成绩评定

武昌工学院
WUCHANG INSTITUTE OF TECHNOLOGY

《编译原理》

报告名称： 基于 PL/0 编译程序生成器的设计
学　　院： 信息工程学院
专业年级： 计科 1801
姓　　名： 宋媛　**学　　号：** 183101940111
任课老师： 舒畅　**完成时间：** 2018.12.30

武昌工学院
2018 年 09 月印制

1. 报告名称

根据编译程序的基本结构和工作原理，以 PL/0 程序为基础，编写并调试编译过程的中间代码。

2. 设计框架及内容要求

2.1 基本要求

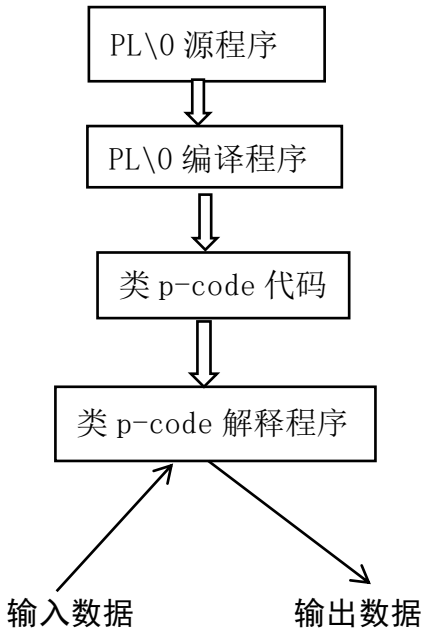
以一个 PL/0 程序为基础，写出编译过程中的中间代码。

基于编译程序的工作原理、工作过程、基本方法和实现技术，采取报告的形式，一方面掌握计算机程序设计语言的编译和实现方法，加深对程序设计语言的理解。另一方面在学习理论知识的同时增强实践能力，达到分析简单编译程序的能力。通过报告的撰写和 PL/0 编译程序的调试和运行，掌握编译的整个过程，熟悉各环节的衔接性。

3. 设计思路要求

3.1 PL/0 编译程序结构

3.1.1 系统结构框架图



3.1.2. 各阶段设计思路

PL/0 语言是 Pascal 语言的一个子集，这里分析的 PL/0 的编译程序包括了对 PL/0 语言源程序进行分析处理、编译生成类 PCODE 代码，并在虚拟机上解释运行生成的类 PCODE 代码的功能。

PL/0 语言编译程序采用以语法分析为核心、一遍扫描的编译方法。词法分析和代码生成作为独立的子程序供语法分析程序调用。语法分析的同时，提供了出错报告和出错恢复的功能。在源程序没有错误编译通过的情况下，调用类 PCODE 解释程序解释执行生成的类 PCODE 代码。

3.1.3 PL/0 语言介绍

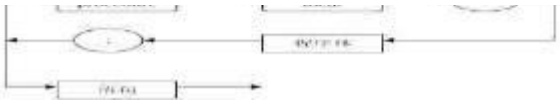
PL/0 程序设计语言是一个较简单的语言，它以赋值语句为基础，构造概念有顺序、条件

和重复(循环)三种。PL/0 有子程序概念，包括过程定义(可以嵌套)与调用且有局部

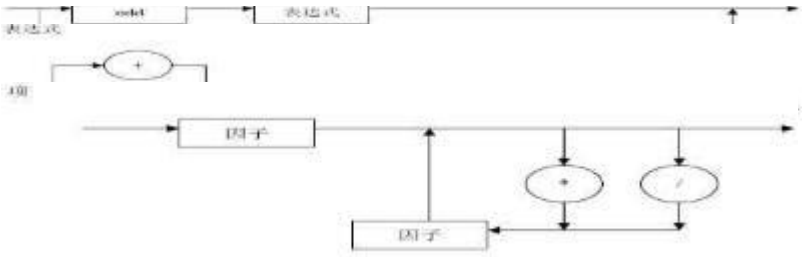
变量说明。PL/0 中唯一的数据类型是整型，可以用来说明该类型的常量和变量。当然 PL/0

也具有通常的算术运算和关系运算。

3.1.4 PL/0 语法图如下



2



3

3.2 各功能模块描述

3.2.1 词法分析子程序分析：

词法分析子程序名为 GETSYM，功能是从源程序中读出一个单词符号（TOTAKEN），把它的信息放入全局变量 SYM、ID 和 NUM 中，字符变量放入 CH 中，语法分析器需要单词时，直接从这三个变量中获得。Getch 过程通过反复调用 Getch 子过程从源程序过获取字符，并把它们拼成单词。GETCH 过程中使用了行缓冲区技术以提高程序运行效率。

词法分析器的分析过程：调用 GETSYM 时，它通过 GETCH 过程从源程序中获得一个字符。如果这个字符是字母，则继续获取字符或数字，最终可以拼成一个单词，查保留字表，如果查到为保留字，则把 SYM 变量赋成相应的保留字类型值；如果没有查到，则这个单词应是一个用户自定义的标识符（可能是变量名、常量名或是过程的名字），把 SYM 置

为 IDENT，把这个单词存入 ID 变量。查保留字表时使用了二分法查找以提高效率。如果 Getch 获得的字符是数字,则继续用 Getch 获取数字，并把它们拼成一个整数或实数，然后把 SYM 置为 INTEGER 或 REAL，并把拼成的数值放入 NUM 变量。如果识别出其它合法的符号（比如：赋值号、大于号、小于等于号等），则把 SYM 则成相应的类型。如果遇到不合法的字符，把 SYM 置成 NUL。

3.2.2 语法分析

语法分析子程序分析：

语法分析子程序采用了自顶向下的递归子程序法，语法分析同时也根据程序的语义生成相应三元代码，并提供了出错处理的机制。语法分析主要由分程序分析过程（BLOCK）、参数变量分析过程（ParaDeclaration）、参数变量处理过程（ParaGetSub）、数组处理过程（ParaGetSub）、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）构成。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）以及列出类 PCODE 代码过程（Listcode）作过语法分析的辅助过程。

由 PL/0 的语法图可知：一个完整的 PL/0 程序是由分程序和句号构成的。因此，本编译程序在运行的时候，通过主程序中调用分程序处理过程 block 来分析分程序部分（分程序分析过程中还可能会递归调用 block 过程），然后，判断最后读入的符号是否为句号。如果是句号且分程序分析中未出错，则是一个合法的 PL/0 程序，可以运行生成的代码，否则就说明源 PL/0 程序是不合法的，输出出错提示即可。

相关过程有:

```
block()
constdeclaration()
vardeclaration()
statement()
condition()
expression()
term()
```

factor()

3.2.3 语义分析

语法分析开始后，首先调用分程序处理过程（Block）处理分程序。过程入口参数置为：0 层、符号表位置 0、出错恢复单词集合为句号、声明符或语句开始符。进入 Block 过程后，首先把局部数据段分配指针设为 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。然后用 Tx0 记录下当前符号表位置并产生一条 Jmp 指令，准备跳转到主程序的开始位置，由于当前还没有知到主程序究竟在何处开始，所以 Jmp 的目标暂时填为 0，稍后再改。同时在符号表的当前位置记录下这个 Jmp 指令在代码段中的位置。在判断了嵌套层数没有超过规定的层数后，开始分析源程序。首先判断是否遇到了常量声明，如果遇到则开始常量定义，把常量存入符号表。接下去用同样的方法分析变量声明，变量定义过程中会用 Dx 变量记录下局部数据段分配的空间个数。然后如果遇到 Procedure 保留字则进行过程声明和定义，声明的方法是把过程的名字和所在的层次记入符号表,过程定义的方法就是通过递归调用 Block 过程，因为每个过程都是一个分程序。由于这是分程序中的分程序，因此调用 Block 时需把当前的层次号 Lev 加一传递给 Block 过程。分程序声明部分完成后，即将进入语句的处理，这时的代码分配指针 CX 的值正好指向语句的开始位置，这个位置正是前面的 Jmp 指令需要跳转到的位置。于是通过前面记录下来的地址值，把这个 Jmp 指令的跳转位置改成当前 cx 的位置。并在符号表中记录下当前的代码段分配地址和局部数据段要分配的大小（DX 的值）。生成一条 INT 指令，分配 DX 个空间，作为这个分程序段的第一条指令。下面就调用语句处理过程 Statement 分析语句。分析完成后，生成操作数为 0 的 OPR 指令，用于从分程序返回（对于 0 层的主程序来说，就是程序运行完成，退出）。

3.2.4 代码生成

对于源程序的每一个过程（包括主程序），在被调用时，首先在数据段中开辟三个空间，存放静态链 SL、动态链 DL 和返回地址 RA。静态链记录了定义该过程的直接外过程（或主程序）运行时最新数据段的基地址。动态链记录调用该过程前正在运行的过程的数据段基址。返回地址记录了调用该过程时程序运行的断点位置。对于主程序来说，SL、DL 和 RA 的值均置为 0。静态链的功能是在一个子过

程要引用它的直接或间接父过程（这里的父过程是按定义过程时的嵌套情况来定的，而不是按执行时的调用顺序定的）的变量时，可以通过静态链，跳过个数为层差的数据段，找到包含要引用的变量所在的数据段基址，然后通过偏移地址访问它。

在过程返回时，解释程序通过返回地址恢复指令指针的值到调用前的地址，通过当前段基址恢复数据段分配指针，通过动态链恢复局部段基址指针。实现子过程的返回。对于主程序来说，解释程序会遇到返回地址为 0 的情况，这时就认为程序运行结束。

解释程序过程中的 base 函数的功能，就是用于沿着静态链，向前查找相差指定层数的局部数据段基址。这在使用 sto、lod、stoArr、lodArr 等访问局部变量的指令中会经常用到。

类 PCODE 代码解释执行的部分通过循环和简单的 case 判断不同的指令，做出相应的动作。当遇到主程序中的返回指令时，指令指针会指到 0 位置，把这样一个条件作为终至循环的条件，保证程序运行可以正常的结束。

3.3 程序设计情况

3.3.1 主要成分描述

3.3.1.1 符号表

为了组成一条指令,编译程序必须知道其操作码及其参数(数或地址)。这 些值是由编译程序本身联系到相应标识符上去的。这种联系是在处理常数、变量 和过程说明完成的。为此，标识符表应包含每一标识符所联系的属性;如果标识 符被说明为常数，其属性值为常数值;如果标识符被说明成变量，其属性就是由 层次和修正量(偏移量)组成的地址;如果标识符被说明为过程,其属性就是过 程的入口地址及层次。

常数的值由程序正文提供，编译的任务就是确定存放该值的地址。我们选择顺序分配变量和代码的方法;每遇到一个变量说明，就将数据单元的下标加一(PL/0 机中,每个变量占一个存贮单元)。开始编译一个过程时,要对数据单元 的下标 dx 赋初值，表示新开辟一个数据区。dx 的初值为 3，因为每个数据区包 含三个内部变量 RA，DL 和 SL。

相关过程:enter(),该函数用于向符号表添加新的符号,并确定标识符的 有关属性。

3.3.1.2 运行时存储组织和管理

PL/0 处理机有两类存贮，目标代码放在一个固定的存贮数组 code 中，而所 需数据组织成一个栈形式存放。

PL/0 处理机的指令集根据 PL/0 语言的要求而设计，它包括以下的指令:

(1)LIT /* 将常数置于栈顶 */

(2)LOD /* 将变量值置于栈顶 */

(3)STO /* 将栈顶的值赋与某变量 */

(4)CAL /* 用于过程调用的指令 */

(5)INT /* 在数据栈中分配存贮空间 */

(6)JMP, JPC /* 用于 if, while 语句的条件或无条件控制转移指令 */

(7)OPR /* 一组算术或逻辑运算指令 */

上述指令的格式由三部分组成:

F L A

其中，f, l, a 的含义见下表:

F L A

INT 常 量

LIT 常 量

LOD 层次差 数据地址

STO 层次差 数据地址

CAL 层次差 程序地址

JMP 程序地址

JPC 程序地址

OPR 运算类别

表 2-1PL/0 处理机指令

上表中，层次差为变量名或过程名引用和声明之间的静态层次差别，程序地址为目标数组 code 的下标，数据地址为变量在局部存贮中的相对地址。 PL/0 的编译程序为每一条 PL/0 源程序的可执行语句生成后缀式目标代码。 这种代码生成方式对于表达式、赋值语句、过程调用等的翻译较简单。 PL/0 的每一个过程可能包含着局部变量,因为这些过程可以被递归地调用，故在实际调用前，无法为这些局部变量分配存贮地址。各个过程的数据区在存贮 栈 S 内顺序叠起来，每个过程，除用户定义的变量外，还摇篮有它自己的内部信 息，即调用它的程序段地址(返回地址)和它的调用者的数据区地址。在过程终 止后，为了恢复原来程序的执行，这两个地址都是必须的。我们可将这两个内部 值作为位于该过程数据区的内部式隐式局部变量。我们把它们分别称为返回地址 (return address)RA 和动态链(dynamic link)DL。动态链的头，即最新分 配的数据区的地址，保存在某地址寄存器 B 内。

因为实际的存贮分配是运行(解释)时进行的，编译程序不能为其生成的代码提供绝对地址,它只能确定变量在数据区内的位置,因此它只能提供相对地址。为了正确地存取数据，解释程序需将某个修正量加到相应的数据区的基地址上去。若变量是局部于当前正在解释的过程，则此基地址由寄存器 B 给出，否则，就需要顺着数据区的链逐层上去找。然而遗憾的是，编译程序只能知道存取路线表 2-2 if-while 语句目标代码生成模式 12 的表态长度，同时动态链保存的则是过程活动的动态历史，而这两条存取路线并不总是一样。

相关过程:base()，interpret()。其中 base()的功能是根据层次差并从当前数据区沿着静态链查找,以便获取变量实际所在的数据区其地址;interpret()则完成各种指令的执行工作。

3.4.1 语法分析方法

源程序采用递归下降的方法来设计 PL/0 编译器。以下是该语言的 FIRST 和 FOLLOW 集合。

非终结符(S) FIRST(S) FOLLOW(S)

const var procedure ident call

程序体 . ;

if begin while

语句 ident call begin if while then do

条件 odd + - (ident number then do

表达式 + - (ident number . ;) R end then do 项 ident number (. ;) R + - end then do 因子 ident number (. ;) R + - * / end then do 以下是语法分析程序的一般方法。

假定图 S 所对应的程序段为 T(S)，则:

(1) 用合适的替换将语法约化成尽可能少的单个图;

(2) 将每一个图按下面的规则(3)-(7)翻译成一个过程说明; (3) 顺序图对应复合语句:

S1 S2 Sn

对应:begin T(S1); T(S2); ...; T(Sn) end

(4) 选择:

S1

S2

S3

对应:case 语句或者条件语句:

case ch of if ch in L1 then T(S1) else

L1: T(S1); if ch in L2 then T(S2) else

L2: T(S2); 或 ...

... if ch in Ln then T(Sn) else Ln: T(Sn); error

其中 Li?FIRST(Si)，ch 为当前输入符号。(下同)

(5) 循环

S

对应:while ch in L do T(S)

(6) 表示另一个图 A 的图:

A

对应:if ch == x then read(ch) else error

3.5.1 中间代码表示

PL/0 的编译程序为每一条 PL/0 源程序的可执行语句生成后缀式目标代码。这种代码生成方式对于表达式、赋值语句、过程调用等的翻译较简单。 如赋值语句 X := Y op Z(op 为某个运算符)，将被翻译成下面的目标代码 序列:(设指令计数从第 100 号开始)

No. f L a

100 LOD Level_diff_Y Addr_Y

101 LOD Level_diff_Z Addr_Z

102 OPR op

103 STO Level_diff_X Addr_X 而对 if 和 while 语句稍繁琐一点，因为此时要生成一些跳转指令，而跳转 的目标地址大都是未知的。为解决这一问题，我们在 PL/0 编译程序中采用了回 填技术，即产生跳转目标地址不明确的指令时，先保留这些指令的地址(code 数组的下标)，等到目标地址明确后再回过来将该跳转指令的目标地址补上，使 其成为完整的指令。下表是 if、while 语句目标代码生成的模式。(L1，L2 是代 码地址)

if C then S While C do S

L1: 条件 C 的目标代码

条件 C 的目标代码

JPC - L2

JPC -- L1

语句 S 的目标代码

语句 S 的目标代码

JMP L1

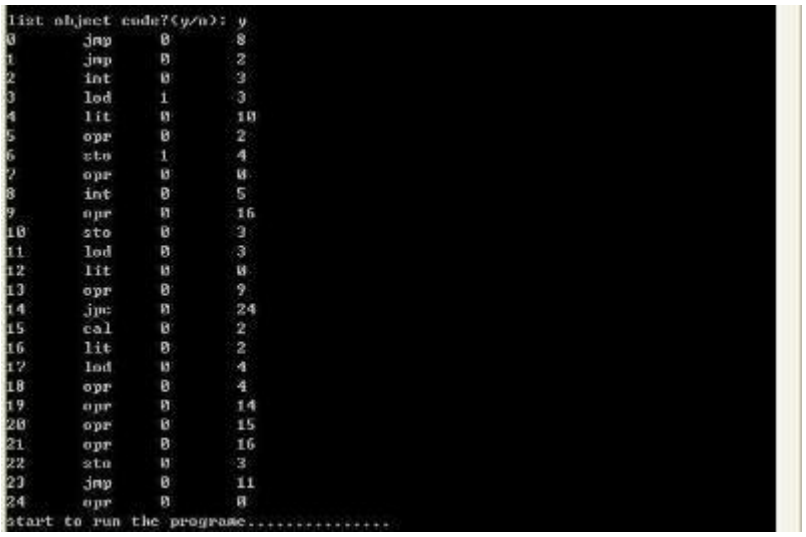
L1: ...

L2: ...

4. 运行结果

测试结果

上图显示测试代码，以及询问用户是否显示 object code,如按 y 则显示下图



5. 心得体会

经过近两周的努力，查资料，编译原理课程设计终于完成了。在这两周里，我学到了以前没有了解的知识，也知道了更多的技巧和方法，还深深地体会到编程来不得半点虚假，要踏踏实实地去理解程序的要求，要穷举问题的所有情况，也就是在写程序的过程中要小心,不要出现一些不必要的错误从而减少修改程序的时间，提高编程的效率。而且一旦决定了要做，就要坚持到底，千万不要因为一点小原因就半途而废，那样到头来只会是竹篮子打水，一场空。

总的来说，编译原理课程设计，难度挺大，加上本身时间比较重叠，所以很多问题都没有时间去尝试。而且也遇到像编译器存在某些小 BUG 的问题(比如 OPRO, 5),导致调试了很久，才发现问题不在自己的代码。最印象深刻的是对 FOR 语句的处理，因为 FOR 语句的语义分析比较繁现,老师虽然在课堂讲了两次，也听明白了，但是实际动手还是发现有比较大的困难。这也在一些方面体现 了知识的活学活用还是不够。

对于编译原理这门课，我觉得考志问题的分析方式还是很有意思，也对如何编译这个过程的处理有了一个大致的摸索。切实的进行编译器编程之后，对这个编译器如何实现我们的代码转换有了新的认识,站在这个基础上去理解高级语言

的语法分析的过程，去想象高级语言的语义分析的思路，对自己的编程感悟也有了-一些提高。

编译原理实验是一个要很注意细节的事情,因为在处理细节上必须注意,比如我在处理保留字数量和单词数量上就一直摸索了好久，期间也造成了-些像内存滋出,无法识别保留字的错误。而对于像 FOR 语句，主要是对于语义处理，和语句之间的衔按跳转回调的理解。

总之,编译原理是一-门需要细心和耐心的学科，对于我们理解高级语言的实现由很大的促进。

教师评定结果一览表

报告评定分别从整系统结构框架、各阶段设计思路和设计详细方法、程序运行结果、总结以及格式等进行打分，具体细则如表 1 所示。

表 1 评分标准表

评分点		分值	得分
1．整系统结构框架（共 20 分） 内容丰富，涵盖编译程序基本过程，条理清晰	系统结构框架介绍	10	
	设计思路	10	
2．各阶段设计思路和设计详细方法（共 40 分） 每个阶段的分析算法设计正确，程序代码编写规范，注释清楚，容易理解	各阶段设计思路	10	
	每个阶段的分析算法	15	
	程序代码编写	15	
3．程序运行结果（共 20 分）：调试和运行结果正确		20	
4．总结（共 10 分）：理顺全篇、描述准确、特点鲜明等		10	
5．格式（共 10 分）：符合要求、格式完整		10	
综合评价		总分： （100）	
教师 签名/日期			

执笔：舒畅　　审阅：　　审定：