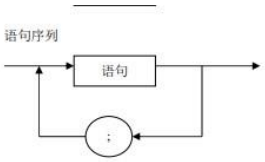
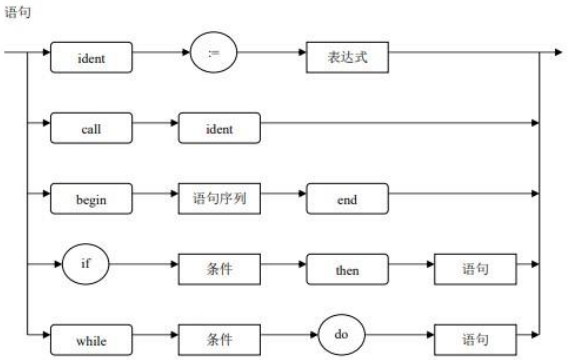
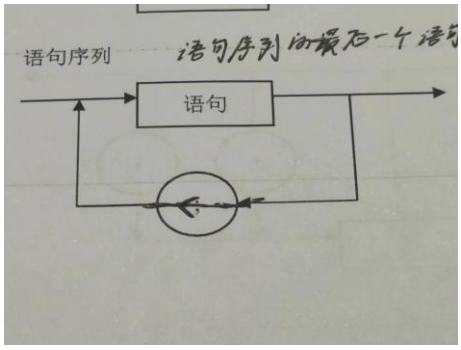
在实现具体功能之前，我把对“语句”的分号的判定修改了一下。教程给的是这个：

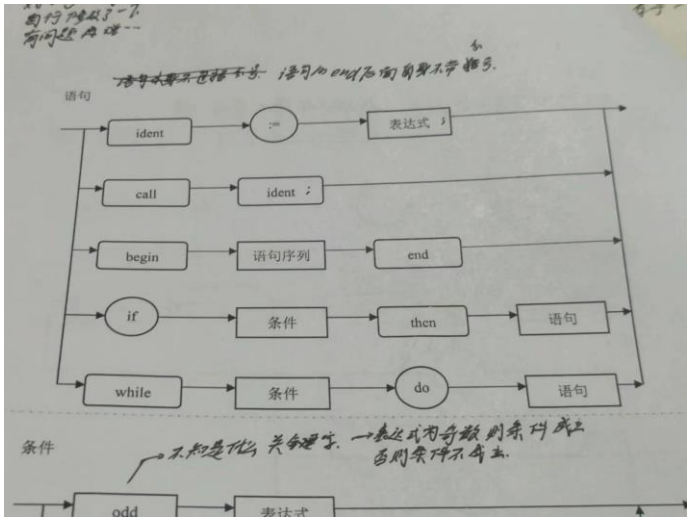



3




我改成了下面这样，感觉这样比较合理：



一共有 6 个非终结符，分别是程序体 block、语句 statement、条件 condition、表达式 expression、项 term、因子 factor。我没增加新的非终结符。

下面是具体实现方法，顺序是按照我自己的实现顺序写的：

4、else 子句

这个很简单，在 statement()函数里面加一个 else 判定就完了。

格式：

```
if 条件 then 语句 1 else 语句 2
```

汇编代码的生成模式：

① 条件
② JPC ⑤/⑥
③ 语句 1
④ JMP ⑥
⑤ 语句 2
⑥ …

2、print

这个也简单，在 statement()里增加了一个 print 开头的 else if。之后增加了一个 PRT 指令，写在 interpret()里面了。

格式：

```
print ( 表达式 , … ) ;
```

3、for

在 statement()里面加了一个 for 开头的 else if。

格式：

```
for ( var ident : ( low , up , step ) ) 语句
```

step 可省。

汇编代码的生成模式：

① 赋值 id=low

② 判断 id<=up 或者 id>=up 是否成立（<=或>=取决于 step 的正负）
③ JPC ⑦
④ 语句
⑤ 赋值 id=id+step
⑥ JMP ②
⑦ …

5、赋值语句扩展成赋值表达式

主要改动在 factor()里面的 ident 这块，主要是变量和数组类型。在后面读进来一个符号，看看是不是 :=。处理方案就跟处理一般赋值语句差不多，不是很难。

1、数组

主要改动在 block()里面，是定义数组方面的。还改在 factor()里面，是使用数组方面的。

定义格式:

```
var ident [ number ] … ;
```

使用格式:

```
ident [ 表达式 ] …
```

定义了新指令 STR 和 LDR，在 interpret()里面有解释。

6、setjmp 和 longjmp

这两个操作相当于游戏存档和恢复存档。定义了新指令 SJP 和 LJP，在 interpret()里面有解释。setjmp 当成表达式，主要改在 factor()里面；longjmp 当成语句，主要改在 statement()里面。interpret()里面也添加了配套的存档用的变量。