数组：

需处理部分：

-数组的声明/dimdeclaration/vardeclaration

-数组的赋值/statement/interpret

-数组的数组引用/factor/interpret

关键问题：

-数组在table中的存放结构以及必要信息

-数组下标的计算

--table中数组的数据结构

需要记录数组的name（ID），层次，相对数组首地址，类型（数组），维数，指向每维维度链表的指针；（修改了table的数据结构）

--声明

数组的声明和变量的声明要分开：当前得到一个标识符的话，并不知道其是数组类型还是变量类型，所以需要看下一个sym，如果是左方括号则表明这是数组，则调用dimdeclaration来处理数组类型（将数组存入table）；如果不是，则将这视为变量，存入table；（修改vardeclaration以及新设一个函数来处理数组类型dimdeclaration）

维数的确定是在dimdeclaration中，循环读入[expression]，直到下一个字符不是’[‘；

--地址计算和存取

多维数组下标地址的计算利用公式来实现，其中每一维的维数已经在声明时确定，所以可以从table里得到，但是下标确是变动的，所以要执行时确定，将这个公式的计算翻译成相对应的pcode代码，在栈顶进行操作，所以计算得到的数组内相对偏移量放置在栈顶，因此需要利用此时的栈顶值来得到数组元素存储单元的地址，现有指令STO和LOD无法实现，因此新增两个指令STA和LAD来实现；

STA：将栈顶数据存放到“次栈顶的偏移量加上数组元素首地址”所指示的存储单元

LAD：将“次栈顶的偏移量加上数组元素首地址”所指示的存储单元内的数据复制到栈顶

这两个指令的格式和STO、LOD一样，只是在interpret里的具体实现不同；

（修改了interpret，增加了两条新指令STA,LAD，以及statement和factor里面对数组元素的

地址计算）

规范说明：

-实现多维数组（最大维数可以设定）

-实现数组的声明、存取、引用（也可以作为参数传递）

-实现数组错误恢复

-数组相关动作都可实现

Break：

需处理部分：

-地址回填

-错误处理

关键问题：

-多地址回填（回填链）

-嵌套地址回填

--嵌套地址回填

嵌套的形式如下：

For ......

...

Break；

For ......

...

Break;

...

...

...

我们可以利用statement的嵌套调用来写break；把break当做一条语句，在读到for的时候需要调用statementA，在statementA里面会调用statementB来处理内层for，在statementB里接着调用statementC，假如内层for只有一个break；语句，则statmentC分析break，先会生成jmp指令，这条指令的代码保持在一个全局变量里，为了日后回填，然后break分析完毕，回到内层for的处理，这时候回到的是statementB，那么到了for的末尾，这个时候知道了回填什么地址（jmp应跳到这里），所以用刚刚的全局变量里面存放的地址回填回去。此时还有一个问题，就是全局变量跟着全局变，那我可能会覆盖了statementA里面的break的地址，所以我需要在调用statement之前保存全局变量之前的值，然后再调用statement，statement出来之后，处理完回填后再将全局变量的旧值恢复，然后退出此层的statement，这样就不会覆盖之前的break地址。

--多地址嵌套

如果一个for里面有多个break；这是可能的，如下：

For ......

...

If ......

Break；

If......

Break；

...

此时第二个break的地址会覆盖掉第一个break地址，所以上述方法只会回填第二个break的地址，但是这有一个共同之处，就是这两个break应回填的地址都是一样的（即for的跳出地址），所以我可以用一个链表的形式把这两个break的地址链起来，然后for的结尾处处理这个回填链即可；

同时，循环里利用flag这个标志位来判断全局变量里回填链是否有效（即是否有break要回填）；

--错误处理

Break的作用是跳出当前循环，如果没有在循环内调用，则会报错，实现原理就是设置一个信号sign，如果我前面分析的时候有循环，那我就将这个sign置位，每读到break的时候我就查看则个sign是否有效就可判断是否出错，当然，这个sign是和上述的全局变量一起的结构体中的一个域；