H12

7.4

```
L -> id{
    insert(L.list, id.lexeme);//将id.lexeme插入L.list
}
L -> L1, id{
    insert(L.list, L1.list.begin, L1.list.end);//将L1.list中元素从头到尾插进L.list
    insert(L.list, id.lexeme);
}
D -> L : T{
    for (ele in L.list){
        enter(ele, T.type, offset);
        offset = offset + T.width;
    }
}
```

7.9

```
        .file   "ex7-9.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        pushq   %rbp
        movq    %rsp, %rbp //进入作用域
        jmp     .L2
.L5:
        movl    -4(%rbp), %eax //这两行是将j的值赋给i，这里使用了一个寄存器来实现值传递
        movl    %eax, -8(%rbp)
.L2:
        cmpl    $0, -8(%rbp)//这里是对于i != 0的一个判断
        jne     .L3//如果i != 0那么就会跳转到后面去判断j和5的大小
        cmpl    $0, -4(%rbp)//这里是对于j != 0的一个判断
        je      .L4//如果j != 0那么就会跳转到后面去判断j和5的大小
.L3:
        cmpl    $5, -4(%rbp)//这里是对于j > 5的一个判断
        jg      .L5//如果j > 5，那么就会继续进入循环
.L4:
        movl    $0, %eax
        popq    %rbp
        ret
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.5.0-3ubuntu1~16.04) 7.5.0"
```

7.10

(a)

.L2和.L4的出现是因为.L2是if语句中间代码生成时bool表达式判断出来是false的时候的B.next，而.L4则是while语句在中间代码生成时为了支持循环而设置的S.begin。

.L3代表的是if语句中true分支结束后的S1.next，而.L5则是当while语句中bool表达式判断出来是false的时候的S.next，最后一个.L1应该是用来退出的。然后他们合并在一起的时候就会出现重复。

(b)

它的作用可能是用来在return的时候将作用域复原的，因为一个函数结束，理论上来说要将ebp等值重新设置。该函数没有引用该标号我觉得是因为它不是一个被调用的函数。

(c)

.L3代表的是if语句中true分支结束后的S1.next，.L4表示的则是if语句判断出来是false的时候跳往的S2.begin，.L5表示的就是while循环语句bool表达式为true时的中的S1.begin。

没有L1和L2的原因可能是：

1.while循环语句的结构被更改了这样减少了一个标号的需求。

2.可能进行了冗余标号的合并和减去。

7.17

(a)

比如说对于a[i] [j]这种定义，那么纠将其地址定义为base + i * wi + j * wj，这样wi是存放一行元素的字节数，然后wj是存放行中一个元素的字节数，然后这个也可以拓展到多维的情况下。

(b)

```
L -> L1[E]{
    L.ndim = L1.ndim + 1;
    L.array = L1.array;
    w = getw(L.array, L.ndim );
    L.offset = L1.offset + E.place * w;
}

L -> id {
    L.place = id.place;
    L.offset = null;
    L.ndim = 0;
    L.array = id.place;
}
```