高等学校教材

编译程序的设计与实现

刘 磊 金 英 张 晶 张荷花 单 郸

高等教育出版社
HIGHER EDUCATION PRESS

内容提要

编译程序是计算机系统不可缺少的部分，是程序设计者的必备工具。学习并掌握编译程序的构造原理和实现技术，能够增强对程序设计语言的理解，提高程序设计、尤其是大型软件的设计能力。

本教材以一个简单的具有嵌套过程定义的过程式语言 SNL 作为教学语言，详细介绍了该语言编译程序的设计和实现方法，并对已经实现的编译程序的源代码分阶段进行了详细的分析，尤其是对编译程序的组成、实现算法、所用数据结构以及各功能部分所采用的编译技术都做了详细的介绍，并配有相应的框图说明。学生在学习“编译原理”课程的同时，可以配合本教材中编译实例的分析，进一步理解和掌握编译程序的构造原理和实现方法。此外，随书发行的光盘中含有 SNLC(SNL 编译程序)的安装程序、SNLC 的源代码以及 SNL 源程序实例。学生可阅读其中的编译程序源代码，并根据需要对源代码进行改进，从而达到加深对编译原理的理解、提高程序设计能力的目的。

本教材是一本非常实用的编译程序实例分析和教学辅导教材，可作为高等院校计算机及相关专业的本科教材，也可供相关技术人员参考。

图书在版编目(CIP)数据

编译程序的设计与实现/刘磊等. —北京：高等教育出版社，2004. 7

ISBN 7-04-014620-7

Ⅰ. 编… Ⅱ. 刘… Ⅲ. 编译程序-程序设计

Ⅳ. TP314

中国版本图书馆 CIP 数据核字（2004）第 052462 号

出版发行	高等教育出版社	购书热线	010-64054588
社　　址	北京市西城区德外大街 4 号	免费咨询	800-810-0598
邮政编码	100011	网　　址	http://www.hep.edu.cn
总　　机	010-82028899		http://www.hep.com.cn
经　　销	新华书店北京发行所		
印　　刷	北京市白帆印务有限公司		
开　　本	787×960　1/16	版　　次	2004 年 7 月第 1 版
印　　张	21	印　　次	2004 年 7 月第 1 次印刷
字　　数	380 000	定　　价	29.00 元(含光盘)

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

前　言

“编译原理”课程是计算机科学与技术专业最重要的专业课之一，掌握编译方法和技术是每一个优秀计算机软件专业人员的必备素质。对于计算机专业的学生来说，也许将来只有少部分人从事专业开发编译程序、维护编译程序的工作，但学好这门课程是十分重要的，主要体现在以下几个方面：

1. 学好“编译原理”课程可以加深对程序设计语言的理解，因为设计一个编译程序，需要准确认识程序语言的语法和语义，了解目标机及目标代码的结构，这些知识对于学习新的程序设计语言是非常有帮助的。

2. 因为编译程序本身是一个十分庞大而复杂的系统软件，涉及到许多复杂的数据结构和实现算法，若能系统全面地掌握编译技术，必将大大提高程序设计能力，特别是开发大型软件的能力。

3. 编译技术可以应用于许多实际的软件开发工作中，如软件开发平台、软件自动生成、模式匹配等。

4. “编译原理”课程的学习可以培养学生的抽象思维能力，掌握形式化描述技术，这种思想和方法可能对今后从事的软件开发工作产生深远的影响。

5. 编译程序是一种元级程序，即它处理的对象就是程序，因此学习编译原理和实现技术对于掌握元级程序设计方法十分有帮助。

许多学过“编译原理”课程的同学都会感到有些困惑：首先，觉得难度较大，不易掌握；其次，虽然掌握了编译的各个阶段的技术，但缺乏对编译程序的总体理解，各部分之间难以衔接。针对这些情况，我们编写了本教材。

本教材设计了简单的具有嵌套过程的程序设计语言（Small Nested Language，SNL）。该语言具有标准数据类型和结构数据类型，可以嵌套定义过程，过程的参数可以分为值参和变参两种形式，控制语句和 Pascal 语言基本相同，因此除指针类型外，SNL 具备了过程式语言的基本特征，以它作实例语言，构造其编译程序，可使得绝大多数编译技术在该编译程序中得以体现。

本教材介绍了 SNL 编译程序的实现方法，并对 SNL 编译程序的源代码分阶段进行了详细的分析，尤其是对编译程序的组成、数据结构、所用算法以及各功能部分所采用的编译技术都做了详细介绍，并配有框图说明。学生在学习“编译原理”课程的同时，可以通过本教材中编译实例的分析，进一步理解和掌握编译程序的构

造原理和实现方法,同时,通过阅读和改进本教材中提供的编译程序源代码,可以大大地提高程序设计能力。

本书分为十二章,每章的主要内容如下:

第一章:简单介绍编译程序的构造原理;

第二章:介绍本教材使用的类 Pascal 高级程序设计语言 SNL;

第三章:介绍 SNL 编译程序的总体结构和程序的组成;

第四章~第九章:分别介绍 SNL 编译程序的词法分析、语法分析、语义分析、中间代码生成、中间代码优化和目标代码生成部分的实现原理、具体的程序说明以及源程序清单;

第十章:介绍 SNL 编译程序的虚拟目标代码的解释程序的实现方法;

第十一章:总结前面介绍的 SNL 编译程序,并指出可以改进的方面和一些其他的应用实例;

第十二章:介绍 SNL 编译程序系统(Small Nested Language's Compiler,SNLC)的简明使用手册,以方便读者更好地使用 SNL 的编译程序系统。

本教材附光盘一张,包括 SNLC 的安装程序、SNLC 源代码以及 SNL 程序实例。其中,SNLC 可以按编译的过程分阶段执行,读者能够直观地获取编译各个阶段的执行结果,这有助于读者对教材中算法的理解。同时,读者也可参照 SNLC 的源程序自行设计语言的编译程序。

本书是作者根据多年教学实践经验总结而成的,已在吉林大学计算机科学与技术专业本科生教学中使用过多次,效果很好。北京工业大学计算机学院蒋宗礼教授认真审阅了本书的初稿,并提出了很多中肯的修改意见,吉林大学计算机学院金成植教授在本书的编写过程中也给予了多方面的指导。此外,吉林大学计算机学院软件自动化实验室的教师、博士生和硕士生们的帮助和建议也对本书的出版起了积极的作用。在此,作者向所有对本书的编写工作给予支持和帮助的人们表示衷心的感谢。由于编者水平有限,时间匆忙,书中难免存在一些缺点和不足,敬请读者多提宝贵意见,以便在适当的时间再做修订补充。

编 者

2004.5 于吉林大学

目　录

第一章　编译原理概述

1.1　高级程序设计语言的实现

众所周知，计算机硬件系统只能执行机器指令程序，而通常的应用程序都是用高级程序设计语言编写的。因此，要想使高级程序设计语言编写的程序能够被计算机识别并运行，必须有这样一种程序，它能够把用汇编语言或高级程序设计语言编写的程序转换成等价的机器语言程序，把这种转换程序统称为翻译程序(Translator)。其中，完成从高级程序设计语言编写的程序到等价的机器语言程序的转换任务的翻译程序称为编译程序(Compiler)，简称为编译器。

高级程序设计语言的实现通常有三种方式。

1. 编译方式

编译程序的输入是高级程序设计语言程序，称为源程序(Source Program)，输出是低级程序设计语言程序，称为目标程序(Target Program)，其功能如图 1.1 所示。

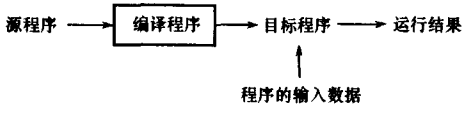

图 1.1　高级程序设计语言的编译实现方式

2. 解释方式

解释程序的输入是源程序和程序的输入数据，其方法是边翻译边执行，当翻译结束时，计算结果也会随之计算出来，其功能如图 1.2 所示。

图 1.2　高级程序设计语言的解释实现方式

3. 转换方式

假如要实现 L 语言，现在有 L′ 语言的编译程序，就可以先把 L 语言程序转换成 L′语言的程序，再利用 L′语言的编译程序实现 L 语言，其功能如图 1.3 所示。

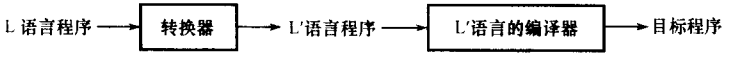

图 1.3　高级程序设计语言的转换实现方式

在这三种实现方式中，编译方式和解释方式最为常见。在编译方式下，源程序的执行是分阶段进行的。一般情况下，首先进行“翻译”，把用高级语言或汇编语言编写的程序翻译成与之等价的机器语言程序，然后再运行机器语言程序，最终得到运行结果。前一阶段的翻译工作由翻译程序（编译程序或汇编程序）来完成，后一阶段的运行计算需要有运行程序来配合完成。所谓运行程序是指运行目标代码程序时必须配置的各种子程序的全体，通常以库子程序的形式存在，如一些连接装配程序以及一些链接库等。在解释方式下，源程序的执行只有一个阶段——解释执行阶段。具体讲，完成解释工作的解释程序将按源程序中语句的动态顺序逐句地进行分析解释，并立即予以执行。解释方式和编译方式最根本的区别在于：在解释方式下，并不生成目标代码，而是直接执行源程序本身。

1.2　编译程序的组成

不同的编译程序都有各自不同的组织结构和实现方式，需要根据源语言和目标语言的特点及要求来确定编译程序的设计实现方案。因此，并没有一种固定的编译程序的程序结构，但功能结构几乎都是一致的。这里说的功能结构是指编译程序内部都做哪些工作，以及它们彼此之间的关系。图 1.4 说明了一般的编译程序的功能结构。

其中各部分完成的主要任务如下：

（1）词法分析：根据源语言的词法规则，扫描源程序的字母（ASCII 码）序列，并识别出一个一个具有独立意义的最小语法单位，即“单词”，同时确定该单词的种类（如标识符，界限符，常数，等等），并把每个单词的 ASCII 码序列替换为统一的标准形式——所谓的机内表示 TOKEN 形式——这种形式既刻画了单词本身，又刻画了它所具有的属性），同时词法分析还要完成词法错误的检查以及去掉注释等。词法分析阶段不依赖于语言的语法定义。

（2）语法分析：根据源语言的语法规则，逐一扫描源程序的 ASCII 码序列或词法分析后的 TOKEN 序列（对于前者，词法分析程序将作为语法分析程序的子程

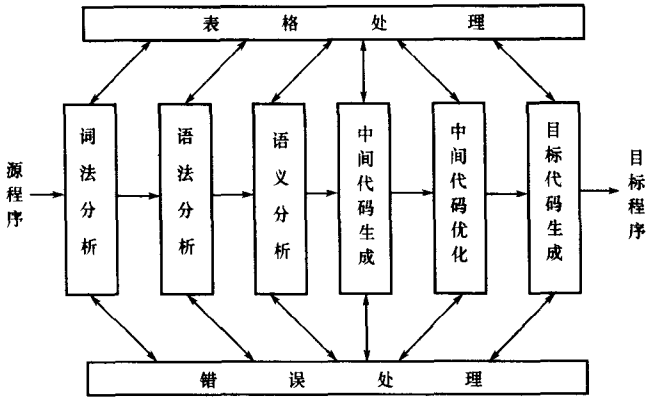

图 1.4 编译程序的功能构图

序),以确定源程序的具体组成结构(语法结构)。分析时如发现有不符合语法规则的地方,则打印出错位置和错误类型,以便程序员进行修改;如果未发现语法错误,则将源程序转换成能够表示程序结构的语法树的形式。很多编译程序在进行语法分析的同时还要完成其他的工作,但如果语法有错误,那么其他工作就没有意义了。

(3) 语义分析:根据源语言的语义规则对语法分析得到的语法树进行语义检查(确定类型,类型和运算的合法性检查等),并建立标识符的符号表等各种信息表。只有在没有语义错误的情况下才继续下面的编译过程,否则将不进行下面的编译工作。语义分为静态语义和动态语义,编译阶段一般只检查静态语义,而静态语义检查中重点是类型检查。

(4) 中间代码生成:扫描对象通常是语义分析后的结果,这一部分把源程序的TOKEN序列转换成更接近于目标代码的中间代码(如三元式或四元式的序列)。如果不做优化,这部分的工作就可以不要。

(5) 中间代码优化:扫描对象是中间代码,任务是把原中间代码转换成可产生高质量目标代码的中间代码,其中传统的优化工作包括常量表达式优化、公共子表达式优化、不变表达式外提和削减运算强度等。

(6) 目标代码生成:扫描对象是中间代码,任务是从中间代码产生目标代码。这一部分的工作与目标机紧密相连,其他部分的工作几乎与目标机无关。

(7) 错误处理:错误包括词法错误、语法错误、静态语义错误、动态语义错误。其中动态语义错误只有在运行目标程序时才能发现,在编译程序的各个阶段都要有错误处理部分,词法错误和语法错误检查集中一次完成,而语义检查则分散在以后的各个阶段在完成别的工作时顺便完成。

(8) 表格管理:较大的编译程序用到很多表格,甚至可以达几十种表。这些表将会占用大量存储区。因此,合理设计和使用表格是编译程序的一个重要问题。表格的分类、表的结构、表格的处理都是所要考虑的基本问题。有些表格以后可能无用,这时应及时删除它们,以扩大空区。如果处理不好表格的管理工作,就可能出现因表区溢出而不得不停止编译的现象,但这时可能还有空区。为了合理地管理表格(构造、查找、更新)和表区,不少编译程序都设立了专门的子程序(称为表格管理程序)负责管理表格。

1.3　编译程序的实现

编译程序是一个相当复杂的系统程序,通常有上万甚至几万条指令。随着编译技术的发展,编译程序的开发周期也在逐渐缩短,但仍然需要很多人年,而且工作很艰巨,正确性也不易保证。

要实现一个编译程序,通常需要做到以下几点:

- 对源语言的语法和语义要有准确无误的理解,否则难以保证编译程序的正确性;
- 对目标语言和编译技术也要有很好的了解,否则会生成质量不高的目标代码;
- 确定对编译程序的要求,如做不做优化、做优化做到哪一级等;
- 根据编译程序的规模,确定编译程序的扫描次数、每次扫描的具体任务和所要采用的技术;
- 设计各遍扫描程序的算法并加以实现。

一般开发编译程序有如下几种可能途径:

1. 转换法(预处理法)

假如要实现 L 语言的编译程序,现在有 L' 语言的编译程序,那么可以把 L 语言程序转换成 L' 语言的程序,再利用 L' 语言的编译程序实现 L 语言,这种方法通常用于语言的扩充。如对于 C++ 语言,可以把 C++ 程序转换成 C 程序,再应用 C 语言的编译程序进行编译,而不是重新设计和实现 C++ 编译程序。常见的宏定义和宏扩展都属于这种情形。

2. 移植法

假设在 A 机器上已有 L 语言的编译程序,想在 B 机器上开发一个 L 语言的编译程序。这里有两种实现方法:

(1) 最直接的办法就是将 A 机的代码直接转换成 B 机代码。

(2) 假设 A 机和 B 机上都有高级程序设计语言 W 的编译程序,并且 A 机上的 L 语言编译程序是用 W 语言写的,可以修改 L 编译程序的后端,即把从中间代码生成 A 机目标代码部分改为生成 B 机的目标代码。这种在 A 机上产生 B 机目标代码的编译程序称为交叉编译程序(Cross Compiler)。

3. 自展法

实现思想:先用目标机的汇编语言或机器语言书写源语言的一个子集的编译程序,然后再用这个子集作为书写语言,实现源语言的编译程序。通常这个过程会分成若干步,像滚雪球一样直到生成预计源语言的编译程序为止。这样的实现方式称为自展技术。

4. 工具法

20 世纪 70 年代随着诸多种类的高级程序设计语言的出现和软件开发自动化技术的提高,编译程序的构造工具陆续诞生,如 20 世纪 70 年代 Bell 试验室推出的 LEX、YACC 至今还在广泛使用。其中,LEX 是词法分析器的自动生成工具,YACC 是语法分析器的自动生成工具。然而,这些工具大都是用于编译程序的前端,即与目标机有关的代码生成和代码优化部分由于对语义和目标机形式化描述方面还存在困难,虽有不少生成工具被研制,但还没有广泛应用。

5. 自动生成法

如果能根据对编译程序的描述,由计算机自动生成编译程序,是最理想的方法,但需要对语言的语法、语义有较好的形式化描述工具,才能自动生成高质量的编译程序。目前,语法分析的自动生成工具比较成熟,如前面提到的 YACC 等,但是整个编译程序的自动生成技术还不是很成熟,虽然有基于属性文法的编译程序自动生成器和基于指称语义的编译程序自动生成器,但产生目标程序的效率很低,离实用尚有一段距离,因此,要想真正地实现自动化,必须建立形式化描述理论。

1.4 其他相关程序

当今的编译程序一般都不产生目标机的机器代码,而是产生汇编语言的目标代码。因此,要得到可执行的目标代码,还需要汇编程序的支持。汇编程序产生的是可重定位的机器代码,因此只有进行装配和连接之后方可执行。

综上所述,将一个高级程序设计语言的程序转换成可执行的机器代码一般包括以下过程:如图 1.5 所示。

(1) 预处理程序:把某个高级程序设计语言 L 的扩充转换成 L 的标准版本。预处理程序一般作为一个单独的程序,由编译程序在翻译之前调用。

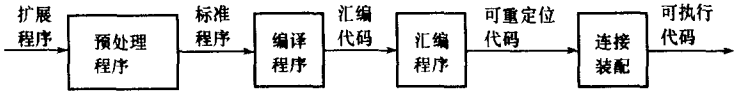

图 1.5　从扩展程序到可执行代码的过程

预处理程序又可以有以下几种类型：

① 宏处理器：宏可以看做是长结构的缩写；

② 文件包含：把头文件加入程序中；

③ 语言扩展：通过内置的宏，给语言增加新的功能。

（2）汇编程序：一个翻译程序，把汇编语言代码翻译成可重定位的机器代码。它的实现相对简单，大部分工作是一一对应地把符号化指令转换成相应的二进制码，把代表存储单元的每个标识符转换成相应的相对地址（偏移量）。

（3）连接程序：把若干个可重定位的机器代码文件（或编译后的目标代码/汇编后系统提供的库函数）合成一个可执行的目标代码文件。连接过程完全依赖于操作系统和处理器的细节。

（4）装配程序：把可重定位的目标代码中需要重定位的地址进行修改（通过与一个给定的基地址/始地址相加）。装配程序或在操作系统中或与连接程序共同合在一起，并不单独使用。

第二章　SNL 介绍

2.1　SNL 的特点

SNL(Small Nested Language)是自行定义的教学模型语言,它是一种类 Pascal 的"高级"程序设计语言。SNL 的数据结构比较丰富,除了整型、字符型等简单数据类型外,还有数组、记录等结构数据类型,过程允许嵌套定义,允许递归调用。SNL 基本上包含了高级程序设计语言的所有常用的成分,具备了高级程序设计语言的基本特征,实现 SNL 的编译程序,可以涉及到绝大多数编译技术。通过对 SNL 编译程序的学习,可以更加深入、更加全面地掌握编译程序的构造原理。但为了教学方便起见,略去了高级程序设计语言的一些复杂成分,如文件、集合、指针的操作等。

2.2　SNL 的词法

2.2.1　语言的字符表

程序是由字符组成的,每一种语言都对应一个字符表。SNL 的字符表定义如下:

<字符表> ::=a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q |
r | s | t | u | v | w | x | y | z |
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P
| Q | R | S | T | U | V | W | X | Y | Z |
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
+ | - | * | / | < | = | (|) | [|] | . | ; | EOF | 空白
字符 | { | } | ' |

注　在程序中,英文字母区分大小写;保留字只能由小写字母组成。

2.2.2　单词的巴科斯范式

SNL 编译系统的单词符号分类如下:

- 标识符　　(ID)
- 保留字　　(它是标识符的子集，if,repeat,read,write,...)
- 无符号整数　　(INTC)
- 单字符分界符　　(+,-,*,/,<,=,(,),[,],.,;,EOF,空白字符)
- 双字符分界符　　(:=)
- 注释头符　　({)
- 注释结束符　　(})
- 字符起始和结束符　　(')
- 数组下标界限符　　(..)

上述各类符号的巴科斯范式如下：

<标识符> ::= 字母 | 字母 | 数字 |

<无符号整数> ::= 数字 | 数字 |

<单字符分界符> ::= + | - | * | / | (|) | [|] | ; | . | < | = | EOF | 空白字符

<双字符分界符> ::= :=

<注释头符号> ::= {

<注释结束符号> ::= }

<字符标识符> ::= '

<数组下标界限符> ::= ..

<字母> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<数字> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

2.3 SNL 的语法

2.3.1 语法的非形式说明

一个 SNL 程序是由程序头、声明部分和程序体组成的。声明部分包括类型声明、变量声明和过程声明。SNL 的语法规定可以声明整型(integer)、字符类型(char)、数组类型以及记录类型的类型标识符和变量。过程声明包括过程头、过程内部声明和过程体部分，过程声明内部还可以嵌套声明内层过程。程序体由语句序列构成，语句包括空语句、赋值语句、条件语句、循环语句、输入/输出语句、过程调用语句和返回语句。表达式分为简单算术表达式和关系表达式。

(1) 程序头的形式是：关键字 program 后面跟着程序名标识符；

(2) 类型定义的形式是:类型名标识符 = 类型定义,其中类型定义可以是类型名或者是结构类型定义,类型名可以是基本类型,或者是前面已经定义的一个类型标识符;

(3) 变量声明的形式是:类型名后面跟着用逗号隔开的变量标识符序列;

(4) 过程声明的形式是:关键字 procedure 跟着过程名标识符以及参数声明、类型定义、变量说明、内层过程声明和程序体;

(5) 程序体的形式是:以关键字 begin 开头,关键字 end 结尾,中间是用分号隔开的语句序列(注意最后一条语句后不加分号)。如果是主程序,则在 end 后用“.”标志整个程序体的结束。

2.3.2 语法的形式定义

下面将用上下文无关文法给出 SNL 的语法。实际上应该把它看成是 TOKEN 化的语法,因为其中已省略了标识符和整型常数的产生式部分,而且关键字的 TOKEN 是用相应的大写表示的。

SNL 的上下文无关文法

总程序:
(1) Program ::= ProgramHead DeclarePart ProgramBody
程序头:
(2) ProgramHead ::= PROGRAM ProgramName
(3) ProgramName ::= ID
程序声明:
(4) DeclarePart ::= TypeDecpart VarDecpart ProcDecpart
类型声明:
(5) TypeDecpart ::= ε
(6) | TypeDec
(7) TypeDec ::= TYPE TypeDecList
(8) TypeDecList ::= TypeId = TypeDef;TypeDecMore
(9) TypeDecMore ::= ε
(10) | TypeDecList
(11) TypeId ::= ID
类型:
(12) TypeDef ::= BaseType
(13) | StructureType
(14) | ID
(15) BaseType ::= INTEGER
(16) | CHAR
(17) StructureType ::= ArrayType

(18)　　| RecType
(19) ArrayType　　::= ARRAY [low. . top] OF BaseType
(20) Low　　::= INTC
(21) Top　　::= INTC
(22) RecType　　::= RECORD FieldDecList END
(23) FieldDecList　　::= BaseType IdList; FieldDecMore
(24)　　| ArrayType IdList; FieldDecMore
(25) FieldDecMore　　::= ε
(26)　　| FieldDecList
(27) IdList　　::= ID IdMore
(28) IdMore　　::= ε
(29)　　|, IdList

变量声明:

(30) VarDecpart　　::= ε
(31)　　| VarDec
(32) VarDec　　::= VAR VarDecList
(33) VarDecList　　::= TypeDef VarIdList; VarDecMore
(34) VarDecMore　　::= ε
(35)　　| VarDecList
(36) VarIdList　　::= ID VarIdMore
(37) VarIdMore　　::= ε
(38)　　|, VarIdList

过程声明:

(39) ProcDecpart　　::= ε
(40)　　| ProcDec
(41) ProcDec　　::= PROCEDURE
　　ProcName (ParamList) ;
　　ProcDecPart
　　ProcBody
　　ProcDecMore
(42) ProcDecMore　　::= ε
(43)　　| ProcDeclaration
(44) ProcName　　::= ID

参数声明:

(45) ParamList　　::= ε
(46)　　| ParamDecList
(47) ParamDecList　　::= Param ParamMore
(48) ParamMore　　::= ε
(49)　　|; ParamDecList
(50) Param　　::= TypeDef FormList
(51)　　| VAR TypeDef FormList
(52) FormList　　::= ID FidMore