**实验目标**

本次实验题目要求是实现一个PL0语言的编译器，实现词法分析，语法分析和执行代码三个功能。

具体要求如下：

1. 要实现两个函数，getsym()用于得到一个symbol，getch() 用于读取代码中一个一个的字符。
2. 语法分析程序block()函数要实现两件事：1）将说明部分的相关变量放入到table表中，2）生成PL0语言的目标指令，存入code数组中。
3. 建立一个解释执行目标程序的函数。

我采用的是递归处理程序的形式实现的，没有采用建立分析表的方式实现。下面依次讲

解各个方法的实现。

**词法分析程序**

这个板块我使用一个lexer类实现了相关的功能，下面对其所有的属性和方法进行介绍。

* br，文件读写流，私有属性，用于对代码文件的读取。
* line，字符串，私有属性，使用br一次读取一行，读取的结果放入此变量中。
* index，数，私有属性，对line字符串是一个字符一个字符处理的，index即为即将要处理的字符的下标。
* linelength，数，私有属性，读取的一行文本的长度，用于判定index是否越界。
* lineNum，数，公有属性，表示的是当前处理的是第几行，这个属性在错误处理模块需要使用，因此将其设为公有属性。
* ch，字符，私有属性，用于当前处理的字符也就是line[index]
* sym，SYMBOL，公有静态属性，SYMBOL类型是一个枚举类型，也就是说sym是一个当前处理的symbol，这个symbol是用character组成的。由于在语法处理过程中需要大量的引用当前处理的sym，所以将其设为了公有静态变量。
* id，字符串，公有静态属性，这个用于存储读取到的标识符。
* num，数，公有静态属性，用于存储读取到的数字。
* word，字符串数组，私有属性，此属性存储了PL0所允许的几个保留字，用于区分保留字和标识符。
* wsym，SYMBOL数组，私有属性，与word数组是一一对应的，同一个下标的word和wsym分别代表的是保留字和保留字所对应的symbol。
* wordLen，数，私有属性，用于表示word数组的大小，也就是关键字的多少。
* puncLen，punc，psym，数，字符数组，SYMBOL数组，私有属性，这三个和word对应的三个属性是含义相似的，不过这三个指的是标点符号。
* 构造函数，此函数只需要一个形参，就是一个文件的读取流对象。
* 一些getter，这是为了应对私有属性被访问的情况。
* getch函数，一次读取一行，赋值给line，并对index，linelength，ch等变量进行更新，返回新读取的一个字符。
* getsym函数，无参，不需要向getch有一个新读取的一个symbol的返回值，因为它把结果写入了sym这个静态变量里面。它的具体实现，以读取标识符为例。

**if** (Character.*isLetter*(**ch**)) {  
 *id* = **""**;  
 **do** {  
 *id* += **ch**;  
 **ch** = getch();  
 } **while** (Character.*isLetter*(**ch**) || Character.*isDigit*(**ch**));  
 **word**[0] = *id*;  
 **int** i = **wordLen**;  
 **while**(!**word**[i].equals(*id*)){  
 i--;  
 }  
 **if**(i>0){  
 *sym* = **wsym**[i];  
 }**else**{  
 *sym* = SYMBOL.***SYM\_IDENTIFIER***;  
 }  
}

**遇到的问题**

Java的文本流对象读取一行时，会自动将末尾的换行符给处理掉，如果没有换行符的特殊判断的话，getsym这个函数面对封装好的getch函数完全不知道内部的实现情况，只是一味的要character，如果没有末尾的特殊判定，那么在读取到标识符的时候十分容易上一行的尾和下一行的头的一堆字符连起来形成一个标识符。

我的处理方式是在每一行的末尾添加了一个空格，因为空格既不影响语法结构，又是可以忽略的元素，还可以作为一个间断隔开首尾。

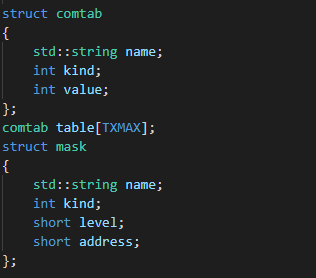
**语法分析程序**

语法分析程序需要实现两个任务，一是将所有说明部分的变量放到table表中，这里补充一点PL0语言是先声明再有执行语句的，所以不用担心在执行语句的时候突然蹦出一个变量声明来，这两个区域是严格分开的。二是生成目标代码，PL0预设的指令格式为（f,l,a）， 分别代表功能码，层次差，位移量。功能码有8个，如下所示：

* LIT，将常数放到栈顶，a域为常数
* LOD，将变量放到栈顶，a域为变量在所说明层中的相对位置，l为调用层与说明层的层差值。
* STO，将栈顶的内容送到某变量单元中。a,l域的含义与LOD的相同。
* CAL，调用过程的指令。a为被调用过程的目标程序的入中地址，l为层差。
* INT，为被调用的过程（或主程序）在运行栈中开辟数据区。a域为开辟的个数。
* JMP，无条件转移指令，a为转向地址。
* JPC，条件转移指令，当栈顶的布尔值为非真时，转向a域的地址，否则顺序执行。
* OPR，关系和算术运算。具体操作由a域给出。运算对象为栈顶和次顶的内容进行运算，结果存放在次顶。a域为0时是退出数据区。

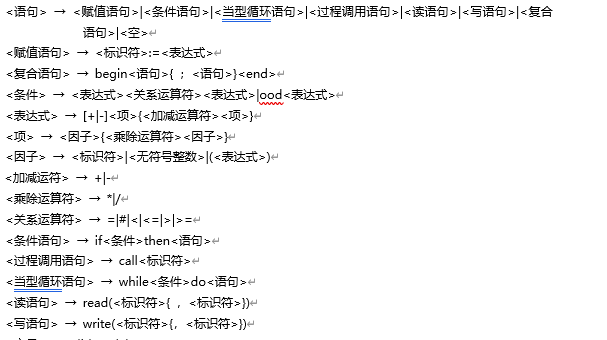
这个部分实现了一个parser类，用于实现语法分析相关的功能，下面对其所有的属性和方法进行一个说明。

对于不同类别的声明会有不同的处理措施，本质上都是填table这个数组，tx作为table的一个指针是一个全局变量。值得一提的是mask这个类型，这个类型是为了处理变量和过程需要填写的信息有四个，常量需要填写的信息有三个的问题。

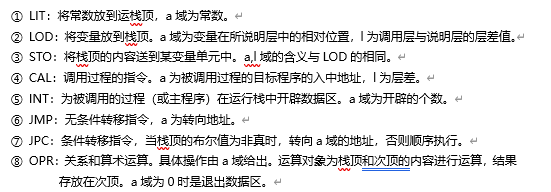


上图就是两个类型，但占用空间是一样的。

完成了填表的操作之后，就是生成PL0目标代码了，具体就是针对于每一个语句，设计一套等价的执行代码：



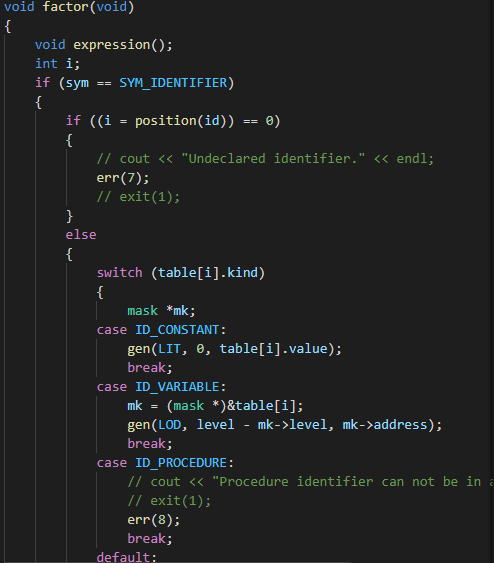
上图是关于PL0语言的一个文法表示。可见需要设计赋值语句，条件，读语句，写语句和复合语句的等价目标代码。



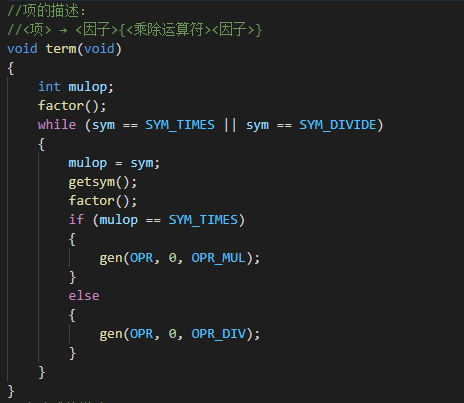
上图是PL0允许的目标指令，

对于赋值语句（a = 5）将常数5放到栈顶，执行STO指令。对于条件语句（a + b > c +d）先处理左边的表达式，将a变量放到栈顶（变量在栈空间内被分配了一块地方），将b变量放到栈顶，执行OPR 0 ADD命令，此命令是将次顶和栈顶的值进行ADD运算，而后将结果存在次顶中（因为运行完之后，次顶就是栈顶！），从两个例子可以看出来栈顶是相当重要的，操作命令都是和栈顶有关的。下面以条件语句为例子进行说明。

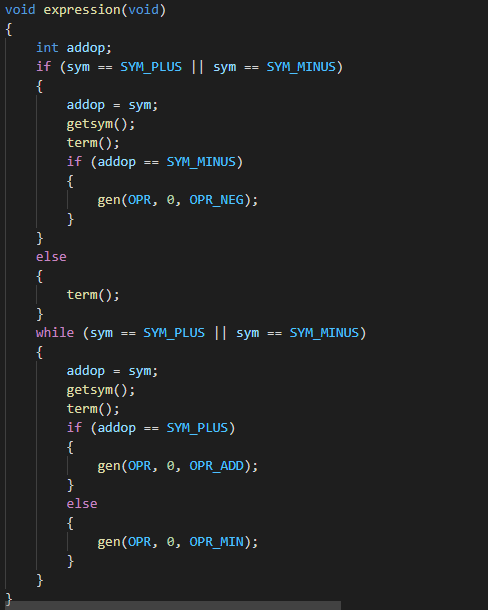
条件->表达式->加减项->因子



上图是在处理因子的情况，



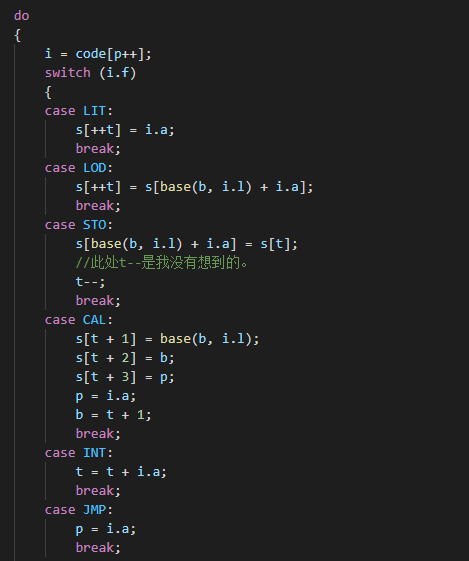
上图是处理项的相关代码。



上图是关于对于整个表达式读取并处理（生成目标代码）的函数。

**建立一个解释执行目标程序的函数**

执行的时候，内存中只有存放目标程序的code数组和运行时的栈式数据空间S，实行这个函数的时候，要重点了解每个过程被调用（包括主过程，要不然dx初始值怎么是3呢）时，栈顶分配三个单元，一个单元是调用链最开始的过程的数据段的基地址，一个单元是该过程运行之前正在运行的过程的数据区基地址，一个单元是当时程序寄存器P的值，用于过程执行完的时候继续执行之前的代码。



上图是执行函数的一个片段，其中s是栈式数据区，执行的逻辑大致就是根据每条命令要做的事情，需要的先把操作数移到栈顶，再进行运算。