编译原理及编译程序构造

高仲仪　金茂忠　编

北京航空航天大学出版社

编译原理及编译程序构造

高仲仪　金茂忠　　编

北京航空航天大学出版社

内 容 简 介

本书为高校计算机专业学习程序设计语言编译原理和方法的教材。全书内容分为两部分：第一部分介绍编译程序的设计原理和构造；第二部分介绍两个较为典型的小型编译系统 PL/0 和PASCAL-S编译程序。

本书较系统地介绍了翻译文法和属性文法的概念和表示，并用它们来描述程序语言的翻译过程。由于这种描述是很接近形式化的，所以能够更系统、更清楚地说明语法、语义分析和代码生成的过程。这将有利于读者学习和理解这部分内容。

书中还介绍了近年来在编译程序的自动生成工具的研制方面所取得的一些成果以及编译的原理和方法在软件工程中的应用。最后介绍了 PL/0 和 PASCAL-S 编译程序。书中给出了这两个系统的全部源程序和编译实例。为了提高可读性，在源程序中加上了必要的注释。

本书取材广泛新颖，在内容组织上注意了理论联系实际、由浅入深及循序渐进的原则，以便于读者阅读。

本书可作为高等院校计算机专业程序设计语言编译课程的教材，也可供软件工程技术人员参考和作为自学用书。

编译原理及编译程序构造

BIANYI YUANLI JI BIANYI CHENGXU GOUZAO

高仲仪　金茂忠　　编

再版编辑　冯学民

北京航空航天大学出版社出版

新华书店总店科技发行所发行　各地新华书店经销

北京宏文印刷厂印装

*

787×1092　1/16　印张：28.5　字数：730 千字

1990 年 12 月第一版，2001 年 3 月第 4 次印刷　印数：13001～18000 册

ISBN 7-81012-186-3/TP・031　　定价：32.00 元

前　言

编写本书的主要目的是为高校计算机专业的学生学习程序设计语言的编译理论和方法提供一本适用的教材。全书内容分为两大部分。第一部分从第一章到第十三章，系统介绍编译的基本理论和编译程序的构造。主要内容有：文法和语言的概念和表示，词法分析，语法分析，语法制导翻译技术，符号表管理，运动时的存贮组织及分配，源程序的中间形式，错误处理，语义分析和代码生成，代码优化，与机器有关的优化和目标代码生成，编译程序的生成方法和工具等。第二部分包括第十四、十五章，介绍两个较为典型的小型编译系统：PL/0 和 PASCAL-S 编译程序。

考虑到编译是计算机专业的重要基础技术之一以及软件工程的迅速发展，因此在编写本教材时结合课程内容介绍了编译的原理和方法在软件工程中的一些应用。在介绍语法制导翻译技术时，较系统地介绍了翻译文法和属性翻译文法的概念及其对翻译过程的描述，使得语义分析和代码生成的过程更趋于形式化，从而有利于这部分内容的教学。

为软件开发过程提供各种工具以实现软件开发的工程化和自动化，一直是人们所追求的目标。在程序语言的翻译领域内，已取得了不少具有实际意义的进展，本书在第十三章编译程序的生成方法和工具中，较为系统地介绍了这方面的成果。

程序语言的编译原理和方法是一门实践性很强的课程，所以，最好的学习方法是通过实践来理解、掌握和加深所学的内容。为了帮助读者学好这门课程，在教材的第二部分提供了两个较为典型的小型编译系统，供读者在学习了前十三章内容以后阅读和分析。这两个系统都经过认真审校，并作了适当的变动和修改，现已在 IBM-PC 等微机上用 TURBO PASCAL 调试成功。为了提高可读性，在程序中还加了适当的注释。在条件许可时，希望读者能自己动手设计并实现一个编译程序。

在各章节之后均附有一定数量的练习，其中打星号的是选作题。

本教材在内容的选取和组织方面，注意了理论和实践并重，理论联系实际，由浅入深、循序渐进，基本概念的阐述力求详尽，以适于读者阅读或作为自学用书。

本书的前十三章由高仲仪编写，后两章的编写和 PL/0 及 PASCAL-S 的修改和调试工作由金茂忠完成。在本书的编写过程中，虞梅保和沈瑛同志对书稿作了许多文字上的修改以及抄写和描图等工作。阎龙同志和赵磊同学协助进行了 PASCAL-S 编译程序的调试。我们对她（他）们为本书所作的工作表示由衷的感谢。

由于我们学识浅薄，编写时间仓促，故会有错误或不妥之处，请读者批评和指正。

编　者

一九八九年七月

目　录

第一章　引　言

1.1　程序设计语言 …… 1
1.2　翻译程序 …… 2
1.3　编译程序模型 …… 4
1.4　编译程序的前后处理器 …… 9
1.4.1　预处理器 …… 9
1.4.2　汇编程序 …… 10
1.4.3　两遍汇编 …… 10
1.4.4　加载器和连接编辑器 …… 11
1.5　编译技术在软件工程中的应用 …… 11
练　习 …… 13

第二章　文法和语言的概念和表示

2.1　文法的非形式讨论 …… 14
2.1.1　语法树 …… 14
2.1.2　规则 …… 15
2.1.3　由规则推导句子 …… 15
练　习 …… 17
2.2　符号、符号串及其集合的运算 …… 18
2.2.1　字母表和符号串 …… 18
2.2.2　符号串及其集合的运算 …… 18
练　习 …… 19
2.3　文法和语言的形式定义 …… 20
2.3.1　文法的形式定义 …… 20
2.3.2　推导的形式定义 …… 21
2.3.3　语言的形式定义 …… 22
2.3.4　递归规则与递归文法 …… 24
2.3.5　短语、简单短语和句柄 …… 25
练　习 …… 26
2.4　语法树和二义性 …… 27
2.4.1　推导与语法树 …… 27
2.4.2　文法的二义性 …… 30
练　习 …… 33
2.5　符号串的分析 …… 35
2.5.1　自顶向下分析 …… 35

2.5.2 自底向上分析…… 36
2.6 有关文法的实用限制…… 37
练 习 …… 38
2.7 扩充的 BNF 表示和语法图 …… 39
2.7.1 扩充的 BNF 表示 …… 39
2.7.2 语法图…… 40
2.8 文法和语言分类…… 41
第三章 词法分析
3.1 词法分析程序的功能…… 43
3.2 源程序的输入与词法分析程序的输出…… 44
3.2.1 源程序的输入…… 44
3.2.2 单词符号的种类及词法分析程序的输出形式…… 45
3.3 正则文法及其状态图…… 45
3.3.1 状态图…… 46
3.3.2 状态图的作用…… 46
3.4 词法分析程序的设计与实现…… 47
3.4.1 文法及其状态图…… 47
3.4.2 词法分析程序的构造…… 48
3.4.3 词法分析程序的实现…… 49
3.5 正则文法和正则表达式…… 52
3.6 有穷自动机(FA) …… 54
3.6.1 确定的有穷自动机(DFA) …… 54
3.6.2 不确定的有穷自动机(NFA) …… 56
3.6.3 NFA 的确定化 …… 57
3.6.4 正则表达式与有穷自动机的等价性…… 59
3.7 词法分析程序的自动生成器…… 63
3.7.1 LEX 源程序(LEX 的输入文件) …… 63
3.7.2 LEX 的实现 …… 64
练 习 …… 67
第四章 语法分析
4.1 自顶向下分析方法…… 69
4.1.1 带回溯的自顶向下分析算法…… 69
4.1.2 存在的问题及解决办法…… 70
4.2 递归下降分析(递归子程序法)…… 75
练 习 …… 80
4.3 LL(1)分析方法 …… 80
4.3.1 LL(1)分析器的逻辑结构及工作过程 …… 80
4.3.2 LL(1)分析表的构造方法 …… 84
练 习 …… 87

4.4 自底向上分析方法…… 88
4.5 算符优先分析法…… 90
4.5.1 方法概述…… 90
4.5.2 直观算符优先分析法…… 92
4.5.3 算符优先分析法的进一步讨论…… 95
练 习…… 100
4.6 LR 分析方法 …… 100
4.6.1 概念和术语 …… 101
练 习…… 104
4.6.2 LR(0)分析器 …… 104
练 习…… 108
4.6.3 SLR(1)分析器…… 109
练 习…… 115
4.6.4 规范 LR(1)分析器 …… 117
练 习…… 123
4.6.5 LALR(1)分析器 …… 123
练 习…… 127
第五章 语法制导翻译技术
5.1 翻译文法(TG)…… 128
5.2 语法制导翻译 …… 130
5.3 属性翻译文法(ATG) …… 131
5.3.1 综合属性 …… 131
5.3.2 继承属性 …… 133
5.3.3 属性翻译文法 …… 134
5.3.4 属性翻译文法举例——算术表达式的翻译 …… 136
练 习…… 138
5.4 自顶向下语法制导翻译 …… 139
5.4.1 翻译文法的自顶向下处理 …… 139
练 习…… 144
5.4.2 属性文法的自顶向下翻译 …… 145
练 习…… 156
5.5 自底向上的语法制导翻译 …… 158
5.5.1 波兰翻译 …… 159
5.5.2 S－属性文法 …… 160
练 习…… 161

第六章　符号表管理技术

6.1　前景和动机 …… 163
6.2　何时建立和访问符号表 …… 163
6.3　符号表内容 …… 165
6.4　在符号表上的操作 …… 167
6.5　非分程序结构语言的符号表组织 …… 169
6.5.1　无序符号表 …… 169
6.5.2　有序符号表 …… 170
6.5.3　散列符号表 …… 171
6.6　分程序结构语言的符号表组织 …… 180
6.6.1　分程序结构语言的概念 …… 180
6.6.2　栈式符号表 …… 182
6.6.3　散列符号表的栈式实现 …… 183
练　习 …… 185

第七章　运行时的存贮组织及管理

7.1　静态存贮分配 …… 186
练　习 …… 188
7.2　动态存贮分配 …… 188
7.2.1　活动记录 …… 190
7.2.2　参数区 …… 190
7.2.3　Display 区 …… 190
7.2.4　运行时的地址计算 …… 192
7.2.5　递归过程的处理 …… 193
练　习 …… 195
7.3　堆式存贮分配 …… 196
7.3.1　隐式存贮请求 …… 196
7.3.2　显示存贮请求 …… 197
7.3.3　堆式存贮管理技术 …… 198

第八章　源程序的中间形式

8.1　波兰表示 …… 205
8.2　N-元表示 …… 206
8.2　抽象语法树 …… 208
8.4　抽象机代码 …… 209
8.4.1　可移植性和抽象机 …… 209
8.4.2　PASCAL 的 P-code 抽象机 …… 210
练　习 …… 211

第九章　错误处理

9.1　概述 …… 212

9.2 错误的分类 …… 212
9.3 错误的诊察与报告 …… 213
9.4 错误处理技术 …… 214
9.4.1 错误改正 …… 215
9.4.2 错误局部化处理 …… 215
9.5 遏止重复的错误信息 …… 217
第十章 语义分析和代码生成
10.1 语义分析的概念 …… 219
10.2 栈式抽象机及其汇编指令 …… 221
10.3 声明的处理 …… 222
10.3.1 常量类型 …… 223
10.3.2 简单变量 …… 223
10.3.3 数组变量 …… 225
10.3.4 记录变量 …… 226
10.3.5 过程声明 …… 227
10.4 表达式 …… 228
10.5 赋值语句 …… 234
10.6 控制语句 …… 235
10.6.1 if 语句 …… 236
10.6.2 分情形语句 …… 237
10.6.3 repeat-while 语句 …… 238
10.6.4 for 循环语句 …… 239
10.7 过程调用和返回 …… 241
10.7.1 参数的基本传递形式 …… 241
10.7.2 过程调用 …… 242
10.7.3 返回语句和过程终止 …… 246
10.8 输入和输出语句 …… 246
10.8.1 输入语句 …… 246
10.8.2 输出语句 …… 249
10.9 编译程序的辅助功能 …… 249
练 习 …… 250
第十一章 代码优化
11.1 概述 …… 251
11.2 基本块 …… 251
11.3 常数合并 …… 252
11.4 冗余子表达式的消除 …… 257
11.5 循环内的优化 …… 265
11.5.1 循环展开 …… 266
11.5.2 频度削弱 …… 269

11.5.3 强度削弱…… 275
11.5.4 循环优化技术的综合…… 278
练 习…… 281
第十二章 与机器有关的优化及目标代码的生成
12.1 与机器有关的优化概述…… 283
12.2 寄存器分配的优化…… 284
12.2.1 单寄存器机器中的寄存器分配…… 284
12.2.2 多寄存器机器中的寄存器分配…… 290
12.3 目标机和目标代码生成…… 294
12.3.1 PDP-11 …… 295
12.3.2 VAX-11 …… 299
练 习…… 303
第十三章 编译程序生成方法和工具
13.1 编译程序的书写语言…… 305
13.2 自展…… 306
13.3 移植…… 306
13.4 编译程序—编译程序…… 307
13.4.1 YACC:一个 LALR(1) 分析器生成器 …… 308
13.4.2 属性 LL(1) 分析器生成器 …… 313
13.4.3 代码生成器的生成系统…… 319
第十四章 PL/0 简单编译系统
14.1 PL/0 语言 …… 324
14.2 PL/0 编译系统结构 …… 328
14.3 PL/0 的词法分析 …… 329
14.4 PL/0 的语法分析 …… 330
14.5 出错处理…… 332
14.6 目标代码的生成和解释执行…… 333
14.7 PL/0 程序编译和运行举例 …… 335
第十五章 Pascal-S 编译系统
15.1 Pascal-S 语言 …… 344
15.2 Pascal-S 编译程序的结构 …… 349
15.3 Pascal-S 编译程序 …… 353
15.3.1 表格…… 353
15.3.2 编译初启…… 358
15.3.3 实用程序…… 359
15.3.4 词法分析及处理…… 359
15.3.5 语法分析处理…… 359
15.3.6 出错处理…… 364

15.4 Pascal-S 解释执行程序 …… 366
15.4.1 P 代码指令系统 …… 366
15.4.2 运行栈 …… 368
15.4.3 运行时的 display …… 369
15.4.4 运行出错处理和现场剖析打印 …… 370
15.5 编译及运行的例子 …… 370
参考资料 …… 380
附录 A PL/0 编译系统源程序清单 …… 383
附录 B Pascal-S 编译系统源程序清单 …… 395

第一章 引 言

编译程序设计原理和构造是计算机科学家、软件工程技术人员所必备的专业基础知识。编译程序设计涉及到程序设计语言，计算机体系结构，语言理论，算法，数据结构和软件工程。在本章中，我们将通过描述程序设计语言，编译程序的组成以及编译程序的工作环境来介绍本书的主要内容。

1.1 程序设计语言

语言是人与人之间传递信息的媒介和手段。在计算机的应用领域，程序设计语言充当了人与问题和协助解决该问题的计算机之间的通讯工具。一种有效的程序设计语言能提高计算机程序的开发效率和对问题的表达能力；它必须能填补通常人们的非结构化的思维方式与计算机执行所要求的精确性之间的沟壑。程序设计语言正是沿着为更好地满足这方面要求而向前发展的。

在计算机发展的初期，人们直接使用机器语言(机器指令)编写程序(所编出的程序称为手编程序)。这种语言很不直观、难写、难读、容易出错，且出错以后又难于检查，查出错误也难于修改。程序设计和检查错误所花的时间往往比机器解题所需的时间多出百倍，甚至数千倍以上；而且机器指令因机器不同而异，难于在别的机器上运行。程序设计也要由受过一定训练并熟悉机器的程序员来做。这就大大限制了计算机的推广和使用。

之后，开始出现了符号语言，即用较直观的符号来代替用纯粹数字表示的机器指令代码。这样使程序便于记忆，也便于阅读和检查。在符号语言的基础上，又进一步发展成为汇编语言。用汇编语言编写程序，除用直观的记忆码代替操作命令以对应一条条机器指令之外，还增加了若干宏指令，这些宏指令构成指令码的扩展。宏指令对应一组机器指令，完成一些特定的功能。

但是汇编语言仍是依赖于具体机器的，对问题的描述仍处于低层次，使用起来仍不方便，且程序设计的效率也很低。为了进一步解决这些问题，人们参照数学语言设计了一类便于描述算法的语言，如 ALGOL 和 FORTRAN 等。由于这类语言完全摆脱了机器指令形式的约束，用它编出的程序更接近于自然语言和习惯上对算法的描述，故称为不依赖于具体机器的或面向用户的语言。

随后，又相继出现了许多专门用于描述某个应用领域问题的专用语言，如应用于数据库查询的语言 SEQUEL 以及用于土木工程的 COGO 语言等。这类语言我们称之为是面向问题或面向对象的。相对于依赖机器的机器语言和汇编语言，我们将不依赖于机器的面向用户的和面向问题或对象的语言统称为高级程序设计语言，简称高级语言；称机器语言和汇编语言为低级语言。

高级语言与低级语言相比有如下一些主要优点：

① 由于高级语言独立于机器，故对计算机了解甚少、甚至一点儿也不了解的用户也可学习和使用。且在一般情况，使用这类语言所编制的程序只需做很少的修改就能在别的机器上运

行，即良有好的可移植性。

② 程序员在编写程序时不必对程序中所出现的常数或变量分配具体的存贮单元，也不必知道如何将数据的外部形式转换成计算机存贮器中的内部形式等实现细节。这些任务都可留给将高级语言程序翻译为机器语言程序的系统进行处理，这就大大简化了编写程序的过程。

③ 高级语言提供了比低级语言丰富得多的数据结构和控制结构。数据结构有数组、记录等。使用这些结构能更好地表达数据本身及数据之间的内在关系。控制结构有条件语句、循环语句、分情形结构及过程调用等。这些结构改善了程序的风格，便于采用科学的方法（如结构化方法）来开发程序，降低了程序的复杂性，从而提高了程序可靠性，也降低了开发费用。

④ 最后一点也是至关重要的一点，就是高级语言更接近于自然语言。这就使得高级语言中的一条语句在一般情况下都对应着好多条汇编指令或机器指令，所以用高级语言编程比用低级语言编程效率要高得多。另外，用高级语言编写的程序具有良好的可读性，更便于进行交流和维护。

总之，用高级语言取代低级语言编写程序极大地方便了软件开发者，特别是非计算机专业的用户，提高了程序的开发效率、可靠性和可维护性。较大幅度地缩短了程序的开发周期，降低了程序的开发和维护费用。这在计算机特别是在软件的发展史上是一次具有重大意义的突破。

自 1956～1958 年第一种高级语言 FORTRAN 问世以来，各种语言如雨后春笋般地发展起来，相继出现了 ALGOL，COBOL，PL/I，PASCAL，JOVIAL，C，ADA 等高级语言。目前世界上高级语言大约已有几百种之多，其中大多数是专用语言，比较通用的也有几十种。

但是，正如大家所知，用高级语言编制的程序，计算机是不能立即执行的（因机器只能执行机器指令），必须通过一个“翻译程序”的加工，使之转变为与其等价的机器语言程序，机器才能执行。这种翻译程序，我们称之为“编译程序”。某种高级语言的编译程序加上一些相应的支持用户程序运行的子程序就构成了该语言的编译系统。编译系统是计算机系统的重要组成部分。所以用高级语言编程上机需要有相应的软件系统支持。

本课程的任务就是介绍这种软件系统的设计原理，构造及实现的技术。

1.2 翻译程序

翻译程序扫描所输入的源程序，然后将该源程序转换为目标程序。源程序是用高级语言或汇编语言写的，而目标程序则是用目标语言表示。

如果所翻译的源语言是汇编语言而目标语言是机器语言，那么，该翻译程序就称为汇编程序。汇编语言仍然是与机器密切相关的，在大多数汇编语言中，其指令的绝大多数均是相应的机器语言指令的符号表示。每一条汇编指令设有多个域，如操作符域和操作数域。且大多数汇编语言都有宏指令设施。一条宏指令可以扩展为一组机器语言指令。在一般情况下，在汇编语言中不包括高级语言的结构，如分情形语句、嵌套语句和分程序等。

若一个翻译程序将 FORTRAN、PASCAL 或 COBOL 那样的高级语言程序转变为一台具体计算机的机器语言或汇编语言程序，那么，我们称该翻译程序为编译程序。实现源程序到目标程序的转换所占用的时间称为编译时间，而目标程序是在运行时执行的。图 1.1 说明高级语言程序的编译和运行过程。可注意到源程序和数据是在不同时间，即分别在编译阶段和运行阶段进行处理的。

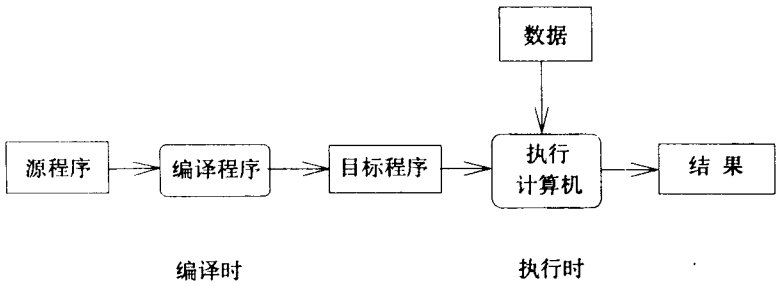

图 1.1　高级语言程序的编译和运行

另一类翻译程序称为解释程序，它同时处理源程序和数据。解释程序解释执行源程序但并不生成目标程序。图 1.2 说明了该解释过程。某些解释器每次直接分析一条所要执行的源语句。因这种方法很费时间，所以极少采用。一种更有效的方法是先将源程序转变成一种中间源程序形式，然后再来解释执行该中间源程序。例如，对于一个赋值语句，解释程序可以先将源程序转换成如图 1.3 所示的一棵树，然后在遍历该树时，执行结点上所规定的操作。如在树根结

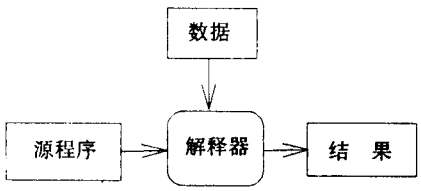

图 1.2　源程序解释过程

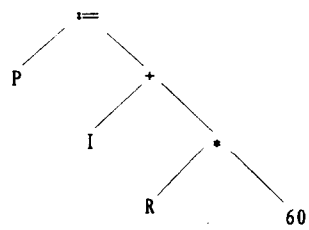

图 1.3　语句 P：=I+R * 60 的树结构表示

点，解释程序将发现有一个要执行的赋值操作，所以它将调用一个程序来计算右边的表达式，当返回时就可将计算结果放到与标识符 P 相关的存贮单元中。在树根结点的右子结点上，被调用程序将发现它必须计算两个表达式的和。因此，它将递归地调用它自己以计算表达式 R * 60 的值。然后将该值与变量 I 的值相加。

现在，解释程序已用得相当普遍，特别是在微型计算机环境中。原因在于解释执行方式更便于人机对话。当程序员需要根据前面执行的情况，随时调整后面的工作、随时更改后面的程序时，采用解释程序是很合适的。例如，BASIC、APL、LISP 和 SMALLTALK-80 这样一些语言所以能成为十分流行的语言，其主要原因之一就是它们都是在解释的环境下实现的。

在本书中，我们将主要讨论编译原理，编译程序构造和实现的技术。但是很多技术也同样适用于解释程序。一个熟悉编译原理与编译程序构造的人是不难了解解释程序的设计原理及实现方法的。至于汇编程序，它与编译程序相比是相当简单的，可以说对它们的讨论可以包含在对编译程序的讨论之中。

1.3 编译程序模型

为一种具体语言构造一个编译程序是件很复杂的任务。编译过程的复杂性在很大程度上取决于源语言。图 1.4 给出了一个编译程序的基本模型。虽然该模型对于编译不同的高级语言可以有所变化，但它仍不失是一种典型的编译过程的表示。

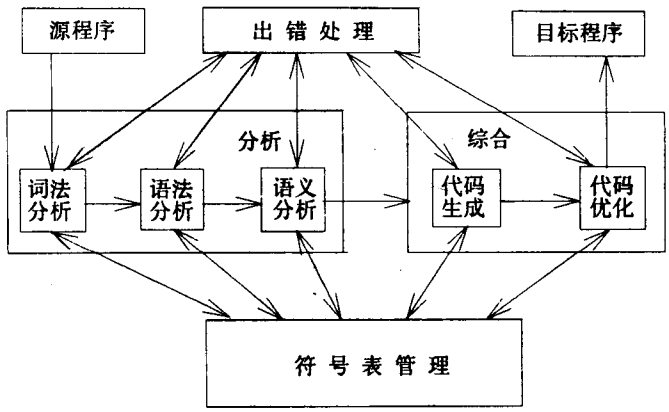

图 1.4 编译程序的组成

编译程序必须完成两个主要任务：一是源程序的分析，二是目标程序的综合。分析部分是将源程序分解为源程序的各个基本部分，并生成与它们相等价的源代码中间形式。这些任务很容易通过建立和使用几个表来完成。通常将分析部分划分为词法分析，语法分析和语义分析三个阶段。综合部分接受分析部分产生的中间代码，然后根据具体的目标机系统生成目标代码，并对代码进行优化处理。一般是将综合部分分为目标代码生成和代码优化两个阶段。下面对编译的各个阶段作简要的介绍。

词法分析 一个源程序通常是由字母、数字和某些专用符号如＋、－、(、)等所组成的串。一个源程序包含有许多基本的语言结构，如变量名、标号、常量、关键字和操作符等。所以编译程序先要识别出这些不同类型的语言结构。这些语言结构一般由语言的定义给出。

源程序(见图 1.4)被输入到词法分析程序(或称扫描程序)中，其目的是将字符串形式的源程序分解为具有独立语法意义的单词符号(Token)。如常量、变量名、关键字(如 PL/I 中的 DO、IF 和 THEN 等)和操作符。从本质上说。词法分析程序就是执行低级的语法分析。为了实现上的方便，每一类单词可用一个整数码表示。例如变量名用 1 表示，常量用 2 表示，标号用 3 表示，操作符“＋”用 4 表示等等。例如下面的 PL/I 语句：

TEST： IF A>B THEN X=Y；

将由词法分析程序翻译为下列的单词序列(单词用整数码表示)：

TEST	3
:	26
IF	20
A	1

>	15
B	1
THEN	21
X	1
=	10
Y	1
;	27

应该注意的是，在对源语句进行词法分析并产生每个单词的内部表示时已经略去了语句中的所有空格字符。一般来说，词法分析程序必须处理空格和注释。因为在一般的高级语言中，空格（出现在字符串常量中的例外）和注释不表示一个程序的可执行部分，所以可以在词法分析时删去。

某些程序设计语言如 FORTRAN，语句之间不设专门的分隔符且又允许语句在多行上继续。对于这种语言，词法分析程序必须对这种占有多行的语句进行处理（即增设语句分隔符和完成续行连接）。有时候，某些扫描程序还将常量、标号和变量名填入相应的表中。例如，一个变量的表项可以包含变量名，类型（例如实型、整型或布尔型），目标地址，变量值以及声明该变量的行号等。

词法分析程序将所识别的单词提供给语法分析程序。单词可以取一对偶的形式（即两个项）。第一项给出地址或单词在某个符号表中的位置，第二项是单词的整数码。这种表示法为语法分析提供了方便，使得所有单词都能用固定长度的信息表示：一个地址（或指针）和一个整数。在第三章中，我们将详细介绍词法分析的过程和词法分析器的构造。

语法分析 语法分析程序比词法分析程序要复杂得多。语法分析程序的功能是从词法分析程序取得源程序（单词串形式的），并将一个或多个单词组合为语言的各种语法类。该分析过程是与在英语中分析和确定一个句子的结构相类似的。在分析英语句子的时候，我们感兴趣的是要识别出各类语法类如“冠词”、“名词”、“动词”、“形容词”、“主语”、“谓语”、“简单句”、“复合句”等等。

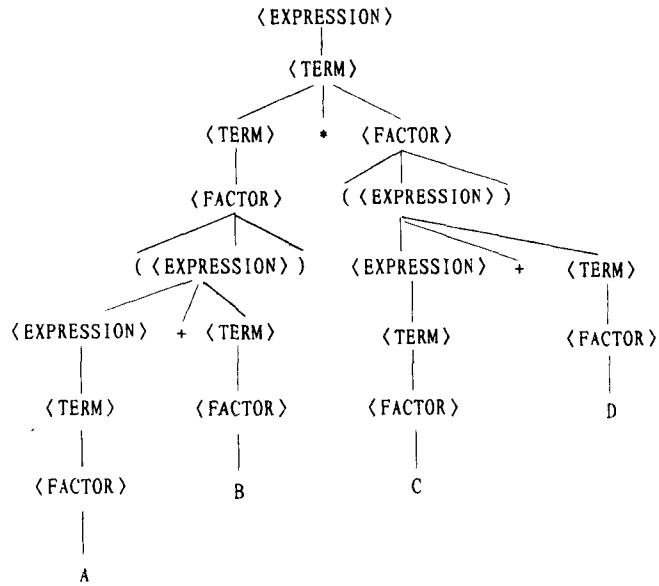

图 1.5 表达示(A+B) * (C+D)的语法树

在语法分析过程中，我们将单词进一步组合成大的语法类，如表达式、语句和过程等。语法分析程序（简称分析程序）可输出一棵语法树（或它的等价物）。在该语法树中，树的叶结点是语言的单词，而每一个非叶结点代表一个语法类类型。例如，对源语句(A+B) * (C+D)的分析能够产生语法类〈factor〉、〈term〉和〈expression〉，如图 1.5 中的语法树所示。在下一章，我们将介绍使用语法树，即用一组规则（也

就是文法)来精确定义源语言的方法。而语法分析程序则是使用这种定义源语言的文法来识别源程序中的各种结构。我们将这种识别过程称为分析。相应地我们将语法分析程序也简称为分析程序。在第四章中,我们将详细介绍各种类型的分析程序。

语义分析 语义分析程序使用由语法分析程序所产生的语法树。语义分析程序的功能是确定源程序的意义(语义)。虽然从概念上讲将源程序的语法与它的语义分开是合乎需要的,但是语法和语义分析程序则是以密切合作的方式工作的。例如,对于表达式(A+B)*(C+D),语义分析程序必须确定由算术操作符加和乘所规定的动作。当分析程序识别出一个操作符如"+"或"*"时,就调用一个语义程序,该语义程序执行由操作符所规定的动作。语义程序还要对两个相加的操作数是否已经作过声明,是否具有相同的类型(如果类型不同,则要做类型转换使其相同)以及两个操作数是否都已有值等进行检查。语义分析程序在执行语义分析任务时通常要使用编译程序所建立的各种表。

语义分析程序的动作还包括生成源程序的中间形式(即中间代码),对于表达式(A+B)*(C+D),其中间源代码可以是下列四元组:

$$(+,A,B,T_1)$$
$$(+,C,D,T_2)$$
$$(*,T_1,T_2,T_3)$$

第一条四元式可解释为"A 加上 B 并将结果存放到临时变量 T_1 中",第二条四元式为"C 加上 D,结果放到临时变量 T_2",第三条四元式为"T_1 乘以 T_2 结果存入临时变量 T_3"。也可以将一个中缀表达式转换为称作波兰表示的中间代码形式。使用这种方法,中缀表达式(A+B)*(C+D)将转换为等价的后缀波兰表达式(也称逆波兰表示)——AB+CD+*。在后缀波兰表达式中,自左向右的操作符排列顺序就是它们要被分别执行的顺序。另一类源程序的中间形式可以是一棵表示源程序分析过程的语法树(关于各种源程序的中间形式,我们将在第八章中做较为详细的介绍)。

代码生成 语义分析程序的输出被传送给代码生成程序。至于具体应用何种形式的中间代码,在很大程度上取决于在综合阶段如何来处理它们。代码生成程序通常是将源程序的中间形式转换为汇编语言或者机器语言。作为一个例子,对上述表达式的三条四元式的翻译能够产生下列单地址、单累加器的汇编指令序列:

LDA A	将 A 的内容加载到累加器
ADD B	将 B 的内容与累加器的内容相加
STO T_1	将累加器的内容存入临时存贮单元 T_1
LDA C	将 C 的内容加载到累加器
ADD D	将 D 的内容与累加器的内容相加
STO T_2	将累加器的内容存入临时存贮单元 T_2
LDA T_1	将 T_1 的内容加载到累加器
MUL T_2	将 T_2 的内容与累加器的内容相乘
STO T_3	将累加器的内容存入临时存贮单元 T_3

我们将在第十章中较详细地介绍代码生成的问题。代码生成程序的输出被传送到代码优化程序。

代码优化 进行代码优化的目的是要获得更高效的目标程序。在局部范围可能做的优化包括常数表达式的计算,操作符的某些性质诸如结合性、可交换性和分配性的使用以及公共子