 中，在每个 for 循环开始处创建一个符号表，在循环结束处删除，将初始化子句中声明的变量插入这个符号表中。如果循环语句是复合语句则创建单独的符号表并在循环语句结束处删除。

在 C++ 中，在每个 for 循环开始处创建一个符号表，在循环结束处删除，将初始化子句中声明的变量插入这个符号表中。如果循环语句是复合语句则不额外创建符号表。

避免重复声明的方法：向符号表中插入表项前检查在当前表中是否有重复项。

练习 2.1.5：给出一个文法，生成所有被 3 整除的非负整数的二进制串并给出证明

分开考虑除以 3 余 0、1、2 的串即可

S=>S0|A1 | 0

A=>S1|B0|1

B=>A0|B1