4.12 为文法

`S→(L) | a`

`L→L,S | S`

(b) 分别写出相应的语法制导定义、翻译方案以及预测翻译器，它打印出每个`a`在句子中是第几个字符。例如，当句子是`(a,(a,(a,a),(a)))`时，打印的结果是`2 5 8 10 14`。

为非终结符设置继承属性 l 和综合属性 r , 其中，l 属性表示非终极符可以推导出的第一个字符在句子中是第几个字符，r 属性则表示非终极符可以推导出的最后一个字符在句子中是第几个字符

则有如下语法制导定义:

产生式	语义规则
S' -> S	S.l = 1
S -> (L)	L.l = S.l +1 S.r = L.r + 1
S -> a	S.l = S.r print(S.r)
L -> L_1 , S	L_1.l = L.l S.l = L_1.l+2 L.r = S.r
L -> S	S.l = L.l L.r = S.r

翻译方案

S' -> { S.l = 1 } S { S'.r = S.r }

S -> { L.l = S.l +1 } (L) { S.r = L.r + 1 }

S -> a { S.l = S.r ; print(S.l) }

L -> { L_1.l = L.l } L_1 , { S.l = L_1.l+2 } S { L.r = S.r }

L -> { S.l = L.l } S { S.l = L.l }

预测翻译器

First(S) = { (, a }

First(L) = { (, a }

Follow(S) = { $, ',' ,) }

Follow(L) = { $, ',' ,) }

对于产生式 L -> L , S | S，First(L , S) = First(L) = { (, a }，而First(S) = { (, a }，二者有交集，故此文法不是 LL(1) 文法，无法构造出对应的预测翻译器。

```
syntaxTreeNode* S(SyntaxTreeNode* i){
    SyntaxTreeNode* i1 , s1 , s;
    if(lookahead == '('){       // S -> (L)
        match('(');
        i1 = i + 1;
        s1 = L(i1);
        s = s1 + 1;
    }
```

```
    else if(lookahead == 'a'){      // S -> a
        match('a');
        s = i;
        print(s);
    }
    else error();
    return s;
}
```

4.14 程序的文法如下：

`P→D`

`D→D;D | id:T | proc id;D;S`

(a) 写一个语法制导定义，打印该程序一共声明了多少个id。

(b) 写一个翻译方案，打印该程序每个变量id的嵌套深度。例如，当句型是`a:T;proc b;ba:T;S`时，`a`和`b`的嵌套深度是1，`ba`的嵌套深度是2。

(a)

产生式	语义规则
P -> D	P.val = D.val
D -> D_1 ; D_2	D.val = D_1.val + D_2.val
D -> id : T	D.val = 1
D -> proc id ;D_1 ; S	D.val = D_1.val + 1

(b)

P -> { D.depth = 0 } D

D -> { D_1.depth = D.depth + 1 } D_1 ; { D_2.depth = D.depth + 1 } D_2

D -> id : T { addType(id.entry , D.depth) ; print(id.entry) }

D -> proc id ; { D_1.depth = D.depth + 1 } D_1 ; S { addType(id.entry , D.depth) ; print(id.entry) }