DOI:10.16208/j.issn1000-7024.2005.08.044

PL/0 语言编译机制实现研究

宋　亮，　刘廷龙，　许　敏
（成都军区联勤部　指挥自动化工作站，四川　成都 610015）

摘　要：为了帮助加深对计算机语言编译技术的理解和应用，引述了 PL/0 语言的文法体系，简要介绍了 PL/0 语言编译程序的基本构成部分及其大体功能。通过跟踪 PL/0 语言编译程序对一段示例程序进行处理的内部过程，对构成语言编译技术的主要组成部分即词法分析、语法分析、中间代码生成、存储器分配、表格管理、错误处理等过程进行了重点分析，详细叙述了其工作原理与实现机制以及相互之间的关系。

关键词：PL/0 语言；　编译机制；　实现；　文法；　语法分析
中图法分类号：　TP314　　文献标识码：　A　　文章编号：　1000-7024 (2005) 08-2113-05

Analyses of implementation of PL/0's language compiler

SONG Liang,　LIU Ting-long,　XU Min
（Automatic Commanding Workstation, Joint Logistic Ministry, Chengdu Military Area, Chengdu 610015, China）

Abstract：It's helpfal to comprehend the principles and technologies of computer programming language's compilers. It's one of the most important baseand kernel of computer sciences. The grammar of PL/0 was firstly quoted, and then its primary composition and function respective were introduced. The main process and implement mechanism of PL/0's compiler is interpreted detailedly by using a piece of PL/0 code asexample, including morphology analyzing, parsing, internal code generating, memory managing, table managing, error disposing and so on.

Key words：PL/0;　code generation;　implement;　grammar;　parsing

1　引　言

编译技术既是计算机应用的基础和核心之一，也是计算机教育科学体系中的重点。然而读者普遍反映目前高校中开设的编译原理课程内容抽象、理论枯燥、难以理解，是公认的计算机课程中的难点。PL/0 语言是由瑞士籍计算机科学家 Niklaus Wirth 设计的一个微型计算机语言系统。它虽功能简单，但语法严谨，结构清晰，具备了一个高级语言编译程序所必需的基本要素和功能，因而在教学、科研领域得到了广泛应用。本文通过跟踪一段示例程序被编译的过程，深入研究 PL/0 语言编译程序的工作机制，帮助读者理解编译技术的基本原理和实现方法。本文采用的运行环境是 Borland Turbo Pascal 7.0，PL/0 语言编译程序采用由吕映芝等编著的《编译原理》[1] 提供的源程序（由于运行环境的不同，作者对代码做了少量修改）。

2　PL/0 语言概述

PL/0 语言的编译程序是一个编译解释执行系统，它的编译过程采用“一趟”扫描方式，PL/0 语言编译程序使用的是自顶向下的语法分析方法，生成的目标程序是与具体机器无关的假想栈式计算机的汇编语言。PL/0 语言的编译程序和目标程序的解释执行程序都是用 PASCAL 语言书写的。PL/0 语言可在配备 PASCAL 语言的任何机器上实现。

2.1　PL/0 语言文法体系

1：〈程序〉 ::=〈分程序〉 .

2：〈分程序〉 ::= [〈常量说明部分〉][〈变量说明部分〉][〈过程说明部分〉]〈语句〉

3：〈常量说明部分〉 ::=CONST〈常量定义〉 {,〈常量定义〉 }；

4：〈常量定义〉 ::=〈标识符〉 =〈无符号整数〉

5：〈无符号整数〉 ::=〈数字〉{〈数字〉}

6：〈变量说明部分〉 ::=V AR〈标识符〉 {,〈标识符〉 }；

7：〈标识符〉 ::=〈字母〉{〈字母〉|〈数字〉}

8：〈过程说明部分〉 ::=〈过程首部〉〈分程序〉{ ;〈过程说明部分〉}；

9：〈过程首部〉 ::=PROCEDURE〈标识符〉 ；

10：〈语句〉 ::=〈赋值语句〉 |〈条件语句〉 |〈当型循环语句〉 |〈过程调用语句〉 |〈读语句〉 |〈写语句〉 |〈复合语句〉 |〈空〉

11：〈赋值语句〉 ::=〈标识符〉 :=〈表达式〉

收稿日期：　2004-07-07。

作者简介：宋亮　(1978-)，男，四川绵阳人，硕士，研究方向为面向对象软件开发、复杂系统和数据库；刘廷龙　(1958-)，男，重庆武隆人，硕士，高级工程师，研究方向为分布式计算、网格计算和大型数据库；许敏　(1963-)，男，云南昆明人，硕士，工程师，研究方向为计算机安全、网络系统和软件工程。

12 :〈复合语句〉 ::=BEGIN〈语句〉{ ;〈语句〉 } END

13 :〈条件〉 ::=〈表达式〉〈关系运算符〉 〈表达式〉 |ODD〈表达式〉

14 :〈表达式〉 ::= [+|-]〈项〉{〈加法运算符〉 〈项〉}

15 :〈项〉::=〈因子〉{〈乘法运算符〉 〈因子〉}

16 :〈因子〉 ::=〈标识符〉 |〈无符号整数〉 |（ ’〈表达式〉‘ ） ’

17 :〈加法运算符〉 ::= + | -

18 :〈乘法运算符〉 ::= * |/

19 :〈关系运算符〉 ::= = | # |< | <= | > | >=

20 :〈条件语句〉 ::=IF〈条件〉 THEN〈语句〉

21 :〈过程调用语句〉 ::=CALL〈标识符〉

22 :〈当型循环语句〉 ::=WHILE〈条件〉 DO〈语句〉

23 :〈读语句〉 ::=READ ‘（ ’〈标识符〉 {, 〈标识符〉 }‘ ） ’

24 :〈写语句〉 ::=WRITE ‘（ ’〈表达式〉 { , 〈表达式〉 }‘ ） ’

25 :〈字母〉 ::= a | b | ... | X | Y | Z

26 :〈数字〉 ::=0 | 1 | 2 | ... |8 | 9

2.2 PL/0 语言编译程序的基本构成

PL/0 语言编译程序仅由 17 个过程和 2 个函数构成, 非常简洁精悍。它们的名字和功能简介如表 1 所示。

表 1 PL/0 语言编译程序组成过程或函数简介

过程或函数名	功能简介
pl0	主程序
error	出错处理, 打印出错位置和错误编码
getsym	词法分析, 读取一个单词
getch	漏掉空格, 读取一个字符
gen	生成目标代码, 并送入目标程序区
test	测试当前单词符号是否合法
block	分程序分析处理过程
enter	登录名字表
position（函数）	查找标识符在名字表中的位置
constdeclaration	常量定义处理
vardeclaration	变量声明处理
listcode	列出目标代码清单
statement	语句部分处理
expression	表达式处理
term	项处理
factor	因子处理
condition	条件处理
interpret	对目标代码的解释执行程序
base(函数）	通过静态链求出数据区的基地址

3 一段 PL/0 语言示例程序

本文将以一段示例程序为例, 来阐述 PL/0 语言编译程序的工作机制。 该示例程序是为了分析方便而经过极端简化的, 但它已经具备了一个合法的 PL/0 语言程序必需的基本组成部分。示例程序如下所示。

```
1：const a=5; (* 常量说明部分 *)
2：var b; (* 变量说明部分 *)
   (* 过程说明部分 （3~5）*)
3：procedure c;
4:  begin
5:  end;
6： b:=a+2.（* 语句 *）
```

4 PL/0 语言编译过程

4.1 构建编译各阶段所需的信息表

在 PL/0 语言编译程序中, 主过程 pl0 首先建立 4 个数组类型的变量 word、wsym、ssym、mnemonic 和 3 个集合类型的变量 declbegsys、statbegsys、facbegsys, 它们的任务就是存储编译过程中需要使用的各种信息。其功能简介如下所示。

word :存储 PL/0 语言的 13 个保留字;

wsym :与 word 数组一起判别各保留字的类别;

ssym :用于识别 PL/0 语言中的各种符号 （如 +、-、*、/ 等）;

mnemonic :存储 PL/0 语言中 8 条目标指令的助记符, 用于输出目标代码;

declbegsys、statbegsys、facbegsys :根据相应文法, 判断所处的语法结构。

这 7 个变量构成了编译过程中必需的 “ 信息字典 ” 。下面我们介绍其中较为特殊的 word 数组的结构和工作原理, 其余 6 个变量功能类似且相对简单, 读者可以自行分析。

数组 word 包含 PL/0 语言的 13 个保留字。 PL/0 语言编译程序在对源程序进行词法分析的时候, 会将每个读入的单词与该数组中的元素进行比较, 以确定其是否为保留字。数组 word 的构成如图 1 所示。

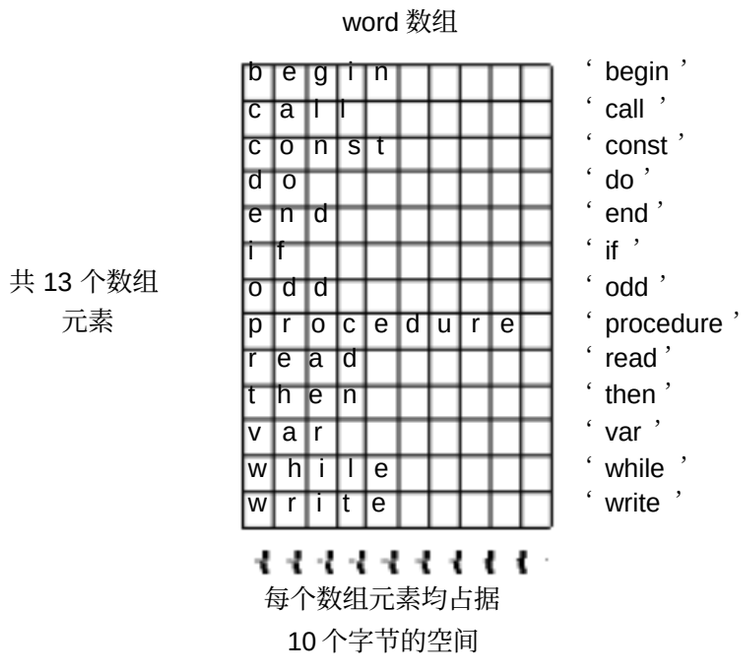

图 1 word 数组结构

细心的读者会发现, 该数组所包含的元素不论其内容是什么, 一律都是 10 个字符长, 不够 10 个字符长的单词用空格补齐。尽管如此之小的空间 “ 浪费 ” 完全可忽略不计, 但这究竟是为什么呢？因为 PL/0 语言编译程序在进行词法分析的过程中, 要扫描识别大量的字符流源程序, 并且每个读入的单词都要与该数组中的元素进行比较, 如果按照传统的逐个字符进行比较的方法, 其效率将是非常低下的。因此将单词元素统一设为 10 个字符长, 便于运用 “ 二分法 ” 进行比较以提高效率。关于使用 “ 二分法 ” 识别单词的程序片段如下所示。

```
（1） id := a;
（2） i := 1;
（3） j := norw;
```

```
(4)   repeat
(5)     k := (i + j) div 2;
(6)     if id <= word[k] then
(7)       j := k - 1;
(8)     if id >= word[k] then
(9)       i := k + 1
(10)  until i > j;
(11)  if i - 1 > j then
(12)    sym := wsym[k]
(13)  else
(14)    sym := ident
(15)  end
```

4.2 读入第 1 个单词

在建立好“信息字典”之后，PL/0 编译程序的主过程 pl0 调用 getsym 过程读入源程序的第 1 个有效单词（这里所指的有效单词不包括源程序中的空格、换行符和注释等与编译过程无关的内容）。PL/0 语言的有效单词分为 5 类。

保留字：如常见的 procedure、begin、end 等；

用户自定义标识符：如用户自定义的常量、变量名、过程名等；

常数：如 1, 5, 23 等无符号整数；

运算符：如+、-、*、/、<、>、=、and、or 等；

界符：如 ", "、" ;"、"." 、"(" 、")" 等。

getsym 过程的实质就是完成编译过程中的词法分析工作。它每被调用一次，就从源程序文件中读入一些字符，直到识别出一个单词为止，并判断其各种属性，包括类别、层次、值等，然后存入相应的全局变量中，便于在后续的编译过程中使用。而 getsym 过程的扫描功能又是通过反复调用 getch 过程实现的。getch 过程负责每次从源程序中读入一个字符。在读入源程序中第 1 个单词的时候，为了提高效率，getch 过程会首先在内存中开辟一个字符型数组 line 作为缓冲区，长度为 81，用来存放一行源代码（由此可见，在使用 PL/0 语言编制程序时，每行源代码不能超过 81 个字符长），然后把数组 line 的第 1 个字符赋给全局变量 ch 供 getsym 过程分析。getsym 过程能够根据每个单词的第 1 个字符判断其大体类型，然后选择进入其相应的分支语句做进一步处理，以获取更具体的信息。就像我们平时使用的电话，由于每个城市电话号码的组成规则不同：北京是 010，上海是 021，广州是 020……因此，程控交换机只需要读入电话号码的前 3 位，就可以判断被叫用户属于哪座城市。getsym 过程也正是根据这样的原理来对正在读入的单词的类别进行判别的。

以示例程序为例，其第 1 行代码是“const a=5;”。因此，pl0 在调用 getsym 过程之后，数组 line 中的内容就成为 ('c', 'o', 'n', 's', 't', ' ', 'a', '=', '5',';')，全局变量 ch 的值为 "c"。由 PL/0 语言文法可知，对于每种类型的单词来讲，其构成方式都有不同的定义。例如，保留字和用户自定义标识符的构成必须符合以下规则：[illegible]在首字符之后，变量名可以为：数字（0～9）和字母；[illegible]

[illegible])={[illegible] *[illegible] [illegible] [illegible] [illegible] [illegible] *}

因此，可以计算

FIRST（常量声明）={const}

FIRST（变量声明）={var}

FIRST（过程声明）={procedure}

FIRST（语句）={ident, begin, call, if, while, read, write}

并且，上下文无关文法要求各非终结符的开始符号集不相交，即

FIRST（常量声明）∩FIRST（变量声明）∩FIRST（过程声明）∩FIRST（语句）=[illegible]

·?3ìD[illegible]

const〈常量定义〉[变量说明][过程说明]语句.

相应的语法推导树如图 2 所示。

因此，语法分析过程 block 在调用 getsym 读入“const”的下

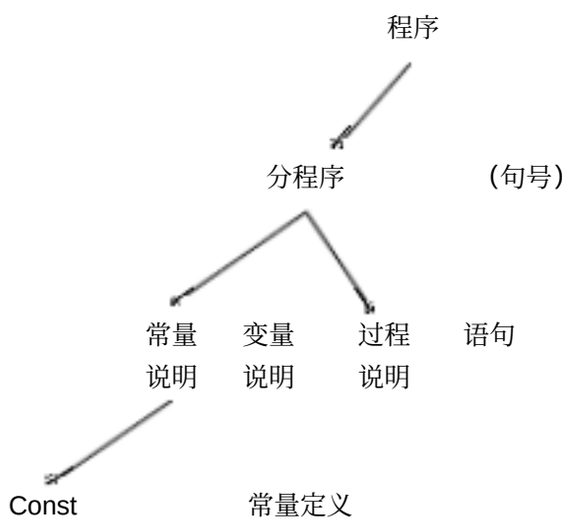

图 2　由“ const ”推导出的语法树

一个单词 “ a”之后， 紧接着调用常量声明分析过程 constdeclaration 进行分析， 过程源代码如下所示。

```
procedure constdeclaration;
begin
   if sym = ident then
   begin
      getsym;
      if sym in [eql, becomes] then
      begin
         if sym = becomes then
            error(1);
         getsym;
         if sym = number then
         begin
            enter(constant);
            getsym
         end
         else
            error(2)
      end
      else
         error(3)
      end
      else
         error(4)
   end;
```

有过 Pascal 程序开发经验的人都知道， 紧跟“ const ”关键字之后出现的必定是开发者定义的常量名。 因此，constdeclaration 首先判断紧接着 “ const ”出现的单词是否为一个标识符。这里最新读入的单词“ a”是一个合法标识符。因此再依次读入“ =”和常量值 “ 5”，然后调用 enter 过程， 将这个常量记录到编译程序开始声明的记录型数组 table 中。记录的内容包括常量 名称、类型 和值。 block 过 程还可 以通 过反 复调 用 constdeclaration 过程把所有以逗号 “ , ”分隔的常量声明全部登录到 table 中，直到出现 “ ;”——表示常量声明部分结束为止。 其代码如下所示。

```
while sym = comma do
   begin
      getsym;
      constdeclaration
   end;
if sym = semicolon then (* 直到出现 “ ;”, 标志常量声明部分的结束 *)
   getsym
else
   error(5)
until sym <> ident
```

处理完常量声明部分之后， 接下来执行变量声明的处理。将变量声明部分声明的变量录入 table 中的方法与常量声明部分相似， 区别是不但要填入变量名和类别， 还要填入变量声明所在的层次和地址；在处理过程声明部分时，开始仅将过程名、类别、层次和该过程数据区所占的大小等信息录入 table 中，然而一条完整的关于过程的记录还包括该过程的地址信息，这就必须等到编译程序执行完全部程序的语法分析工作之后， 利用“ 回填 ” 技术才能确定。因此， 在 PL/0 编译程序分析完示例程序的前 3 行代码后， table 中的内容如表 2 所示。

表 2　说明部分分析产生的元素

Name	Kind	Val	Level	Adr	Size
a	Constant	5	0	-	-
b	Variable	-	0	3	-
c	Procedure	-	0	未定	3

如果没有出现错误， 对示例程序中说明部分的语法分析到此就结束了。

4.4　目标代码的产生

处理完示例程序的声明部分后， 接下来要做的就是调用目标代码生成过程产生与源代码功能等价的目标代码。这里必须强调： 目标代码是在编译程序分析程序体中各种语句时产生的， 在处理说明部分时并不生成目标代码。 PL/0 语言合法的语句有： 赋值语句、 条件语句、 当型循环语句、 过程调用语句、读语句、 写语句、 复合语句和空语句。生成目标代码的工作主要是由 statement 过程完成的，该过程通过反复使用 if..then...else 控制语句， 将对各种句型的处理转入相应的分支流程中。每个分支对应一种语句处理功能的实现。下面我们就以示例程序中的第 6 行代码——一条赋值语句为例，详细阐述 PL/0 语言中间代码的产生过程。

产生赋值语句的目标代码的程序片断如下。

```
if sym=ident then
begin
   i := position (id);
   if i = 0 then
      error(11)
   else
   if table[i].kind <> variable then
   begin
      error(12);
      i := 0
   end;
```

```
    getsym;
    if sym = becomes then
        getsym
    else
        error(13);
    expression(fsys);
    if i <> 0 then
    with table[i] do
        gen(sto, lev-level, adr)
end
```

其工作流程如图 3 所示。

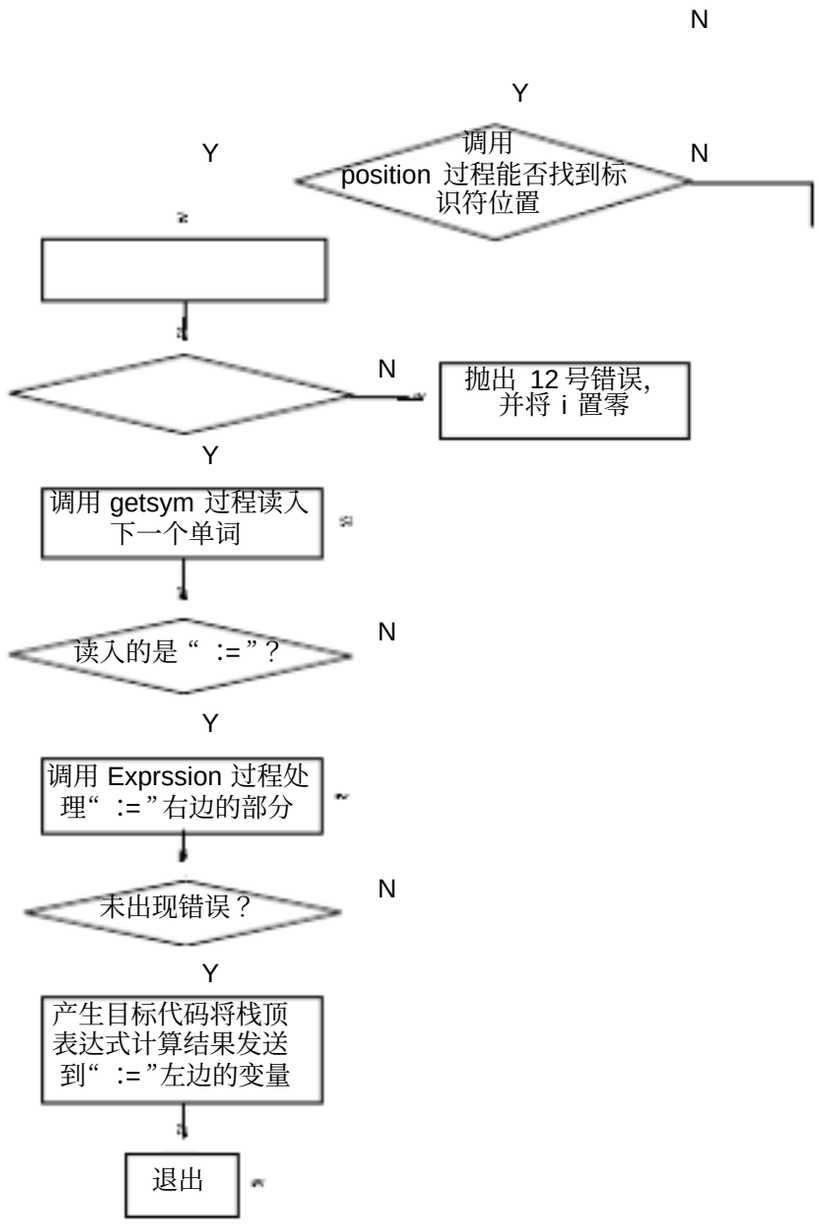

图 3 产生赋值语句目标代码的工作流程

赋值语句的构成形式是 “〈标识符〉 :=〈表达式〉 ”。因此，statement 过程首先判断语句的第 1 个单词是否为用户自定义标识符。如果不是，则退出本判断分支，测试下一个句型分支；如果是， 则调用 position 过程查找本标识符在 table 数组中的位置。 如果没有找到， 则抛出“ 标识符未声明 ” 的 11 号错误；如果找到， 则返回该标识符的地址， 并检查其是否为一个变量类型。若不是， 则抛出 12 号错误， 指出在赋值语句中， 赋值号左边标识符的属性应是变量； 若是，则判断紧跟标识符的单词是否为赋值符号 “ := ”。若不是， 抛出 13 号错误指明紧跟标识符的应为赋值符号； 若是， 则调用 expression 过程处理出现在“ := ”左边的表达式。 如果 expression 过程执行完毕并且没有发生任何错误， statement 过程就产生一条 STO 指令， 将存放在栈顶的表达式的结果发送到“ := ”左边的变量中。

expression 过程通过调用 term 过程处理构成表达式的项，而 term 过程又调用 factor 过程处理构成项的各因子。 factor 过程首先判断读入的单词是数还是标识符， 然后再根据单词的类型产生相应的指令。其代码如下：

```
    with table[i] do
    case kind of
        constant: gen(lit , 0 , val );(* 若标识符为常量， 则产生一条将其取至栈顶的指令 *)
        variable: gen(lod , lev-level , adr);(* 若标识符为变量， 则产生一条将该变量取至栈顶的指令 *)
        procedure: error(21);(* 如果出现在 “ :=”右部的是一个过程名， 则抛出 21 号错误， 指出在表达式内， 标识符属性不能是过程 *)
    end;
```

在示例程序中， 表达式的第 1 个单词是常量 a, 值为 5, 故编译程序产生一条 “ LIT 0 5”语句。 expression 过程在读入表达式中的 “ + ”后，因为“ +”是一个二目运算符， 所以 expression 过程必须再次调用 term 过程， 处理位于 “ + ”后的项。 这个项是常数“ 2 ”，所以 term 过程又以同样的方式调用 factor 过程来产生一条将常量值 “ 2 ”置于栈顶的目标代码 “ LIT 0 2 ”。这时，expression 过程才能产生将二者相加的指令 “ OPR 0 2”。这条语句执行的操作是将次栈顶的常量 a 与栈顶的 2 相加， 计算的结果存放在次栈顶， 同时将栈顶元素 “ 2 ”退栈， 这样结果所在的单元就变为栈顶。编译程序最后再产生一条指令 “ STO 0 3”，将栈顶所存结果 “ 7 ”送入变量 “ b ”所在单元 3，赋值语句翻译完毕。由此可见， 为了完成一条示例程序中赋值语句的功能，PL/0 语言编译程序要产生 4 条目标代码指令， 才能达到与之等价的效果。

最后， 编译程序读入代表程序结束的 “ . ”。如果代码生成过程没有出现任何错误， pl0 主过程就调用 interpret 过程解释执行生成的目标代码。 这个过程相对简单， 限于本文篇幅， 不再赘述， 读者可以自行分析。

5 结 论

深入研究 PL/0 语言编译程序的实现机制和方法， 结合编译原理课程中的相关理论知识， 再在加上读者积极、 创造性的实践， 可以极大地帮助我们理解和掌握计算机编译技术的基本内容， 这对于提高计算机应用水平和功底具有非常重要的作用。

参考文献 ：

[1] 吕映芝 ,张素琴 , 蒋维杜 . 编译原理 [M].北京 :清华大学出版社 , 1998.

[2] Alfred V Aho, Ravi Sethi, Jeffrey D Ullman. 编译原理 [M].北京:机械工业出版社 , 2003.

[3] Dick Grune, Henri E Bal, Ceriel J H Jacobs, et al. 现代编译程序设计 [M]. 北京 : 人民邮电出版社 , 2003.

[4] 钱能 . C++程序设计教程 [M]. 北京 :清华大学出版社 , 1999.

[5] 张尧学 ,史美林 . 计算机操作系统教程 [M].北京 :清华大学出版社, 1993.

[6] 殷人昆 , 陶永雷 , 谢若阳 , 等 . 数据结构 (用面向对象方法与 C++描述）[M].北京 :清华大学出版社 , 1999.