编译技术原理

姜文清 编

国防工业出版社

编译技术原理
国防工

314
59

编译技术原理

姜文清　编　　高永慧　审校

国防工业出版社

（京）新登字 106 号

图书在版编目（CIP）数据

编译技术原理/姜文清编．—北京：国防工业出版社，1994

ISBN 7-118-01136-3

Ⅰ．编… Ⅱ．姜… Ⅲ．编译程序 Ⅳ．TP314

中国版本图书馆 CIP 数据核字(94)第 03560 号

编译技术原理

姜文清 编　高永慧 审校

责任编辑 王祖珮

*

国防工业出版社出版发行

（北京市海淀区紫竹院南路 23 号）

（邮政编码 100044）

新华书店经售

北京市怀柔县王史山胶印厂印刷

*

开本 787×1092 1/16 印张 19½ 448 千字

1994 年 7 月第 1 版 1994 年 7 月北京第 1 次印刷 印数 1—3000 册

ISBN 7-118-01136-3/TP·147 定价：19.50 元

（本书如有印装错误，我社负责调换）

376326

前　言

随着计算机科学与软件工程的飞速发展，程序设计语言的编译原理、方法、技术和软件工具也在不断地发展和完善。近十几年来相继产生的C、Pascal、Ada等结构化程序设计语言，各种数据库管理系统和办公室自动化的应用软件，以及各类程序开发环境和软件工具，一方面应用编译技术原理的发展成果，另一方面使编译技术原理进一步规范、系统和完善，促进了编译技术原理的发展。

本书在选材方面注重反映80年代世界上关于编译技术原理的成熟成果。本书的编写，在对有代表性的结构化程序设计语言进行深入研究的基础上，较多地参考了国内外权威人士的权威著作，主要有：David Gries 的著作"Compiler Construction for Digital Computers"、A. V. Aho 和 J. D. Ullman 于1977年的著作"Principles of Compiler Design"，更重要的是 A. V. Aho，R. Sethi 和 J. D. Ullman 于1986年的著作"Compilers：Principles，Techniques，and Tools"，还有陈火旺、钱家骅和孙永强于1980年编著的《程序设计语言编译原理》等。

本书主要以Pascal和C语言的编译实例为背景，除了系统地介绍国内其它同类教材成熟部分的共同内容(比如词法分析和语法分析等)之外，本书的突出特点是对"语义"和"翻译"给出了规范化的形式描述，全面、系统地引入了关于属性(Attribute)、语法制导定义(Syntax—Directed Definition)和翻译规程(Translation Scheme)等概念；同时，书中介绍了不少关于Pascal、C和Ada等的实现技术，还有软件工具Lex和Yacc的实现原理和使用方法。

全书共分十章。第一章引论；介绍编译程序的功能、构造和与编译有关的预处理和后续处理。第二章，简单的编译程序设计；这一章是全书的缩影，通过对算术表达式的翻译，包括翻译成后缀表达式、求值和构造语法结构树(Syntax Tree)，引入上下文无关文法、语义和翻译等概念；并且通过具体的编译程序设计，引导读者学习编译技术原理入门，并且为编译实验奠定基础。第三章，词法分析；这一章先引入词汇(Token)、模式(Pattern)和词形(Lexeme)的概念，接着介绍正规表达式(Regular Expression)和词法分析程序设计技术。在这一章的最后，着重介绍有穷自动机理论，以弥补有些读者没学过"形式语言"课程的不足。第四章，语法分析；先介绍上下文无关文法。对自顶向下分析和自底向上分析，分别讨论它们的共同问题。对自顶向下分析来说，由于第二章已经介绍了递归下降法，所以，这里只讨论LL(1)分析法。对自底向上分析法来说，比较简要地介绍50年代产生的传统分析技术——算符优先分析；然后着重讨论LR分析技术，这种分析技术理论水平较高，可以导致编译程序设计自动化。这一章和第二章讨论的都是确定的语法分析方法，本书不讨论带回溯的语法分析方法。第五章，语法制导翻译；这一章引入属性、属性文法、语法制导定义、翻译规程等，以及有关的一系列新概念，这些概念反映80年代编译技术和编译理论的成熟成果；以后各章应用这些概念描述语义和所进行的翻译，使之规范化和形

式化。第六章，类型检查；这是结构程序设计语言的重要问题。第七章，中间代码生成；主要讨论结构化程序设计语言的翻译，随着语法分析翻译出中间代码。第八章，运行环境；主要介绍关于运行时的存储分配。第九章和第十章分别讨论代码优化和目标代码生成。书的最后是关于各章的习题和实验题。

该书初稿由哈尔滨工业大学姜文清副教授编写，作为教材在其校内使用。北京计算机学院高永慧副教授对该书进行了详细的审阅，提出了若干修改意见，并且根据其本人多年的教学经验，对书中第四章以及其他章节的内容进行了修改，完成了最后的定稿工作。

由于编者水平有限，书中难免存在缺点和错误，殷切希望广大读者和同行专家批评指正。

编　者

JS188/04

试读结束：需要全本请在线购买： www.ertongbook.com

内 容 简 介

本书在选材方面注重反映80年代世界上编译理论和编译技术的成熟成果，较多地参考了国内外权威人士的权威著作。本书的特点是对“语义”和“翻译”给出了规范化的形式描述，全面系统地引入了“属性”、“语法制导定义”和“翻译规程”等概念，用来描述语义和翻译。此外，书中介绍了不少关于Pascal、C和Ada等编译程序的实现技术，还有软件工具Lex和Yacc的原理和使用方法，这些都是七八十年代的成熟成果，无论是对科研、教学，还是对软件工程，既有理论意义，又有应用价值。本书的最后附有习题和实验题。本书可供工程技术人员参考，亦可作为大专院校编译课程的教材。

目　录

第一章　引论 …… 1
§1.1　概述 …… 1
§1.2　编译程序的构造 …… 2
1.2.1　分析 …… 2
1.2.2　符号表 …… 5
1.2.3　错误的检查与恢复 …… 5
1.2.4　综合 …… 6
§1.3　编译的预处理和后续处理 …… 8
1.3.1　预处理 …… 8
1.3.2　汇编 …… 9
1.3.3　装入和连接 …… 9
§1.4　编译的遍 …… 10
第二章　简单的编译程序设计 …… 11
§2.1　算术表达式的文法 …… 11
2.1.1　语法图和语法结构树 …… 11
2.1.2　上下文无关文法和语法树 …… 12
§2.2　语法分析程序设计 …… 15
2.2.1　确定的递归下降法 …… 15
2.2.2　确定的递归下降分析程序 …… 17
§2.3　关于语义的形式描述 …… 20
§2.4　关于翻译的形式描述 …… 22
§2.5　简单表达式求值 …… 26
§2.6　词汇的识别 …… 29
2.6.1　标识符的识别 …… 29
2.6.2　数的识别 …… 32
2.6.3　词法分析程序 …… 33
§2.7　语法结构树的建立 …… 34
第三章　词法分析 …… 40
§3.1　概述 …… 40
§3.2　词汇、模式和词形 …… 42
§3.3　输入缓冲 …… 43
§3.4　正规表达式 …… 45
3.4.1　串和语言 …… 45
3.4.2　正规表达式 …… 47
3.4.3　正规定义 …… 48
§3.5　词法分析程序设计 …… 49
3.5.1　词法分析器的规格说明 …… 49
3.5.2　状态转换图 …… 50
3.5.3　状态转换图的实现 …… 53
§3.6　软件工具 Lex …… 56
3.6.1　Lex 规格说明 …… 56
3.6.2　向前查看 …… 59
§3.7　有穷自动机 …… 60
3.7.1　非确定的有穷自动机(NFA) …… 60
3.7.2　确定的有穷自动机(DFA) …… 62
3.7.3　从 NFA 到 DFA 的转换 …… 62
3.7.4　由正规表达式构造 NFA …… 65
3.7.5　由正规表达式构造 DFA …… 67
3.7.6　DFA 的化简 …… 72
第四章　语法分析 …… 75
§4.1　概述 …… 75
§4.2　上下文无关文法 …… 76
4.2.1　上下文无关文法的定义 …… 76
4.2.2　推导和语法树 …… 77
4.2.3　文法的二义性 …… 79
4.2.4　非上下文无关语言 …… 81
§4.3　自顶向下分析的共同问题 …… 82
4.3.1　自顶向下分析法对文法的限制 …… 82
4.3.2　首符号集和后随符号集 …… 83
§4.4　LL(1)分析法 …… 85
4.4.1　LL(1)文法 …… 85
4.4.2　LL(1)分析器 …… 86
4.4.3　LL(1)分析的错误恢复 …… 88
4.4.4　LL(1)分析表的构造 …… 90
§4.5　自底向上分析的共同问题 …… 93
§4.6　算符优先分析法 …… 95
4.6.1　算符优先文法 …… 95
4.6.2　算符优先分析算法 …… 97
4.6.3　优先函数 …… 99
§4.7　LR 分析法 …… 100
4.7.1　LR 分析器 …… 100
4.7.2　LR(0)分析表 …… 104
4.7.3　SLR 分析表 …… 108
4.7.4　规范 LR 分析表 …… 114

4.7.5 LALR 分析表 …………………… 119

4.7.6 二义性文法的处理 ………… 126

4.7.7 LR 分析的错误恢复 ………… 129

§4.8 软件工具 Yacc …………………… 131

第五章 语法制导翻译 …………………… 136

§5.1 语法制导定义…………………… 136

§5.2 属性的依赖关系…………………… 137

5.2.1 综合属性 …………………… 137

5.2.2 继承属性 …………………… 138

5.2.3 属性的依赖关系图 ………… 139

§5.3 语法结构树…………………… 141

§5.4 S 属性定义的求值 …………………… 145

§5.5 L 属性定义与翻译规程…………… 147

§5.6 自顶向下翻译…………………… 150

5.6.1 消除翻译规程中的左递归 …… 150

5.6.2 自顶向下翻译器的设计 ……… 154

§5.7 L 属性定义的求值 …………………… 156

§5.8 属性值的存储与生存期…………… 160

第六章 类型检查 …………………… 168

§6.1 类型系统…………………… 168

§6.2 类型检查器的规格说明…………… 171

6.2.1 表达式的类型检查 ………… 172

6.2.2 语句的类型检查 …………… 173

6.2.3 函数的类型检查 …………… 173

§6.3 类型表达式的等价性…………… 174

6.3.1 类型表达式的结构等价 ……… 174

6.3.2 类型表达式的名称等价 ……… 176

6.3.3 指针类型的循环定义………… 178

§6.4 类型转换…………………… 179

§6.5 函数和运算符的重载…………… 181

第七章 中间代码生成 …………………… 183

§7.1 中间语言…………………… 183

7.1.1 图表示法 …………………… 183

7.1.2 三地址代码 …………………… 184

§7.2 关于说明的翻译…………………… 189

7.2.1 变量说明 …………………… 190

7.2.2 过程说明 …………………… 190

7.2.3 记录说明 …………………… 193

§7.3 赋值语句的翻译…………………… 193

7.3.1 简单变量的访问 …………… 193

7.3.2 数组元素和记录的访问……… 196

§7.4 布尔表达式的翻译…………………… 200

7.4.1 数值表示法 …………………… 201

7.4.2 控制流表示法 …………………… 202

§7.5 CASE 语句的翻译 …………………… 205

§7.6 回填技术…………………… 208

7.6.1 布尔表达式的翻译技术 ……… 208

7.6.2 控制流语句 …………………… 212

§7.7 过程调用…………………… 214

第八章 运行环境 …………………… 216

§8.1 过程的活动态…………………… 216

§8.2 存储器组织…………………… 220

8.2.1 运行时存储器的划分………… 221

8.2.2 活动记录 …………………… 221

8.2.3 编译时局部数据的表示 ……… 222

§8.3 存储分配策略…………………… 223

8.3.1 静态存储分配 …………………… 224

8.3.2 栈存储分配 …………………… 226

8.3.3 堆存储分配 …………………… 229

§8.4 非局部名称的访问…………………… 230

8.4.1 分程序 …………………… 230

8.4.2 非嵌套过程的静态作用域 …… 232

8.4.3 嵌套过程的静态作用域……… 233

§8.5 参数传递…………………… 237

8.5.1 传值 …………………… 237

8.5.2 传地址 …………………… 239

8.5.3 传值结果 …………………… 239

8.5.4 传名 …………………… 240

§8.6 符号表…………………… [illegible]0

8.6.1 符号表项 …………………… 241

8.6.2 符号表的数据结构 ………… 242

8.6.3 符号表中的作用域信息 ……… 243

§8.7 动态存储分配…………………… 245

8.7.1 动态存储分配的语言设施 …… 245

8.7.2 动态存储分配技术 ………… 247

8.7.3 隐式存储回收 …………………… 248

第九章 代码优化 …………………… 251

§9.1 概述…………………… 251

9.1.1 程序优化 …………………… 251

9.1.2 代码优化器的结构 ………… 252

§9.2 基本块与控制流图…………………… 253

9.2.1 基本块及其划分 …………… 253

9.2.2 控制流图 …………………… 254

§9.3 基本块的优化…………………… 255

§9.4 局部优化…………………… 257

9.4.1 冗余的公共表达式 ………… 257

9.4.2 重复传送 …………………… 258

9.4.3 删除死块 …………………… 259

9.4.4 循环优化 …………………… 259

§9.5 循环优化 …… 259
9.5.1 代码外移 …… 259
9.5.2 归纳变量 …… 259
9.5.3 削弱计算强度 …… 260
第十章 目标代码生成 …… 262
§10.1 代码生成器的设计要点 …… 262
10.1.1 地址映射 …… 262
10.1.2 指令的选择 …… 263
10.1.3 寄存器的分配 …… 264
10.1.4 求值顺序的选择 …… 264
10.1.5 代码生成器的设计 …… 264
§10.2 目标机器 …… 264
§10.3 运行时的存储管理 …… 266
10.3.1 静态存储分配 …… 267
10.3.2 栈存储分配 …… 268
10.3.3 运行时名称的地址 …… 271
§10.4 名称的引用信息 …… 271
§10.5 简单的代码生成器 …… 272
10.5.1 代码生成算法 …… 273
10.5.2 函数 getreg …… 274
10.5.3 其它语句的代码生成 …… 275
§10.6 窥孔优化 …… 277
10.6.1 冗余传送 …… 277
10.6.2 死代码 …… 278
10.6.3 控制流优化 …… 278
习题 …… 280
实验题 …… 296
参考文献 …… 301

第一章 引 论

通常，程序设计都使用高级程序设计语言。编译程序把用高级语言编写的程序翻译成计算机可以执行的程序。编译程序的设计原理和实现技术具有普遍性。本书中介绍的原理、方法和技术，对计算机科学工作者来说，都能经常用到。设计编译程序涉及到程序设计语言、形式语言与自动机理论、计算机系统结构、数据结构与算法设计，以及软件工程等各个方面。这一章介绍编译程序的功能、基本构造、编译的对象和编译程序的工作环境，以期使读者对本书的内容和范围有初步的了解。这一章涉及到的基本知识将在以后各章逐步详细地讨论。

§1.1 概 述

简单地说，**编译程序**(compiler)是一个翻译程序，它读用一种语言写的程序，称为**源程序**(source program)，将其翻译成等价的另一种语言的程序，称为**目标程序**(target program)，并且把源程序中存在的错误报告给用户，如图 1.1.1 所示。

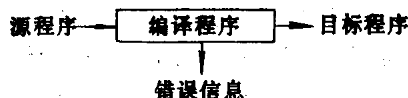

图 1.1.1 编译程序的功能

用来编写源程序的语言称为**源语言**。源语言有许多种，既有众所周知的程序设计语言，像 FORTRAN、Pascal、C 等，也有在某一应用领域中专用的语言。同样，描述目标程序的语言，称为**目标语言**，也有许多种。它可以是某种计算机的机器语言，也可以是另一种程序设计语言。

编译程序工作时是把一个结构上具有独立性的完整程序或程序模块翻译成具有独立性的目标程序；这就像把一篇外文文章翻译成中文文章一样。与此不同的是**解释程序**(interpreter)，它翻译源程序中的一个片段，随之执行，然后接着翻译下一个片段，再随之执行。这和一个“口头翻译”一样，外国人讲一段话，翻译人员译出一段话。可见编译程序与解释程序的区别是后者不生成目标程序。

和编译程序类似的还有**汇编程序**(assembler)，它把用某种计算机的汇编语言写出的程序翻译成这种机器的机器语言程序。

通常，由编译程序和汇编程序翻译出来的机器语言程序还要进行连接装配，使各程序模块以及系统所提供的库子程序连接成一个完整的程序，并且还要确定程序在执行时每条指令及其所访问的地址，从而形成一个能在计算机系统中执行的程序，称为**可执行程序**(executable program)。因而，把完成这种连接和装配工作的程序称为**连接装配程序**

(linker 或 loader)。

源语言的种类很多，其共同特点是便于描述数据、对象、算法和操作，并且一般不依赖机器；此外，还要便于人们学习和使用。目标语言的种类也不少，一种程序设计语言可能做为另一种程序设计语言的目标语言。从程序设计的目的来讲，最终要求计算机完成源程序中规定的任务。可见，最终的目标语言还是机器语言。从计算机这方面来讲，从做为单片机的微处理器，到超大规模的计算机，名目繁多。然而尽管如此，编译程序的基本任务实质上都是一样的。掌握编译的基本任务，应用基本的原理、基本的方法和基本的技术，可以为各种各样的源语言和目标语言构造编译程序。

50 年代初开始出现第一个 FORTRAN 编译程序，它的研制当时用了 18 人年。于是，人们普遍认为设计编译程序是十分困难的，令人望而生畏。在自此以后的 30 年里，直到 Ada 语言编译程序的实现，已经逐步地建立起了一整套成熟的编译理论、方法和技术，开发了高效率的结构化程序设计语言，可以方便地用来描述编译程序，建立了开发编译程序的环境和软件工具。在此基础上设计并实现一个编译程序已经不再是高不可攀的难题。

§1.2 编译程序的构造

按照编译程序的执行过程和它所完成的任务，可以把它分成前后两个部分，前边是**分析部分**，后边是**综合部分**。在分析部分里，分析源程序的结构成分，看其是否符合语言的规定，确定源程序所表示的对象和规定的操作，并且用一定的中间形式表示出来。分析部分包括**词法分析**、**语法分析**和**语义分析**。综合部分根据分析产生的结果构造所要求的目标程序。这部分包括**中间代码生成**、**代码优化**和**目标代码生成**。为了记录分析过程中识别出的标识符及其有关的信息，需要引入一个**符号表**。此外，如果在编译过程中发现错误，则要向用户报告，并且为了使编译继续进行下去，要进行适当的处理。

编译程序的基本构造如图 1.2.1 所示。

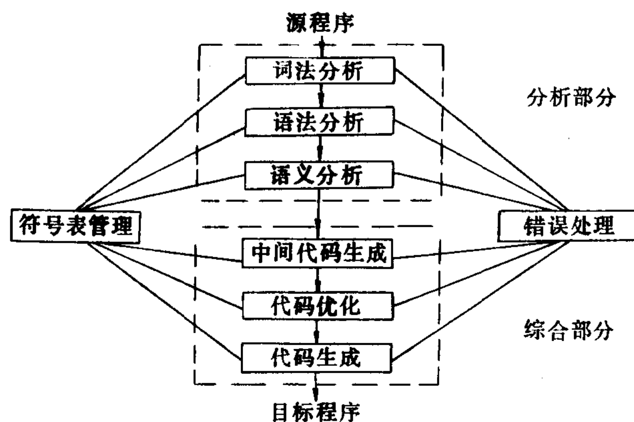

图 1.2.1 编译程序的基本构造

1.2.1 分析

编译的**分析部分**由三个阶段组成：**词法分析**、**语法分析**和**语义分析**。

试读结束：需要全本请在线购买： www.ertongbook.com

词法分析阶段(lexical analysis)也称为**扫描**(scanning)，是一种线性分析，依次读源程序中的每个字符，识别出每个具有独立意义的字符串，作为**词汇**(token)。例如，对赋值语句

position := initial + rate * 60　　(1.2.1)

进行词法分析，将得到下列词汇：

(1) 标识符　position
(2) 赋值号　:=
(3) 标识号　inital
(4) 加号　+
(5) 标识符　rate
(6) 乘号　*
(7) 数　60

识别标识符要根据构词规则，即**词法**。例如，标识符的构词规则是以字母开头的字母数字序列。在扫描输入流时，当遇到第一个字母之后继续扫描，直到发现既不是字母也不是数字时可以确定，从第一个字母开始到最后一个字母或数字的字符串构成一个标识符。识别出来的标识符要记入符号表。

在词法分析过程中，通常跳过空格，不做任何处理。同样，对程序中的注释也全部跳过，不做处理。

语法分析(Parsing 或 Syntax analysis)是一种层次结构的分析，按照语法规则识别源程序中的词汇序列所构成的句子。例如，对语句(1.2.1)进行语法分析得到图 1.2.2 所示的层次结构，称为**语法分析树**(parsing tree)，简称**语法树**。

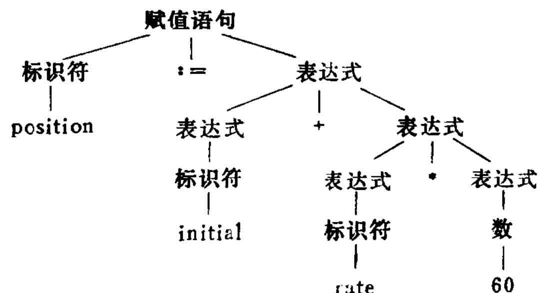

图 1.2.2　语句 position:=initial+rate*60 的语法树

语法树的层次结构可以用递归规则表示。例如，下列规则定义包含+、*运算的表达式：

(1) 任意一个标识符是一个表达式；
(2) 任意一个数是一个表达式；
(3) 如果 expression1和 expression2都是表达式，则

expression1 + expression1
expression1 * expression2
(expression1)

都是表达式。

上述(1)和(2)是基本规则，是非递归的。规则(3)通过把运算符用于其它表达式，从而定义了新的表达式。这样，按照规则(1)，initial 和 rate 都是表达式；按 照 规则(2)，60是一个表达式，再按照规则(3)，先推导出 rate * 60 是一个表达式；于是，最后推导出 initial＋rate * 60是一个表达式。

类似地，可以用下列规则递归地定义语句：

(1) 如果 identifer1是一个标识符，expression2是一个表达式，则

identifer1 := expression2

是一个语句。

(2) 如果 expression1是一个表达式，statemeent2是一个语句，则

while (expression1) **do** statement2

if (expression1) **then** statement2

都是语句。

语义分析(semantic analysis)是对语句的意义进行检查，确认程序中各构造成分组合到一起的含义，或者发现其中存在的错误，并且为生成目标代码收集类型等必要的信息。语义分析使用语法分析确定的层次结构表示表达式和语句。

语义分析的一项重要任务是类型检查，按照语言的类型规则检查和每个运算符相关的操作数。例如，许多程序设计语言都要求编译程序报告发现用实型数做数组下标的错误。然而，一般允许二元运算符的操作数既可以是整型数，也可以是实型数，这样，运算时要进行类型转换。

赋值语句(1.2.1)的右部表达式中的整型数和实型数，即使其值相同，在计算机里表示其值的二进制代码也不相同。对图1.2.1所示赋值语句的语法树，用紧缩的形式表示，可以表示成图1.2.3(a)所示的语法树；假定其中的所有标识符都说明为实型变量，60本身是一个整型数。对该语法树进行类型检查发现，乘法运算符 * 施于一个实型量

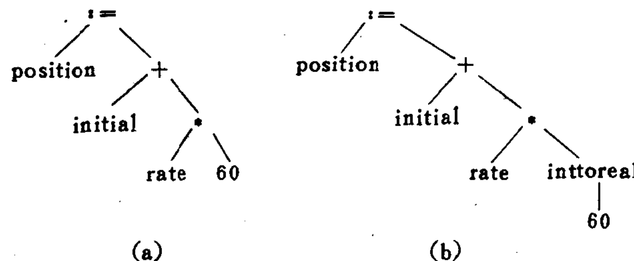

图1.2.3 在语法树中插入表示类型转换的结点

rate 和一个整型数60。运算时一般要把整型量转换成实型量。于是，对该语法树插入一个结点 inttoreal，表示把整型量变成实型量的类型转换，得到图1.2.3(b)所示的语法树。

在以上每个分析阶段里，编译程序都把源程序变换成便于下一阶段处理的内部表示形式。对语句(1.2.1)经每一阶段分析之后产生的内部表示形式如图1.2.4所示，其中的 **id1**、**id2**和 **id3**分别表示一个标识符。

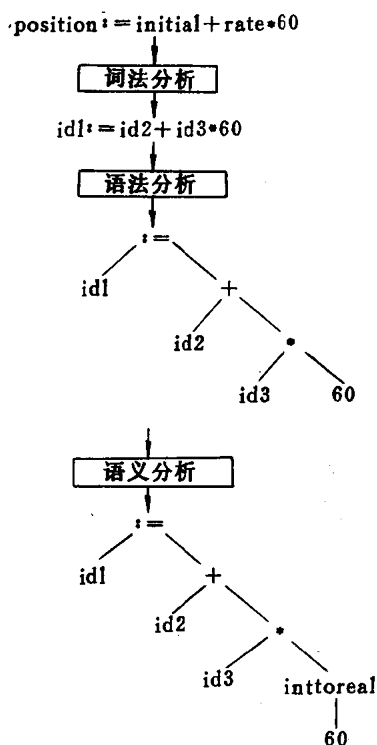

图1.2.4　对语句进行分析的各阶段

1.2.2　符号表

为了记录和引用标识符及其有关的信息，编译程序引入一个**符号表**。在词法分析阶段把识别出来的每个标识符记入符号表。在以后的分析与综合阶段还要随时收集关于每个标识符的信息填入符号表，以便引用。这些信息主要包括标识符用来表示变量名、过程名、函数名、还是形式参数名等等；对变量名还要记入变量的类型，对形式参数名还要记入参数传递的方式，等等。关于标识符的这些信息，我们将其统称之为"**属性**"。

符号表是由若干记录组成的数据结构，每个记录包含标识符及其各种属性的域。

标识符的各种属性是在编译的各个不同阶段填入符号表的。例如，对 Pascal 的变量说明：

var position，initial，rate：**real**

词法分析时只能识别出 position、initial 和 rate 是标识符，**var** 和 **real** 是关键字，但还不知道 **var** 和 **real** 的含义，所以并不能确定它们是用来说明变量的类型。语法分析时根据语法规则可以识别出这三个标识符都表示变量，并且 **real** 表示这三个变量的类型，于是，可以把这两种属性填入符号表。在以后生成目标代码时还要在符号表中填入对这三个 real 类型的变量进行存储分配的信息。

1.2.3　错误的检查与恢复

在编译的每个阶段都可能发现错误。发现错误之后应做适当处理，使得编译过程可以继续进行。如果发现第一个错误就停止编译，往往会降低调试程序的效率。当然，在交互式软件开发环境下进行编译，发现一个错误后立即进入编辑，为修改源程序提供了

方便。

词法分析阶段只能检查诸如“非法字符”之类的错误。语法分析和语义分析阶段可以检查出源程序中的大部分错误。语法分析阶段按照语法规则进行检查，发现不符合语法规则的错误。语义分析是在具有正确语法结构的前提下，检查不合法操作的错误。比如，表达式中加号的两侧都是标识符，但其中一个是变量名，另一个是数组名或者过程名。这种情况，按照语法规则，是正确的表达式，但其相加没有意义。

通常在编译过程中确定错误的种类和出错的位置，并且记录下来，以便编译程序执行完之后向用户提示。

1.2.4 综合

综合部分有三个阶段：**中间代码生成**、**代码优化**和**目标代码**生成。

经语法分析和语义分析以后，编译程序通常由源程序生成一种中间表示形式，即所谓**中间代码**。我们可以把这样的中间代码看成是一种抽象机器的程序。中间代码应该具有两个重要的特点：第一，容易产生；第二，容易翻译成目标程序。

中间代码有多种形式，其中一种称为**三地址代码** (three address code)。这种代码类似某种机器的汇编语言，所用到的每个内存单元都能起到类似寄存器的作用。**树地址代码**由指令序列组成，每条指令最多有三个操作数。用树地址代码，赋值语句 (1.2.1)翻译成如下的指令序列：

temp1 ：=inttoreal(60)
temp2 ：=**id3** * temp1
temp3 ：=**id2** + temp2
id1 ：=temp3　　　　(1.2.2)

这种中间代码具有三个性质：第一，每条树地址指令除赋值之外还有最多一个运算符；这样，当生成这些指令的时候，编译程序必须确定将来要进行的运算顺序。在语句(1.2.1)里，乘运算一定要在加之前进行。第二，编译程序必须生成临时变量名，以便保留由每条指令求出的值。第三，一些三地址指令少于三个操作数，比如上述的最后一条指令。

除此之外，还有其它形式的中间代码。一般来说，中间代码所表示的操作比源程序语句所表示的操作较为详细，因为这里要考虑在计算机上实现时用汇编指令表示的细节，另外还要考虑控制流和过程调用以及参数传递的各个细节。

代码优化(code optimization)是对代码进行改进，使之占用的空间少，运行速度快。编译程序的代码优化首先是在中间代码一级进行的，由优化的中间代码可以得到较优的目标代码。

上述指令序列(1.2.2)所示的中间代码，是根据语义分析之后所产生的树结构，用三地址代码表示，使每个运算符用一条指令，从而得到这组中间代码，这种算法是很自然的。可是，为了完成同样的运算还有更好的算法，原来的四条指令可以用两条指令完成：

temp1 ：=**id3** * 60.0
id1 ：=**id2** + temp1　　　　(1.2.3)

由于在代码优化阶段问题已经确定，编译程序可以使60由整型数转换成实型数，并且这一转换和它参加运算同时完成，所以，操作 inttoreal 可以被省略。此外，在指令序列

(1.2.2)中的 temp3只用一次，便把它的值传给 **id1**，而 **id1**在此之前并未使用，所以，用 **id1**替换 temp3是安全的。于是得到指令(1.2.3)的优化代码。

不同的编译程序所进行的代码优化量差别很大。进行优化最多的叫"**优化编译**"(opimizing compiler)，其编译的大部分时间花费在代码优化阶段。也有的编译只做简单的优化，其主要目的在于改进目标程序的运行时间，并且不至于使编译的时间太长。

编译的最后阶段是生成目标代码。目标代码一般是可重定位的机器代码或汇编语言代码。为了生成目标代码，对程序中使用的每个变量要指定存储单元，并且把每条中间代码的指令一一地翻译成完成相同任务的机器指令。为了得到运行速度快的目标代码，重要的问题是对变量分配寄存器。

例如，对上述指令(1.2.3)的中间代码翻译成汇编代码，我们使用两个寄存器 R1和 R2，代码如下：

```
MOVF  id3    , R2
MULF  #60.0, R2
MOVF  id2    , R1          (1.2.4)
ADDF  R2     , R1
MOVF  R1     , id1
```

其中每条指令操作码后边的 F 表示指令执行浮点数操作；每条指令后边的第一个操作数指定源操作数，第二个操作数指定目的操作数；符号 # 表示该操作数是个立即常数。这一组代码表示把地址 **id3**中的内容移到 R2，再乘以实常数60.0仍存在 R2。第三条指令把地址 **id3**中的内容取到 R1，然后把 R1和 R2的内容相加结果放在 R1。最后把 R1的内容存入地址为 **id1**的存储单元。于是完成语句(1.2.1)中的赋值。

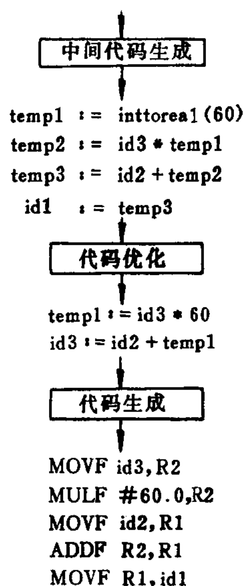

图1.2.5　综合部分的各个阶段

对赋值语句(1.2.1)，经图1.2.4所示的各分析阶段之后，继续进行综合过程各阶段

的编译，所产生的结果如图1.2.5所示。

§1.3 编译的预处理和后续处理

为了翻译源程序，常常需要预先对其做某些处理，把处理的结果作为编译程序的输入。我们把编译之前预先处理源程序的程序称为**预处理程序**(preprocessor)。在编译之后，产生可执行的目标代码之前，对编译产生的结果往往还要继续进行某些处理，比如汇编、连接和装配，如图1.3.1所示。

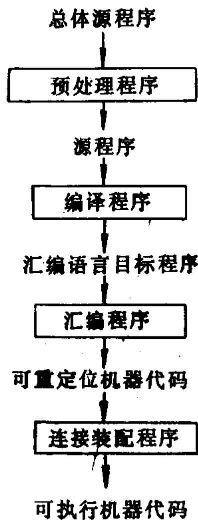

图1.3.1 编译的预处理和后续处理

1.3.1 预处理

在编译之前，由预处理程序对源程序进行处理，主要有下列几种：

宏定义:在源程序中允许用户定义**宏**(macro)。比如在C语言的源程序里，

#define prompt(s) fprintf(stderr,s)

是一个**宏定义**。在编译之前需要先进行宏处理，用 fprintf(stderr,s)代替程序里的全部 prompt(s)。

文件蕴含:通常，在源程序里允许出现文件蕴含控制行。比如，C 语言程序的"头文件"蕴含控制行：

#inclnde <stdio.h>

还有 MS Pascal 程序里的

(* $INCLUDE:'GRAPH.PAS' *)

在进行文件蕴含预处理时，把文件名标识的文件全部内容插在源程序中该蕴含控制行所在的位置。

有理预处理程序(rational processor)对一种语言的老版本编译增加某些结构化数据和控制流设施。比如，某种语言的老版本根本没有像 **while** 循环或 **if—then—else** 这样的控制结构。当程序中使用这样的设施时，由有理预处理程序通过**宏调用**实现语言新增加

试读结束：需要全本请在线购买： www.ertongbook.com