**实验报告:PL0语言编译器分析**

**一、实验目的**

通过阅读与解析一个实际编译器（PL/0语言编译器）的源代码， 加深对编译阶段（包括词法分析、语法分析、语义分析、中间代 码生成等）和编译系统软件结构的理解，并达到提高学生学习兴趣的目的。

**二、实验要求**

(1)要求掌握基本的程序设计技巧（C语言）和阅读较大规模程序 源代码的能力；

(2)理解并掌握编译过程的逻辑阶段及各逻辑阶段的功能；

(3)要求能把握整个系统（PL/0语言编译器）的体系结构，各功能 模块的功能，各模块之间的接口；

(4)要求能总结出实现编译过程各逻辑阶段功能采用的具体算法与技术。

**三、实验步骤**

(1) 根据PL/0语言的语法图，理解PL/0语言各级语法单位的结构，掌握PL/0语言合法程序的结构；

(2)从总体上分析整个系统的体系结构、各功能模块的功能、各模块之间的调用关系、各模块之间的接口；

(3)详细分析各子程序和函数的代码结构、程序流程、采用的主要算法及实现的功能；

(4)撰写分析报告，主要内容包括系统结构框图、模块接口、主要算法、各模块程序流程图等。

**四、报告内容**

　 pl/0语言是pascal语言的一个子集，我们这里分析的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。

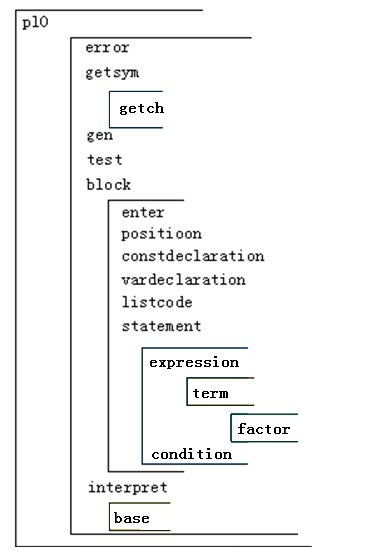
pl/0语言编译程序采用以语法分析为核心、一遍扫描的编译方法。词法分析和代码生成作为独立的子程序供语法分析程序调用。语法分析的同时，提供了出错报告和出错恢复的功能。在源程序没有错误编译通过的情况下，调用类pcode解释程序解释执行生成的类pcode代码。

PL/0语言文法的EBNF表示

EBNF表示的符号说明。

|  |  |
| --- | --- |
| 〈 〉 | 用左右尖括号括起来的中文字表示语法构造成分，或称语法单位，为非终结符。 |
| ∷= | 该符号的左部由右部定义，可读作“定义为”。 |
| | | 表示“或”，为左部可由多个右部定义。 |
| { } | 花括号表示其内的语法成分可以重复。在不加上下界时可重复0到任意次数，有上下界时为可重复次数的限制。 |

PL/0编译程序过程与函数定义层次结构图



PL/0的解释执行结构

|  |
| --- |
| PL/0语言解释执行程序 |

|  |
| --- |
| PL/0语言目标程序 |

|  |
| --- |
| 输入数据 |

|  |
| --- |
| 输出数据 |

PL/0编译程序结构

词法分析程序

语法语义分析程序

代码生成程序

表格管理程序

出错处理程序

PL/0源程序

目标程序

编译程序总体流程图



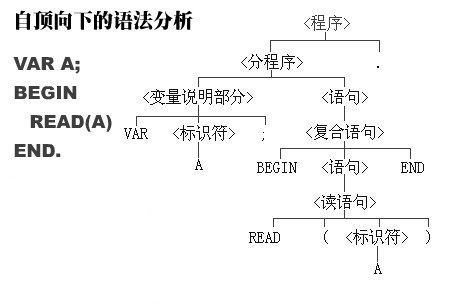
PL/0编译程序的语法分析

　　PL/0编译程序语法、语义分析是整个编译程序设计与实现的核心部分,要求学员努力学习掌握实现技术和方法。现分别说明语法分析实现的主要思想方法和语义分析的实现。

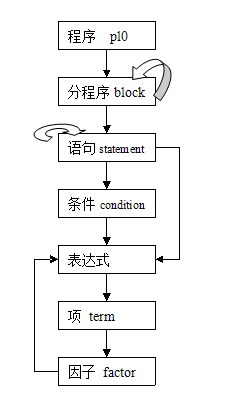
　　语法分析的任务是识别由词法分析给出的单词符号序列在结构上是否符合给定的文法规则。PL/0语言的文法规则已在2.1节中给出。本节将以语法图描述的语法形式为依据，给出语法分析过程的直观思想。

　　PL/0编译程序的语法分析采用了自顶向下的递归子程序法。

可用下面简单的PL/0程序为例构造其语法分析树



语法调用关系图



PL/0编译程序语法分析的设计与实现

〈表达式〉的递归子程序实现

procedure expr;

begin

if sym in [ plus, minus ] then

begin getsym; term;

end

else term;

while sym in [plus, minus] do

begin

getsym; term;

end

end;

〈因子〉∷=〈标识符〉|〈无符号整数〉|‘（’〈表达式〉‘）’

〈因子〉的递归子程序实现

procedure factor;

begin

if sym <> ident then

begin

if sym <> number then

begin

if sym = ‘(‘ then

begin

getsym;

expr;

if sym = ‘)’ then

getsym

else error

end

else error

end

else getsym

end

else getsym

end;

说明部分的分析与处理

对每个过程（含主程序）说明的对象（变量，常量和过程）造符号表, 登录标识符的属性。标识符的属性:种类，所在层次,值和分配的相对位置。登录信息由ENTER过程完成。

常量定义语句的处理语法：<常量说明部分>: := const <常量定义>{， <常量定义>}；

<常量定义>: := <标识符>=<无符号整数>

<无符号整数>: := <数字>{ <数字>}

if sym = constsym then

begin

getsym; (\* 获取下一个token，正常应为用作常量名的标识符 \*)

repeat (\* 反复进行常量声明 \*)

constdeclaration; (\* 声明以当前token为标识符的常量 \*)

while sym = comma do (\* 如果遇到了逗号则反复声明下一常量\*)

begin

getsym; (\* 获取下一个token，这里正好应该是标识符 \*)

constdeclaration (\* 声明以当前token为标识符的常量 \*)

end;

if sym = semicolon then (\* 如果常量声明结束，应遇到分号 \*)

getsym (\* 获取下一个token，为下一轮循环做好准备 \*)

else

error(5) (\*提示5号错误 \*)

until sym <> ident (\* 如果遇到非标识符，则常量声明结束 \*)

end;

常量说明处理

procedure constdeclaration;

begin

if sym = ident then

begin

getsym;

if sym in [eql, becomes] then (\* 如果是等号或赋值号 \*)

if sym = becomes then (\* 如果是赋值号(常量生明中应该是等号) \*)

error(1); (\* 提示1号错误 \*)

getsym; (\* 获取下一个token，等号或赋值号后应接上数字 \*)

if sym = number then (\* 如果的确是数字 \*)

begin

enter(constant); (\* 把这个常量登陆到符号表 \*)

getsym (\* 获取下一个token，为后面作准备 \*)

end

else error(2) (\* 如果等号后接的不是数字，提示2号错误 \*)

else error(3)(\* 如常量标识符后不是等号或赋值号，提示3号错误 \*)

end

else error(4)

end(\* constdeclaration \*);

变量定义语句的处理语法：<变量说明部分>: := var <标识符>{， <标识符>}；

if sym=varsym then

begin

getsym;

repeat

vardeclaration;(\*变量说明处理\*)

while sym=comma do

begin

getsym;

vardeclaration

end;

if sym=semicolon then

getsym

else error(5)

until sym<>ident;

end;

变量说明处理procedure ardeclaration;

begin

if sym=ident then

begin

enter(variable);

getsym

end

else error(4)

end(\*vardeclaration\*);

过程定义语句的处理程序：

while sym = procsym do (\* 循环声明各子过程 \*)

begin

getsym; (\* 获取下一个token，此处正常应为作为过程名的标识符 \*)

if sym = ident then (\* 如果token确为标识符 \*)

begin

enter(procedur); (\* 把这个过程登录到名字表中 \*)

getsym (\* 获取下一个token，正常情况应为分号 \*)

end

else

error(4); (\* 否则提示4号错误 \*)

if sym = semicolon then (\* 如果当前token为分号 \*)

getsym (\* 获取下一个token，准备进行语法分析的递归调用 \*)

else

error(5); (\* 否则提示5号错误 \*)

pl/0语言是pascal语言的一个子集，我们这里分析的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。

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（token），把它的信息放入全局变量sym、id和num中，语法分析器需要单词时，直接从这三个变量中获得。（注意！语法分析器每次用完这三个变量的值就立即调用getsym子程序获取新的单词供下一次使用。而不是在需要新单词时才调用getsym过程。）getsym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，然后把sym置为number，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。

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意生成相应的代码，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）、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。

语法单元分析：

1、分程序处理过程：

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）。

2、常量定义过程：

通过循环，反复获得标识符和对应的值，存入符号表。符号表中记录下标识符的名字和它对应的值。

3、变量定义过程：

与常量定义类似，通过循环，反复获得标识符，存入符号表。符号表中记录下标识符的名字、它所在的层及它在所在层中的偏移地址。

4、语句处理过程：

语句处理过程是一个嵌套子程序，通过调用表达式处理、项处理、因子处理等过程及递归调用自己来实现对语句的分析。语句处理过程可以识别的语句包括赋值语句、read语句、write语句、call语句、if语句、while语句。当遇到begin/end语句时，就递归调用自己来分析。分析的同时生成相应的类pcode指令。

5、赋值语句的处理：

首先获取赋值号左边的标识符，从符号表中找到它的信息，并确认这个标识符确为变量名。然后通过调用表达式处理过程算得赋值号右部的表达式的值并生成相应的指令保证这个值放在运行期的数据栈顶。最后通过前面查到的左部变量的位置信息，生成相应的sto指令，把栈顶值存入指定的变量的空间，实现了赋值操作。

6、read语句的处理：

　　确定read语句语法合理的前提下（否则报错），生成相应的指令：第一条是16号操作的opr指令，实现从标准输入设备上读一个整数值，放在数据栈顶。第二条是sto指令，把栈顶的值存入read语句括号中的变量所在的单元。

7、write语句的处理：

与read语句相似。在语法正确的前提下，生成指令：通过循环调用表达式处理过程分析write语句括号中的每一个表达式，生成相应指令保证把表达式的值算出并放到数据栈顶并生成14号操作的opr指令，输出表达式的值。最后生成15号操作的opr指令输出一个换行。

8、call语句的处理：

从符号表中找到call语句右部的标识符，获得其所在层次和偏移地址。然后生成相应的cal指令。至于调用子过程所需的保护现场等工作是由类pcode解释程序在解释执行cal指令时自动完成的。

9、if语句的处理：

按if语句的语法，首先调用逻辑表达式处理过程处理if语句的条件，把相应的真假值放到数据栈顶。接下去记录下代码段分配位置（即下面生成的jpc指令的位置），然后生成条件转移jpc指令（遇0或遇假转移），转移地址未知暂时填0。然后调用语句处理过程处理then语句后面的语句或语句块。then后的语句处理完后，当前代码段分配指针的位置就应该是上面的jpc指令的转移位置。通过前面记录下的jpc指令的位置，把它的跳转位置改成当前的代码段指针位置。

10、begin/end语句的处理：

通过循环遍历begin/end语句块中的每一个语句，通过递归调用语句分析过程分析并生成相应代码。

11、while语句的处理：

首先用cx1变量记下当前代码段分配位置，作为循环的开始位置。然后处理while语句中的条件表达式生成相应代码把结果放在数据栈顶，再用cx2变量记下当前位置，生成条件转移指令，转移位置未知，填0。通过递归调用语句分析过程分析do语句后的语句或语句块并生成相应代码。最后生成一条无条件跳转指令jmp，跳转到cx1所指位置，并把cx2所指的条件跳转指令的跳转位置改成当前代码段分配位置。

12、表达式、项、因子处理：

根据pl/0语法可知，表达式应该是由正负号或无符号开头、由若干个项以加减号连接而成。而项是由若干个因子以乘除号连接而成，因子则可能是一个标识符或一个数字，或是一个以括号括起来的子表达式。根据这样的结构，构造出相应的过程，递归调用就完成了表达式的处理。把项和因子独立开处理解决了加减号与乘除号的优先级问题。在这几个过程的反复调用中，始终传递fsys变量的值，保证可以在出错的情况下跳过出错的符号，使分析过程得以进行下去。

13、逻辑表达式的处理：

　　首先判断是否为一元逻辑表达式：判奇偶。如果是，则通过调用表达式处理过程分析计算表达式的值，然后生成判奇指令。如果不是，则肯定是二元逻辑运算符，通过调用表达式处理过程依次分析运算符左右两部分的值，放在栈顶的两个空间中，然后依不同的逻辑运算符，生成相应的逻辑判断指令，放入代码段。

14、判断单词合法性与出错恢复过程分析：

　　本过程有三个参数，s1、s2为两个符号集合，n为出错代码。本过程的功能是：测试当前符号（即sym变量中的值）是否在s1集合中，如果不在，就通过调用出错报告过程输出出错代码n，并放弃当前符号，通过词法分析过程获取一下单词，直到这个单词出现在s1或s2集合中为止。

　　这个过程在实际使用中很灵活，主要有两个用法：

　　在进入某个语法单位时，调用本过程，检查当前符号是否属于该语法单位的开始符号集合。若不属于，则滤去开始符号和后继符号集合外的所有符号。

　　在语法单位分析结束时，调用本过程，检查当前符号是否属于调用该语法单位时应有的后继符号集合。若不属于，则滤去后继符号和开始符号集合外的所有符号。

通过这样的机制，可以在源程序出现错误时，及时跳过出错的部分，保证语法分析可以继续下去。

15、类pcode代码解释执行过程分析：

这个过程模拟了一台可以运行类pcode指令的栈式计算机。它拥有一个栈式数据段用于存放运行期数据、拥有一个代码段用于存放类pcode程序代码。同时还拥用数据段分配指针、指令指针、指令寄存器、局部段基址指针等寄存器。

16、解释执行类pcode代码时，数据段存储分配方式：

　　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。