本次课程设计实现的是一个简单的PL0编译器，包括词法分析器，语法分析器和代码生成器三部分，能够实现大部分PL0程序的编译工作，并生成基于指定指令集的目标汇编文件。  
我这次选择的题目是第三题，对于题中给出的语法扩展要求做到了全部满足，加入了一定量的语义分析，能够进行一些常见的静态检查工作，并且对于部分语法错误也能够给出精确到行的提示。

**1、扩展要求**

（1）给PL/0语言增加像C语言那样的形式为/\* …… \*/的注释。

（2）给PL/0语言增加带else子句的条件语句和exit语句。

（3）给PL/0语言增加输入输出语句。

（4）给PL/0语言增加带参数的过程。

（5）给PL/0语言增加布尔类型。

（6）给PL/0语言增加数组类型。

（7）给PL/0语言增加函数类型。

（8）给PL/0语言增加实数类型。

• 在类型定义ident = TypeExp中，TypeExp若不是数组类型，则报告错误。即只允许给数组类型命名，只对数组可以类型定义。

• 若exit语句没有处于任何while语句中，则是一个错误。

• 读写语句的变量引用和表达式只能是整型或实型，I/O无Boolean类型。

• 过程和函数是有区别的，call语句只能调用过程，表达式中只能出现函数调用。

• 本语言有布尔类型，有布尔常量false和true。C语言虽有逻辑运算，但没有布尔类型。本语言不存在把0看成假，把非0看成真。Boolean类型严格。

• 数组类型是按名字等价而不是结构等价。

• 布尔类型的表达式按短路方式计算。

**2、目标指令集**

|  |  |  |  |
| --- | --- | --- | --- |
| 指令格式 | | | 作用 |
| 指令 | 目标操作数 | 源操作数 |
| IMM | ax/bx | number | 将立即数存到寄存器中 |
| LDA | ax | address | 将地址存到ax寄存器中 |
| LD | ax/bx | address | 将给定地址中的内容加载到寄存器中 |
| STO | address | ax/bx | 将寄存器中的内容存放到给定地址中 |
| PUSH |  | ax/bx | 将寄存器中的内容存放到栈顶 |
| POP | ax/bx |  | 将栈顶的内容存放到寄存器中 |
| CALL | address |  | 过程调用，将返回地址压入栈中 |
| ENT |  |  | 将bp寄存器内容压入栈中， 并将当前活动记录基址存入bp寄存器 |
| LEV |  |  | 恢复bp寄存器原先内容， 并更新pc寄存器内容为调用返回地址 |
| JZ | address |  | 当ax寄存器内容为0时跳转 |
| JNZ | address |  | 当ax寄存器内容不为0时跳转 |
| JMP | address |  | 无条件跳转 |
| HLT |  |  | 停机 |
| ADD | ax/bx | 栈顶 | 栈顶内容与寄存器内容相加， 结果存入寄存器中 |
| SUB | ax/bx | 栈顶 | 栈顶内容减去寄存器内容， 结果存入寄存器中 |
| MUL | ax/bx | 栈顶 | 栈顶内容与寄存器内容相乘， 结果存入寄存器中 |
| DIV | ax/bx | 栈顶 | 栈顶内容除以寄存器内容， 结果存入寄存器中 |
| MOD | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行取模运算， 结果存入寄存器中 |
| AND | ax | 栈顶 | 栈顶内容与ax寄存器内容进行逻辑与运算， 结果存入ax寄存器中 |
| OR | ax | 栈顶 | 栈顶内容与ax寄存器内容进行逻辑或运算， 结果存入ax寄存器中 |
| NOT | ax |  | ax寄存器内容进行逻辑非运算， 结果存入ax寄存器中 |
| LT | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行小于比较运算， 结果存入ax寄存器中 |
| GT | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行大于比较运算， 结果存入ax寄存器中 |
| LE | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行小于或等于比较运算， 结果存入ax寄存器中 |
| GE | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行大于或等于比较运算， 结果存入ax寄存器中 |
| NE | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行不等于比较运算， 结果存入ax寄存器中 |
| EQ | ax/bx | 栈顶 | 栈顶内容与寄存器内容进行等于比较运算， 结果存入ax寄存器中 |
| ODD | ax |  | 判断ax寄存器内容是否是奇数， 结果存入ax寄存器中 |
| NEG | ax/bx |  | 对寄存器内容进行取相反数操作， 结果存入寄存器中 |
| WRITE | ax/bx |  | 将寄存器中的内容输出到屏幕上 |
| READ | address |  | 从键盘输入中读取内容存放到给定地址中 |

**3、运行时环境**

|  |  |  |  |
| --- | --- | --- | --- |
| 寄存器 | | | |
| ax | 存放整数 | | |
| bx | 存放实数 | | |
| sp | 存放当前栈顶地址 | | |
| bp | 存放当前活动记录基址 | | |
| pc | 存放下一条执行指令的地址 | | |
|  | | 内存地址 |
| 运行栈 | | 大 |
|  |
|
|
| 小 |
| 全局变量数据段 | | 1024 |
| 全局常量数据段 | | 0 |

运行栈：

sub\_function(arg1, arg2, arg3):

| .... | high address

+---------------+

| arg: 1 | new\_bp + 4

+---------------+

| arg: 2 | new\_bp + 3

+---------------+

| arg: 3 | new\_bp + 2

+---------------+

| return address| new\_bp + 1

+---------------+

| old bp | new bp

+---------------+

| local var 1 | new\_bp - 1

+---------------+

| local var 2 | new\_bp - 2

+---------------+

| .... | low address

1.词法分析器采用的是暴力分析法。该部分较为简单，没有什么值得探讨的。  
2.语法分析器采用的是自顶向下的递归下降分析法。  
由于PL0文法简单，而且每个语句前都有特定的关键字，因此不容易产生二义性，对于手工实现的编译器，递归下降分析法无疑是一个较好的选择。  
由于文法已经给定，对于特定的文法结构只需套用模板就能实现，比较简单，只是很多内容写起来比较繁琐，要有足够的耐心，同时还要足够的细心，或者因为遗漏一个分号，就能导致程序出错，很多时候出错的原因让人摸不着头脑。  
3.本程序代码生成和语法分析共同进行，语法分析输出直接是目标代码，并没有生成抽象语法树，实际上语法树更有利于之后的语义分析和代码优化工作。

本次程序设计实现主要分两大部分，词法分析部分和语法制导的翻译部分。  
词法分析部分主要函数为getToken，该函数通过一个getNextChar函数逐个获得字符，经过词法分析部分，并最终返回一个词法token，token的类型是一个自定义的枚举体Token，其中包括变量，常数，关键字，特殊符号等类型，返回的词法token就作为词法分析对语法分析的接口。而getNextChar函数并非直接从源文件通过getChar函数逐个读取字符，由于文件IO操作较慢，应尽量减少对文件的操作。所以本程序通过设置一个缓冲行，每次从源文件中读取一行字符，来完成对文件的读取工作，同时对于多读取到的字符还可以通过ungetChar函数来将字符放回缓冲区中。  
本程序语法分析部分大概设置了20个函数，基本每条产生式对应一个函数，主函数是program，同时也是语法分析的对外接口，贯穿整个语法分析部分的还有一个函数为match，其功能主要是检验给定的token是否匹配当前的词法token，如果匹配则调用getToken获得下一个token，如果不匹配则说明源码不符合给定的文法结构，这时候应该给出一定的错误提示，并积极采取错误恢复措施，来完成更多的语法分析工作。本程序的错误恢复功能比较简陋，如果是分号的不匹配，即当前需要一个分号则补一个分号，如果是其他类型的不匹配，则跳过当前不匹配的token，获得下一个token。通过函数的层层递归调用，如果每次都能准确匹配token，则说明源码完全符合给定语法结构。  
代码生成部分由emitCode，emitCodeRollback，rollback三个函数组成，emitCode函数是在目标文件中直接输出目标代码，而后两个函数用于目标代码回溯使用，主要是用于选择和循环结构的代码生成。  
  
虽然对于课设给定的功能要求都做到了基本的满足，但值得进步的空间还是很大，接下来将对本编译器的一些缺陷及存在的问题进行探讨分析。  
首先，是设计和编码的问题。一个优良的设计不仅能够出色地完成功能要求，还要有极强的可扩展性，即可以在不破坏整体结构的基础上随时增加新功能。但本编译器的实现在设计上就出现了问题，因为没有做到提前的完整设计，完全是边做边改，导致整个程序的可扩展性极差，有些功能要实现就需要推倒现有的工作成果，重头再来，实在是费时费力。而且因为数据结构的设计问题，导致有些地方越写越复杂，完全就是不断地堆全局变量和代码行，使程序没有一点美感，而且代码很多地方晦涩难懂，可读性极差。  
其次，是编译器的错误提示和恢复机制不完备。在最初的实现版本中完全没有错误恢复功能，遇到一个语法错误就强制程序退出，这实在是太不人道，也给自己的调试工作带来了很多不便。编译器的错误提示功能是极为重要的一部分，因为用户需要根据编译器给出的提示进行代码的修改，而要做到准确无误的代码错误提示，是一件基本不可能实现的事。我们要做的就是尽可能多的、尽可能准确地给出错误提示，即语法分析过程如果遇到错误，要能够自动地跳过当前错误，继续分析，这样才能给用户提供更多的辅助信息，而不是简简单单的程序退出。在之后的版本中本编译器采用了一些简单的错误恢复措施，能够跳过一些常见的错误，比如缺少分号等易犯错误，对于程序的第一个语法错误能够较为准确地给出提示，但之后的错误提示就会不太准确。总之，本编译器虽然有错误提示，但功能不完善，会带有一定的随机性，有时候甚至会抛出异常，出现不知名的错误。  
最后，本次实现的只是编译器，而且指令集并不是基于某个常用平台的标准汇编语言，有一些指令甚至是自己为了代码生成的需要而自行创建的，这实在没有十足的说服力。虽然成功地生成了目标代码，但代码能不能够正常运行也是一件说不定的事，想要运行目标代码，有两种基本实现，一是生成汇编语言，汇编后直接运行，但由于没有学过汇编语言，对汇编不熟悉，而且汇编语言运行时环境特别复杂，涉及到寄存器分配等难题都不是一时半刻就能攻克的。另外一种实现就是设计一个虚拟机，为自己生成的目标代码写一个解释器，这相比生成汇编代码而言相对简单一些，而且虚拟机的运行环境可以基于栈式计算机和少量的寄存器，这会减小生成目标代码和编写解释器的难度。可以通过对编译器所生成目标代码的再次解析，来翻译成C或C++语言来运行，这并不是一件复杂的事，但时间不允许再去写解释器，使得编译器的生成代码不具有说服力，因此，如果再实现一个解释器，将会更加完美。  
  
本次课设共用时大概5天，编码用了两天半，调试用了一天，小bug的修改以及小功能的添加又零零散散用了大概一天半，总共五天左右。由于之前有练习过类似的C语言mini编译器，所以编码过程思路比较清晰，比较顺利的完成了编码工作，接下来的调试工作也没有太费劲，基本上都是些小错误，这说明整体上的设计是没有问题的，接下来就是不断的测试，修改一些局部bug。总的来说，实现过程并没有太大障碍，比想象中顺利一些，但实现的功能和之前立下的鸿鹄浩志比起来还是有些简陋，但奈何考试紧张，时间有限，所以可以考虑在暑假期间完完整整地写一个编译器，一来是对所学内容的复习，二来写一个功能完备的编译器也是对自己能力的一种检验。  
以前都是写些小程序，这次的1300行代码，算是自己写过的比较大的程序了，虽然整个实现过程有些小曲折，但也学到了很多知识，比如如何从头设计一个程序，如何高效地调试代码。  
总之，课设已经过去，但只局限于课设水平，是不会成为强者的，唯有不断读经典书籍，不断练习，方能铺就康庄大道。