清华大学计算机系列教材

编　译　原　理

吕映芝　张素琴　蒋维杜　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

本书介绍编译系统的一般构造原理、基本实现技术和一些自动构造工具。主要由语言基础知识、词法分析、语法分析、中间代码生成、代码优化、目标代码生成、符号表的构造和运行时存储空间的组织等8部分组成。

书中在介绍编译程序构造基本原理的同时引入“PL/0语言的编译程序”结构及文本,还引入LEX、YACC使用方法与实例。

本书是高等院校计算机科学与技术专业的教材,也可作为教师、研究生或软件工程技术人员的参考书。

图书在版编目(CIP)数据

编译原理/吕映芝等编著.—北京:清华大学出版社,1998.1
(清华大学计算机系列教材)
ISBN 7-302-02732-3

Ⅰ.编… Ⅱ.吕… Ⅲ.编译程序-高等学校-教材 Ⅳ.TP314

中国版本图书馆CIP数据核字(97)第23938号

出 版 者:清华大学出版社(北京清华大学校内,邮编100084)
因特网地址:www.tup.tsinghua.edu.cn
责任编辑:贾仲良
印 刷 者:北京环球印刷厂
发 行 者:新华书店总店北京科技发行所
开 本:787×1092 1/16 印张:22.5 字数:531千字
版 次:1998年1月第1版 1998年11月第4次印刷
书 号:ISBN 7-302-02732-3/TP·1417
印 数:18001～23000
定 价:21.00元

前　言

本书介绍程序设计语言编译程序构造的一般原理、基本设计方法、主要实现技术和一些自动构造工具。为计算机科学和技术专业的本科生提供教材。

尽管“编译程序”是特指将高级程序设计语言翻译成低级语言的软件，但编译程序构造的基本原理和技术也广泛应用于一般软件的设计和实现，因此本书也是从事系统软件和软件工具研究及开发者的参考书。

本书首先从剖析一个简单的编译程序(PL/0)入手，对编译程序设计的基本理论，如有穷自动机、上下文无关文法等给予必要的介绍；对于广泛使用的语法分析方法和语义分析技术，如递归子程序法、算符优先分析、LR 分析及语法制导翻译等进行了详细的讲解；对编译程序的结构及其各部分功能、实现方法以及整体的设计考虑等给予了描述；此外还介绍了编译程序的构造工具。

“编译原理”是一门对实践性要求较高的课程，为此，我们拟定了教学实验要求，列在附录 D 中。实验所参考的 PL/0 编译程序文本列在附录 A 中。附录 B 和 C 是对编译程序构造工具 LEX 和 YACC 的介绍。为利于教学，我们自行研制了编译原理计算机辅助教学软件。附录 E 对该软件的功能和使用进行了说明。

本书的第 1 章、第 3 章、第 4 章、第 8 章、第 10 章和第 11 章由张素琴编写，第 2 章、第 5 章至第 7 章及第 12 章和第 13 章由吕映芝编写，第 9 章由蒋维杜编写。

书中如有不妥之处，请读者批评指正。

目　录

第 1 章　编译程序概论

1.1　什么是编译程序

编译程序是现代计算机系统的基本组成部分之一,而且多数计算机系统都含有不止一个高级语言的编译程序,对有些高级语言甚至配置了几个不同性能的编译程序。从功能上看,一个编译程序就是一个语言翻译程序。它把一种语言(称作源语言)书写的程序翻译成另一种语言(称作目标语言)的等价的程序。比如,汇编程序是一个翻译程序,它把汇编语言程序翻译成机器语言程序。如果源语言是像 FORTRAN, PASCAL 或 C 那样的高级语言,目标语言是像汇编语言或机器语言那样的低级语言,则这种翻译程序称作编译程序。如果把编译程序看成一个“黑盒子”,它所执行的转换工作可以用图 1.1 来说明。

图 1.1　编译程序的功能

一个编译程序的重要性体现在它使得多数计算机用户不必考虑与机器有关的繁琐细节,使程序员和程序设计专家独立于机器,这对于当今机器的数量和种类持续不断地增长的年代尤为重要。

使用过计算机的人都知道,除了编译程序外,还需要一些其它的程序才能生成一个可在计算机上执行的目标程序。让我们分析一下一个程序设计语言程序的典型的处理过程(见图 1.2),可以从中进一步了解编译程序的作用。

一个源程序有时可能分成几个模块存放在不同的文件里,将这些源程序汇集在一起的任务,由一个叫做预处理程序的程序来完成,有些预处理程序也负责宏展开,像 C 语言的预处理程序要完成文件合并、宏展开等任务。图 1.2 中的编译程序生成的目标程序是汇编代码形式,需要经由汇编程序翻译成可再装配的机器代码,再经由装配/连接编辑程序与某些库程序连接成真正能在机器上运行的代码。也就是说,一个编译程序的输入可能要由一个或多个

图 1.2　高级语言程序的处理过程

预处理程序来产生,另外,为得到能运行的机器代码,编译程序的输出可能仍需要进一步地处理。

前面介绍过,编译程序的基本任务是将源语言程序翻译成等价的目标语言程序。我们知道,源语言的种类成千上万,从常用的诸如 FORTRAN,PASCAL 和 C 语言,到各种各样的计算机应用领域的专用语言,而目标语言也是成千上万的,加上编译程序根据它们的构造不同,所执行的具体功能的差异又分成了各种类型,比如:一趟编译、多趟编译的、具有调试或优化功能的等等。尽管存在这些明显的复杂因素,但是任何编译程序所必须执行的主要任务基本是一样的,通过理解这些任务,使用同样的基本技术,我们可以为各种各样的源语言和目标语言设计和构造编译程序。

据说第一个编译程序的出现是在 20 世纪 50 年代早期,很难讲出确切的时间,因为当初大量的实验和实现工作是由不同的小组独立完成的,多数早期的编译工作是将算术公式翻译成机器代码。用现在的标准来衡量,当时的编译程序能完成的工作十分初步,如只允许简单的单目运算,数据元素的命名方式有很多限制。然而它们奠定了对高级语言编译系统的研究和开发的基础。20 世纪 50 年代中期出现了 FORTRAN 等一批高级语言,相应的一批编译系统开发成功。随着编译技术的发展和社会对编译程序需求的不断增长,20 世纪 50 年代末有人开始研究编译程序的自动生成工具,提出并研制编译程序的编译程序。它的功能是以任一语言的词法规则、语法规则和语义解释出发,自动产生该语言的编译程序。目前很多自动生成工具已广泛使用,如词法分析程序的生成系统 LEX,语法分析程序的生成系统 YACC 等。20 世纪 60 年代起,不断有人使用自展技术来构造编译程序。自展的主要特征是用被编译的语言来书写该语言自身的编译程序。1971 年,PASCAL 的编译程序用自展技术生成后,其影响就越来越大。

随着并行技术和并行语言的发展,处理并行语言的并行编译技术正在深入研究之中,将串行程序转换成并行程序的自动并行编译技术也正在深入研究之中。

1.2 编译过程概述

编译程序完成从源程序到目标程序的翻译工作,是一个复杂的整体的过程。从概念上来讲,一个编译程序的整个工作过程是划分成阶段进行的,每个阶段将源程序的一种表示形式转换成另一种表示形式,各个阶段进行的操作在逻辑上是紧密连接在一起的,图 1.3 给出了一个编译过程的各个阶段,这是一种典型的划分方法。事实上,某些阶段可能组合在一起,这些阶段间的源程序的中间表示形式就没必要构造出来了。图 1.3 中将编译过程划分成了词法分析、语法分析、语义分析、中间代码生成,代码优化和目标代码生成六个阶段,我们将分别介绍各阶段的任务。另外两个重要的工作:表格管理和出错处理与上述六个阶段都有联系。编译过程中源程序的各种信息被保留在种种不同的表格里,编译各阶段的工作都涉及到构造、查找或更新有关的表格,因此需要有表格管理的工作;如果编译过程中发现源程序有错误,编译程序应报告错误的性质和错误发生的地点,并且将错误所造成的影响限制在尽可能小的范围内,使得源程序的其余部分能继续被编译下去,有些编译程序还能自动校正错误,这些工作称之为出错处理。

我们从源程序在不同阶段所被转换成的表示形式的不同来介绍各个阶段的任务。

词法分析阶段　是编译过程的第一个阶段。这个阶段的任务是从左到右一个字符一个字符地读入源程序,对构成源程序的字符流进行扫描和分解,从而识别出一个个单词(也称单词符号或符号)。这里所谓的单词是指逻辑上紧密相连的一组字符,这些字符具有集体含义。比如标识符是由字母字符开头,后跟字母、数字字符的字符序列组成的一种单词。保留字(关键字或基本字)是一种单词,此外还有算符,界符等等。例如某源程序片断如下:

begin var sum, first, count: real; sum: = first + count * 10 end. 词法分析阶段将构成这段程序的字符组成了如下单词序列:

1. 保留字	begin	2. 保留字	var
3. 标识符	sum	4. 逗　号	,
5. 标识符	first	6. 逗　号	,
7. 标识符	count	8. 冒　号	:
9. 保留字	real	10. 分　号	;
11. 标识符	sum	12. 赋值号	: =
13. 标识符	first	14. 加　号	+
15. 标识符	count	16. 乘　号	*
17. 整　数	10	18. 保留字	end
19. 界　符	.		

图 1.3　编译的各个阶段

可以看出,五个字符即 b, e, g, i 和 n 构成了一个分类为保留字的单词 begin,两个字符即: 和= 构成了表示赋值运算的符号: = 。这些单词间的空格在词法分析阶段都被滤掉了。

我们使用 id1, id2 和 id3 分别表示 sum, first 和 count 三个标识符的内部形式,那么经过词法分析后上述程序片断中的赋值语句 sum : = first + count * 10 则表示为 id1: = id2 + id3 * 10

语法分析　是编译过程的第二个阶段。语法分析的任务是在词法分析的基础上将单词序列分解成各类语法短语,如“程序”,“语句”,“表达式”等等。一般这种语法短语,也称语法单位,可表示成语法树,比如上述程序段中的单词序列:

id1: = id2 + id3 * 10　经语法分析得知其是 PASCAL 语言的“赋值语句”,表示成如图 1.4 所示的语法树或是图 1.5 所示的那种形式。

语法分析所依据的是语言的语法规则,即描述程序结构的规则。通过语法分析确定整个输入串是否构成一个语法上正确的程序。

图 1.4　语句 id1: = id2+ id3* 10 的语法树

图 1.5　语句 id1: = id2+ id3* 10 的语法树的另一种形式

程序的结构通常是由递归规则表示的,例如,我们可以用下面的规则来定义表达式:

(1) 任何标识符是表达式。

(2) 任何常数(整常数、实常数)是表达式。

(3) 若表达式 1 和表达式 2 都是表达式,那么:表达式 1+ 表达式 2

表达式 1 * 表达式 2

(表达式 1)

都是表达式

类似地,语句也可以递归地定义,如

(1) 标识符: = 表达式　是语句。

(2) while　(表达式)　　do　语句和

if　　(表达式)　　then　语句 else　语句

都是语句。

上述赋值语句 id1: = id2+ id3* 10 之所以能表示成图 1.4 的语法树,依据的是赋值语句和表达式的定义规则。

词法分析和语法分析本质上都是对源程序的结构进行分析。但词法分析的任务仅对源程序进行线性扫描即可完成,比如识别标识符,因为标识符的结构是字母打头的字母和数字串,这只要顺序扫描输入流,遇到既不是字母又不是数字字符时,将前面所发现的所有字母和数字组合在一起而构成单词标识符。但这种线性扫描则不能用于识别递归定义的语法成分,比如就不能用此办法去匹配表达式中的括号。

语义分析阶段　是审查源程序有无语义错误,为代码生成阶段收集类型信息。比如语义分析的一个工作是进行类型审查,审查每个算符是否具有语言规范允许的运算对象,当不符合语言规范时,编译程序应报告错误。如有的编译程序要对实数用作数组下标的情况

报告错误。又比如某些语言规定运算对象可被强制,那么当二目运算施于一整型和一实型时,编译程序应将整型转换成实型而不能认为是源程序的错误,假如在语句 sum: = first + count* 10 中,* 的两个运算对象: count 是实型,10 是整型,则语义分析阶段进行类型审查之后,在语法分析所得到的分析树上增加一语义处理结点,表示整型变成实型的一目算符 inttoreal,则图 1.5 的树变成图 1.6 所示的那样。

图 1.6　插入语义处理结点的树

中间代码生成　在进行了上述的语法分析和语义分析阶段的工作之后,有的编译程序将源程序变成一种内部表示形式,这种内部表示形式叫做中间语言或中间代码。所谓“中间代码”是一种结构简单、含义明确的记号系统,这种记号系统可以设计为多种多样的形式,重要的设计原则为两点: 一是容易生成; 二是容易将它翻译成目标代码。很多编译程序采用了一种近似‘三地址指令’的‘四元式’中间代码,这种四元式的形式为: (运算符,运算对象 1,运算对象 2,结果)。

比如源程序　sum: = first+ count* 10 可生成四元式序列,如图 1.7 所示,其中 ti(i = 1, 2, 3) 是编译程序生成的临时名字,用于存放运算结果的。

(1)	(inttoreal	10	-	t_1)
(2)	(*	id3	t_1	t_2)
(3)	(+	id2	t_2	t_3)
(4)	(: =	t_3	-	id1)

图 1.7　中间代码

代码优化　在此阶段的任务是对前阶段产生的中间代码进行变换或进行改造,目的是使生成的目标代码更为高效,即省时间和省空间。比如图 1.7 的代码可变换为图 1.8 的代码,仅剩了两个四元式而执行同样的计算。也就是编译程序的这个阶段已经把将 10 转换成实型数的代码化简掉了,同时因为 t_3 仅仅用来将其值传递给 id1,也可以被化简掉,这只是优化工作的两个方面,此外诸如公共子表达式的删除、强度削弱、循环优化等优化工作将在第 11 章详细介绍。

(*	id3	10.0	t_1)
(+	id2	t_1	id1)

图 1.8　优化后的中间代码

目标代码生成　这一阶段的任务是把中间代码变换成特定机器上的绝对指令代码或可重定位的指令代码或汇编指令代码。这是编译的最后阶段,它的工作与硬件系统结构和指令含义有关,这个阶段的工作很复杂,涉及到硬件系统功能部件的运用、机器指令的选

择、各种数据类型变量的存储空间分配以及寄存器和后缓寄存器的调度等。

例如，使用两个寄存器(R_1 和 R_2)，可能生成如图 1.9 的某汇编代码。

```
(1)  MOVF   id3,   R2
(2)  MULF   # 10.0,  R2
(3)  MOVF   id2,   R1
(4)  ADDF   R1,    R2
(5)  MOV    R1,    id1
```

图 1.9　目标代码

第一条指令将 id3 的内容送至寄存器 R_2，第二条指令将其与实常数 10.0 相乘，这里用# 表明 10.0 处理为常数，第三条指令将 id2 移至寄存器 R_1，第四条指令加上前面计算出的 R_2 中的值，第五条指令将寄存器 R_1 的值移到 id1 的地址中。这些代码实现了本节开头给的源程序片断的赋值。

前面提到过上述编译过程的阶段划分是一典型处理模式，事实上并非所有的编译程序都分成这样几个阶段，有些编译程序并不要生成中间代码，有些编译程序不进行优化，优化阶段即可省去，有些最简单的编译程序在语法分析的同时产生目标指令代码，如第 2 章介绍的 PL/0 语言编译程序。不过多数实用的编译程序都采用上述几个阶段的工作过程。

1.3　编译程序的结构

上述编译过程的六个阶段的任务，再加上表格管理和出错处理的工作可分别由几个模块或程序完成，它们分别称作词法分析程序、语法分析程序、语义分析程序、中间代码生成程序、代码优化程序、目标代码生成程序、表格管理程序和出错处理程序。从而可给出一个典型的编译程序结构框图，如图 1.10 所示。

图 1.10　编译程序结构框图

1.4 编译阶段的组合

在第 1.2 节所讨论的编译过程中阶段的划分是编译程序的逻辑组织。有时，常常把编译的过程分为前端(front end)和后端(back end)，前端由那样一些阶段组成：这些阶段的工作主要依赖于源语言而与目标机无关。通常这些阶段包括词法分析、语法分析、语义分析和中间代码生成，某些优化工作也可在前端做，也包括与前端每个阶段相关的出错处理工作和符号表管理工作。后端工作指那些依赖于目标机而一般不依赖源语言，只与中间代码有关的那些阶段，即目标代码生成，以及相关出错处理和符号表操作。

若按照这种组合方式实现编译程序，可以设想，某一编译程序的前端加上相应不同的后端则可以为不同的机器构成同一个源语言的编译程序。也可以设想，不同语言编译的前端生成同一种中间语言，再使用一个共同的后端，则可为同一机器生成几个语言的编译程序。

一个编译过程可由一遍、两遍或多遍完成。所谓“遍”，也称作“趟”，是对源程序或其等价的中间语言程序从头到尾扫视并完成规定任务的过程。每一遍扫视可完成上述一个阶段或多个阶段的工作。例如一遍可以只完成词法分析工作；一遍完成词法分析和语法分析工作；甚至一遍完成整个编译工作。对于多遍的编译程序，第一遍的输入是用户书写的源程序，最后一遍的输出是目标语言程序，其余是上一遍的输出为下一遍的输入。在实际的编译系统的设计中，编译的几个阶段的工作究竟应该怎样组合，即编译程序究竟分成几遍，参考的因素主要是源语言和机器(目标机)的特征。比如源语言的结构直接影响编译的遍的划分；像 PL/1 或 ALGOL 68 那样的语言，允许名字的说明出现在名字的使用之后，那么在看到名字之前是不便为包含该名字的表达式生成代码的，这种语言的编译程序至少分成两遍才容易生成代码。另外机器的情况，即编译程序工作的环境也影响编译程序的遍数的划分。一个多遍的编译程序可以较之一遍的编译程序少占内存，遍数多一点，整个编译程序的逻辑结构可能清晰些，但遍数多即意味着增加读写中间文件的次数，势必消耗较多时间，显然会比一遍的编译要慢。

1.5 编译技术和软件工具

为了提高软件开发的效率和保证质量，人们除了要在软件工程中对软件开发过程所要遵循的规范化或标准化外，还尽量使用先进的软件开发技术和相应的软件工具，而大部分软件工具的开发，常常要用到编译技术和方法。实际上编译程序本身也是一种软件开发工具。为了提高编程效率，缩短调试时间，软件工作人员研制了不少对源程序处理的工具。这些工具的开发不同程度地用到编译程序各个部分的技术和方法。下面仅是一些例子。

1. 语言的结构化编辑器　结构化编辑器是引导用户在语言的语法制导下编制程序，能自动地提供关键字和与其匹配的关键字，如 if 后必须有 then，begin 和 end 的配对，左右括号的配对等，这样可以减少语法上的错误，可加快对源程序的调试，提高效率和质量。

2. 语言程序的调试工具　调试是软件开发过程中一个重要环节，结构化编辑器只能

解决语法错误的问题,而对一个已通过编译的程序来说,需进一步了解的是程序执行的结果与编程人员的意图是否一致,程序的执行是否实现预计的算法和功能。这种对算法的错误或程序没能反应算法的功能等错误就需用调试器来协助解决。调试器的功能愈强,实现愈复杂,但它必须与语法分析、语义处理有紧密联系。

3. 语言程序测试工具　语言程序的测试工具有两种:静态分析器和动态测试器。

静态分析器是对源程序进行静态地分析。它对源程序进行语法分析并制定相应表格,检查变量定值与引用的关系。如某变量未被赋值就被引用,或定值后未被引用,或多余的源代码等一些编译程序的语法分析发现不了的错误。

动态测试工具是在源程序的适当位置插入某些信息,并用测试用例记录(显示语句或函数)程序运行时的实际路径。将运行结果与期望的结果进行比较分析,帮助编程人员查找问题。这种测试工具在国内已有开发,如 FORTRAN 语言和 C 语言的测试工具。

4. 高级语言之间的转换工具　由于计算机硬件的不断更新换代,更新更好的程序设计语言的推出为提高计算机的使用效率提供了良好条件,然而一些已有的非常成熟的软件如何在新机器新语言情况下使用呢?为了减少重新编制程序所耗费的人力和时间,就要解决如何把一种高级语言转换成另一种高级语言,乃至汇编语言转换成高级语言的问题。这种转换工作要对被转换的语言进行词法和语法分析,只不过生成的目标语言是另一种高级语言而已。这与实现一个完整的编译程序相比工作量要少些。在国内已研制出 C,PASCAL,FORTRAN 到 Ada 的翻译器和 IBM 4700 汇编到 C 的转换器,其效果很好。

5. 并行编译技术　由于近几年并行机及多处理机的发展,对软件的并行处理技术提出了新的要求。特别是并行编译技术发展很快,目前处理并行编译技术有两种方法:

第一种方法,运用重构技术把已有的串行语言编写的程序经过相关分析,分解成可并行的成分,分配到多 CPU 或多处理机上运行,这种技术在国内已有 FORTRAN 和 C 语言的并行重构处理系统,相当成功。

第二种方法,即在程序设计语言机制上允许用户自己编写并行语言程序,这当然比编写串行语言对编程人员提出的要求更多。即用户自己必须知道程序各模块之间逻辑结构关系及调用关系乃至运算量,以确定哪些模块可以并行执行。若编程者能按程序设计情况编出并行程序,无疑并行程序效率将比第一种方法要好。

编译技术在软件工程的其它领域中也有广泛应用,本章不再介绍。

第 2 章　PL/0 编译程序的实现

为了使读者在系统地学习本教材以下各章节之前，对编译程序的构造得到一些感性认识和初步了解，本章推荐了世界著名计算机科学家 N.Wirth 先生编写的“PL/0 语言的编译程序”，并对其实现过程作了概括的分析说明，作为读者阅读 PL/0 语言编译程序文本的提示。对 PL/0 语言文法的表示只给出语法图和扩充的巴科斯-瑙尔范式(EBNF)的描述形式，不作文法理论上的讨论。由于 PL/0 语言功能简单、结构清晰、可读性强，而又具备了一般高级语言的必须部分，因而 PL/0 语言的编译程序能充分体现一个高级语言编译程序实现的基本技术和步骤。因此，“PL/0 语言编译程序”是一个非常合适的小型编译程序的教学模型。所以，阅读“PL/0 语言编译程序”文本后，可帮助读者对编译程序的实现建立起整体概念。

2.1　PL/0 语言描述

本节将用语法图和扩充的巴科斯-瑙尔范式(EBNF)两种形式给出 PL/0 语言的语法描述。

2.1.1　PL/0 语言的语法描述图

用语法图描述语法规则的优点是直观、易读。在图 2.1 的语法图中用椭圆和圆圈中的

图 2.1(a)　程序语法描述图

图 2.1(b)　分程序语法描述图