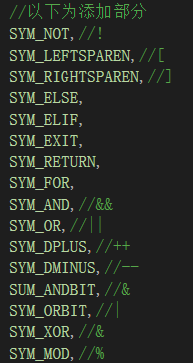
# 词法分析部分

## 词法分析过程

从源文件中读出若干有效字符，组成一个 token 串，识别它的类型 为保留字/标识符/数字或是其它符号。如果是保留字，把 sym 置成相应的保留字类型，如果是 标识符，把 sym 置成 ident 表示是标识符，于此同时，id 变量中存放的即为保留字字符串或标识 符名字。如果是数字，把 sym 置为 number,同时 num 变量中存放该数字的值。如果是其它的操作符， 则直接把 sym 置成相应类型。经过本过程后 ch 变量中存放的是下一个即将被识别的字符

## 我们的处理部分

将判断//和/\*单独作为一个条件，在其中判断是否是除号，因此在csym和ssym中删去判断除号部分，值得注意的是当遇到//时，我们应该执行cc==ll，getchar（）来跳过当前行的其余部分



相应的特殊符号存入csym和ssym，保留字存储在word和wsym中。

# 语法分析部分

### 语法分析过程

采取自顶向下的递归下降分析法，语法分析同时根据语义生成相应代码，并提供了出错处理机制，其中语法分析主要由：程序体分析过程（block）、常量定义分析过程（constdeclaration）、变量定义分析过程（vardeclaration）、语句分析过程（statement）、表达式处理过程（expression）、项处理过程（term）、因子处理过程（factor）和条件处理过程（condition）等过程。这些过程在结构上构成一个嵌套的层次结构。除此之外，还有出错报告过程（error）、代码生成过程（gen）、测试单词合法性及出错恢复过程（test）、登录名字表过程（enter）、查询名字表函数（position）以及列出过程（listcode）作过语法分析的辅助过程。

### 程序体处理过程

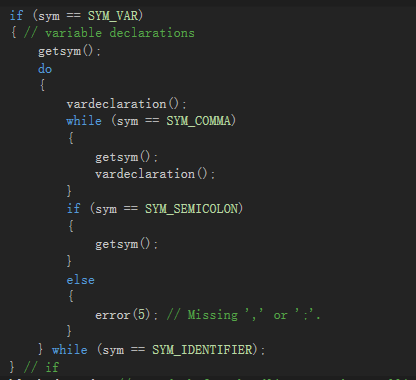
语法分析开始后，首先调用程序体处理过程处理分程序。过程入口参数置为：0层、tx 、出错恢复单词集合为句号、声明符或者语句开始符号。首先把局部数据段分配指针为3，准备分配3个单元供运行期存放静态链SL，动态链DL和返回地址RA

#### 声明部分

如果是const，调用过程constdeclaration(),判断是否满足常量声明定义，如果满足，将标识符登陆进符号表中,name为id，kind为SYM\_CONST，value为数值num这些都是在词法分析部分返回的值，并且tx++；tx为符号表的栈顶

如果是var，先判断是否满足语法，调用过程vardeclaration()，变量的level为当前block的level，address为栈的地址（偏移量dx），且dx++，tx++。

在这里有一点和语法图有一些出入：



这样的代码会对下面的代码也认可



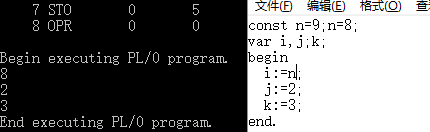
而且这样的顺序要求常数声明必须在变量声明之前，变量声明必须在该层的过程声明之前，即是顺序判断的，那么以下两种代码自然不被允许：



在结束声明部分会用block\_dx存储当前的栈顶

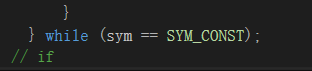
如果是procedure，遇到标识符，则调用enter，其中level=当前level

且enter函数没有解决或者考虑重复定义的问题，因为查找符号表是从上往下查找，因此如果重复定义，则以最后一次定义为准，如下所示：



##### 声明修改部分

为了符合语法图规范，我们对声明的循环判断条件进行改动



和



#### 其余分析部分

在语法调用部分中，注意一致性：进入每一个语法单位处理这个程序之前，其第一个单词已经读出，退出时，应该读入下一个语法单元的第一个单词

# 翻译和代码生成：

对于符号表，单趟编译中对于并列的函数（或分程序）其相应的符号表不会同时存在，

gen(),把三个参数f，l，a组装成一条目标指令并存放于code数组中，增加cx的值，cx表示下一条即将生成的目标指令的地址。

其中，f，l，a的含义如下表

|  |  |  |  |
| --- | --- | --- | --- |
| F | L | A | 含义 |
| INT | —— | 常量 | 在数据栈中分配空间 |
| LIT | —— | 常量 | 常数置于栈顶 |
| LOD | 层次差 | 数据地址 | 将变量值置于栈顶 |
| STO | 层次差 | 数据地址 | 将栈顶的值赋予某变量 |
| CAL | 层次差 | 程序地址 | 过程调用指令 |
| JMP | —— | 程序地址 | 转移指令 |
| JPC | —— | 程序地址 | 转移指令 |
| OPR | —— | 运算类别 | 一组算术或逻辑运算指令 |

修改部分：

非、取模：

在oprcode中增添相应运算符，其中在facbegsys中增添SYM\_NOT，

取模部分的修改主要在term函数中，类似于乘除

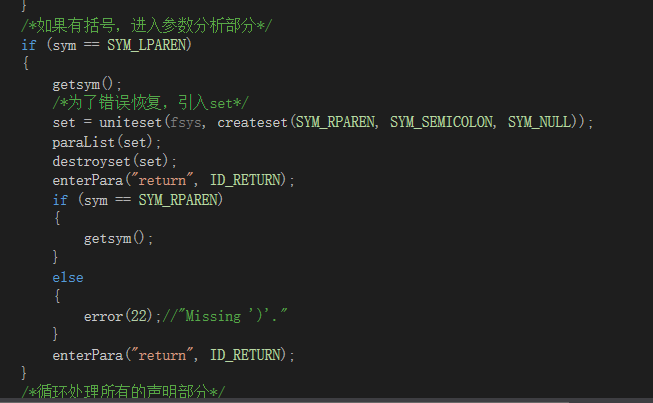
# 出错恢复

在进入某个语法单位时，调用test，检查当前符号是否属于该语法单位的开始符号，如不属于，则滤去所有的非开始符号和后继符号的符号

在语法单位分析结束时，调用test，检查当前符号是否属于调用该语法单位的应有的后继符号集合，如果不属于，滤去后继符号和开始符号外的所有符号

### 函数调用和返回

在函数调用和返回时，我们这里有一个很有意思的处理，就是用一个标识符来存储参数的个数，将其的id设置为return，将其address设置为参数个数，然后在参数返回时用到它来恢复现场，return标识符每次在分析参数时填入



每次在程序读到一个SYM\_RETURN时返回，并用“return”查找标识符return，厉害了