4.12 $S \to (L) \mid a$

$L \to L, S \mid S$

定义 S.in、L.in 为继承属性，表示该文法符号推出的字符序列的前面有多少个字符；S.out、L.out 为综合属性，表示该文法符号推出的字符序列的最后一个字符是句子的第几个字符.（需要看完推出的序列）

语法制导定义：

$S' \to S$	$S.in = 0$	
$S \to (L)$	$L.in = S.in + 1$	$S.out = L.out + 1$
$S \to a$	$S.out = S.in + 1$	print (S.out)
$L \to L_1, S$	$L_1.in = L.in$ $S.in = L_1.out + 1$	$L.out = S.out$
$L \to S$	$S.in = L.in$	$L.out = S.out$

翻译方案：

$S' \to \{S.in = 0;\}\ S$

$S \to \{L.in = S.in + 1;\}\ (L)\ \{S.out = L.out + 1;\}$.

$S \to a\ \{S.out = S.in + 1;\ print(S.out);\}$.

$L \to \{L_1.in = L.in;\}\ L_1, \{S.in = L.out + 1;\}\ S\ \{L.out = S.out;\}$

$L \to \{S.in = L.in;\}\ S\ \{L.out = S.out;\}$

预测翻译器：先将上面的翻译方案消除左递归.

非左递归文法：$S \to (L) \mid a$

$L \to S L_1$

$L_1 \to , S L_2 \mid \varepsilon$

$S' \to \{S.in = 0;\}\ S$

$S \to \{L.in = S.in + 1;\}\ (L)\ \{S.out = L.out + 1;\}$.

$S \to a\ \{S.out = S.in + 1;\ print(S.out);\}$.

$L \to \{S.in = L_1.in;\}\ S\ \{L_1.in = S.out;\}\ L_1\ \{L.out = L_1.out\}$.

$L_1 \to \{S.in = L_1.in + 1;\}, S\ \{L_2.in = S.out;\}\ L_2\ \{L_1.out = L_2.out;\}$

$L_1 \to \varepsilon\ \{L_1.out = L_1.in;\}$

非终结符 S'、S、L、L_1 的翻译函数分别如下.

4.13-3

```
void S'(){
    syntaxTreeNode *in
    *in = 0;
    S(in);
}

syntaxTreeNode * S(syntaxTreeNode *in){
    syntaxTreeNode *out, *in1, *out1; //out表示S.out,in1和out1分别为L的继承属性和综合属性
    if(lookahead == 'a'){ //产生式S->a
        match('a');
        *out = *in + 1;
        print(*out);
    }
    else{  //产生式S->(L)
        *in1 = *in + 1;
        match('(');
        out1 = L(in1);
        match(')');
        *out = *out1 + 1;
    }
    return out;
}

syntaxTreeNode * L(syntaxTreeNode * i){
    syntaxTreeNode *out, *in1, *out1, *in2, *out2;   //out表示L.out,in1和out1分别为S的继承属性和综合属性,in2和out2分别为L1的继承属性和综合属性
    *in1 = *in;
    out1 = S(in1);
    *in2 = *out1;
    out2 = L1(in2);
    *out = *out2;
    return out;
}

syntaxTreeNode * L1(syntaxTreeNode * in){
    syntaxTreeNode *out, *in1, *out1, *in2, *out2;   //out表示L1.out,in1和out1分别为S的继承属性和综合属性,in2和out2分别为产生式右部L1（记作L2）的继承属性和综合属性
    if(lookahead == ','){   //产生式L1->,S L1
        *in1 = *in + 1;
        match(',');
        out1 = S(in1);
        *in2 = *out1;
        out2 = L1(in2);
        *out = *out2;
    }
    else{   //产生式L1->ε
        *out = *in;
    }
    return out;
}
```