编译程序设计原理与构造技术

史杏荣　万炳奎　编著

中国科学技术大学出版社

编译程序设计原理与构造技术

史杏荣　万炳奎编著

责任编辑：黄德　封面摄影：王瑞荣

合肥：中国科学技术大学出版社，1998 年 9 月

*

中国科学技术大学出版社出版发行

（安徽省合肥市金寨路 96 号，230026）

合肥市东方红印刷厂承印

全国新华书店经销

*

开本：787 × 1092 / 16　印张：21.25　字数：520 千字

1997 年 9 月第 1 版　1998 年 9 月第 1 次印刷

印数：1 - 5000 册

ISBN 7-312-01021-0 / TP · 210　定价：25.00 元

前　言

编译程序和操作系统构成了程序设计者和他所使用的计算机之间的基本接口界面。随着电子计算机技术的迅速发展，计算机不但用于科研和教学领域，而且已迅速普及到生产、管理和社会生活的各个领域。现在，计算机使用者不仅要掌握计算机的基本原理、汇编语言和某些高级语言，还应具有一定的计算机软件基础知识和基本技术，以提高应用软件方面的研制能力。

“编译程序设计原理与构造技术”主要介绍编译程序设计的基本原理、实现方法和构造技术。第一章介绍语言、编译程序的基本结构及其基本概念。第二章简要地阐述形式语言的基本知识，即编译程序的理论基础知识。第三章描述词法分析、正规表达式、词法分析器的设计，以及有穷自动机、一个词法分析器的自动生成器 lex 及其应用。第四章系统地讨论了自上而下和自下而上的各种语法分析算法与技术。例如，适用于手工编程实现的递归下降分析器，分析算术表达式的算符优先分析方法， LR 分析算法，并详细介绍了语法分析器的自动生成器 yacc 程序及其应用。第五章讨论了各类语句的语法制导翻译和中间代码生成的翻译规则。第六章讨论了支持程序运行的环境、存储分配策略与存储空间的组织、符号表的组织等编译程序的构造技术。第七章阐述目标代码的生成及其优化技术。第八章讨论了循环优化、数据流分析及其在代码优化中的应用。第九章以 C 语言和 FORTRAN H 等著名语言的编译程序为例介绍编译程序的结构、规划及其构造技术。

在材料的组织上，重点在于介绍编译程序设计的基本原理、方法和构造技术，不拘泥于具体的实现细节；既注意了最经典、最广泛使用的编译技术，又反映了七十年代以来一些最重要的研究成果。在词法分析和语法分析方面，特别注重分析器的自动生成。在叙述上，力求由浅入深，循序渐进，便于自学。各章的后面均附有习题和思考题，供练习使用。

构造编译程序的概念和技术可应用于一般的软件设计之中。例如，构造词法分析器的串匹配技术已用于正文编辑器、信息检索系统和模式识别程序；上下文无关文法和语法制导定义已用于创建许多诸如排版、绘图系统的小语言。

本教材是在“软件技术基础”（ II ）课程中使用了 3 年的“编译原理”讲义的基础上编写的，旨在为工科高等院校电子类专业的本科生、成教院系统的计算机及其应用专业的本科生和大专生讲授编译程序的设计原理和构造技术。本书既介绍了编译程序设计的基本原理与常用算法，又注重了实现方法和构造技术，可作为大学高年级的学生、研究生和科技人员学习和掌握编译技术的参考书，也可作为计算机应用的培训教材。

本书在编写过程中得到了信息处理中心刘政凯教授的热情支持和关心，并对本书提出了许多宝贵意见，在此表示衷心的感谢。

限于编者水平，错误和不妥之处在所难免，恳请读者批评指正。

编者

1998 年 9 月于中国科学技术大学

目 录

第一章 引 论 1
1.1 程序设计语言 1
1.2 编译过程概述 3
1.2.1 编译程序模型 4
1.2.2 分 析 4
1.2.3 综 合 6
1.2.4 符号表管理 10
1.2.5 错误检测和处理 11
1.3 编译器的构造 11
1.3.1 阶段的分组 11
1.3.2 编译器的运行环境 12
1.3.3 构造编译器的工具 15
第二章 形式语言的基本知识 17
2.1 引 言 17
2.1.1 语法树 17
2.1.2 规 则 18
2.1.3 推 导 18
2.2 字母表、符号串及其集合的运算 19
2.2.1 字母表和符号串 20
2.2.2 符号串及其集合的运算 20
2.3 文法和语言的形式定义 22
2.3.1 文法的形式定义 22
2.3.2 推导的形式定义 24
2.3.3 短语、直接短语和句柄 26
2.4 分析树和二义性 26
2.4.1 分析树(Parse tree) 26
2.4.2 子树与短语 28
2.4.3 文法的二义性 29
2.5 文法和语言分类 30
第三章 词法分析 33
3.1 词法分析器的功能 33

3.1.1 词法分析器的作用 33
3.1.2 单词类别、构词规则和属性 34
3.1.3 词法错误 35
3.2 输入缓冲区 36
3.3 正规表达式与正规集 38
3.3.1 正规表达式与正规集 38
3.3.2 正规定义 39
3.4 词法分析器的设计 40
3.4.1 预处理 40
3.4.2 单词符号的识别 41
3.4.3 状态转换图 42
3.4.4 实现转换图 44
3.5 有穷自动机 51
3.5.1 确定有穷自动机 51
3.5.2 不确定有穷自动机 52
3.5.3 NFA N 到 DFA M 的转换 54
3.5.4 从正规式构造 NFAN 57
3.5.5 确定有穷自动机的化简 59
3.6 词法分析器的自动生成器 61
3.6.1 lex 源程序 62
3.6.2 lex 的实现 66
第四章 语法分析 68
4.1 自上而下分析 68
4.1.1 递归下降分析 69
4.1.2 预测分析器 75
4.1.3 预测分析器的转换图 78
4.1.4 非递归预测分析器 80
4.1.5 构造预测分析表 83
4.1.6 LL(1)文法 87
4.2 自下而上分析 88
4.2.1 句 柄 89
4.2.2 剪句柄 90
4.2.3 用堆栈实现移进一归约分析 91
4.3 算符优先分析 93
4.3.1 使用算符优先关系的移进一归约分析 95
4.3.2 算法优先关系表的构造 97
4.3.3 优先函数 101
4.3.4 算符优先分析的错误恢复 103

4.4 LR 分析器 103
4.4.1 LR 分析算法 104
4.4.2 SLR 分析表的构造 106
4.4.3 规范的 LR 分析表的构造 114
4.4.4 LALR 分析表的构造 118
4.5 二义文法的应用 125
4.6 语法分析器的生成器 129
4.6.1 yacc 源程序 130
4.6.2 使用二义文法的 yacc 应用示例 132
第五章 语法制导翻译和中间代码生成 137
5.1 概 述 138
5.1.1 语法制导定义 138
5.1.2 语法制导翻译 143
5.2 逆波兰表示法 146
5.3 语法树 147
5.4 三地址代码 149
5.4.1 三地址语句的种类 150
5.4.2 三地址语句的实现 150
5.5 简单算术表达式和赋值语句到四元式的翻译 153
5.6 布尔表达式到四元式的翻译 169
5.6.1 数值表示法的语法制导翻译 170
5.6.2 布尔表达式的控制流翻译 172
5.7 控制语句的翻译 176
5.7.1 标号和转移语句 176
5.7.2 控制流语句的翻译 179
5.7.3 case 语句的翻译 181
5.8 说明语句的翻译 183
5.8.1 简单变量说明语句的翻译 184
5.8.2 数组说明语句的翻译 185
5.8.3 记录结构说明语句的翻译 186
5.9 数组元素和记录域的引用 189
5.9.1 数组元素引用 189
5.9.2 记录域的引用 191
5.10 过程调用语句的翻译 192
5.11 自上而下分析语法制导翻译概述 193
第六章 运行环境 199
6.1 源语言问题 199
6.1.1 过 程 200

6.1.2 活动树 200
6.1.3 控制栈 202
6.1.4 说明的作用域 203
6.1.5 名字的联编 203
6.2 存储组织 204
6.2.1 运行时内存的划分 204
6.2.2 活动记录 205
6.2.3 编译时局部数据的安排 206
6.3 存储分配策略 207
6.3.1 静态分配 207
6.3.2 栈式存储分配 209
6.3.3 堆式存储分配 213
6.4 访问非局部名字 214
6.4.1 块 214
6.4.2 没有嵌套过程的词法作用域 215
6.4.3 具有嵌套过程的静态作用域 216
6.4.4 动态作用域 221
6.5 参数传递 222
6.5.1 值调用 222
6.5.2 引用调用 223
6.5.3 复制恢复 224
6.5.4 换名调用 225
6.6 符号表 226
6.6.1 符号表的表项 227
6.6.2 线性符号表 228
6.6.3 散列符号表 229
6.6.4 作用域信息的表示 231
第七章 代码生成 235
7.1 目标机器 236
7.2 运行时的存储管理 238
7.2.1 静态分配 239
7.2.2 栈式分配 240
7.2.3 名字的运行地址 242
7.3 基本块和流图 243
7.3.1 基本块 243
7.3.2 基本块的变换 244
7.3.3 流 图 245
7.4 下次引用信息 246

7.5 一个简单的代码生成器 247
7.5.1 寄存器和地址描述符 248
7.5.2 代码生成算法 248
7.5.3 函数 getReg 249
7.5.4 为其它类型的语句生成目标代码 250
7.6 寄存器分配 251
7.6.1 全局寄存器分配 252
7.6.2 执行代价的节省 252
7.6.3 寄存器分配的图着色方法 254
7.7 基本块的 dag 表示及其应用 255
7.7.1 dag 的构造 257
7.7.2 dag 的应用 258
7.7.3 数组、指针和过程调用 259
7.8 从 dag 生成目标代码 260
7.8.1 重排序 261
7.8.2 dag 的启发式排序 261
7.8.3 树的最优代码 263
7.9 窥孔优化 268
第八章 代码优化和数据流分析 272
8.1 控制流分析和循环查找算法 273
8.1.1 程序流图和循环 273
8.1.2 必经结点集 274
8.1.3 查找循环算法 276
8.1.4 可归约流图 277
8.2 到达一定值与引用一定值链 278
8.2.1 到达一定值数据流方程 278
8.2.2 到达一定值的迭代算法 279
8.2.3 引用一定值链（ud 链） 281
8.3 循环优化 282
8.3.1 代码外提 282
8.3.2 强度削弱 284
8.3.3 删除归纳变量 285
8.4 活跃变量与定值一引用链（du 链） 289
8.4.1 活跃变量的数据流方程 289
8.4.2 活跃变量数据流方程的迭代算法 290
8.5 删除全局公共子表达式 292
8.5.1 可用表达式及其数据流方程 292
8.5.2 可用表达式数据流方程的迭代算法 294

8.5.3 删除全局公共子表达式 295
8.6 复写传播 295
8.6.1 复写传播算法 295
8.6.2 代码优化的综合考虑 297
第九章 编译器的设计与实现 301
9.1 编译器的规划 301
9.1.1 源语言问题 301
9.1.2 目标语言问题 302
9.1.3 性能标准 302
9.2 编译器的开发方法 302
9.2.1 编译器的书写语言 303
9.2.2 自 展 303
9.3 编译器的开发环境 306
9.4 测试和维护 308
9.5 几个编译器简介 308
9.5.1 Pascal 编译器 308
9.5.2 C 语言编译器 309
9.5.3 FORTRAN H 编译器 310
参考文献 313

第一章 引 论

术语编译是指将一个源语言表示的算法到目标语言表示的算法的一个等价变换，如图1.1所示。通常，源语言是面向人的，而目标语言则是面向机器的。

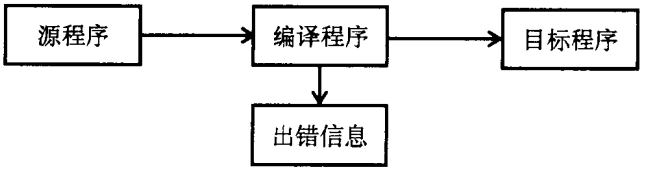


图1.1 编译器

1.1 程序设计语言

语言是人类思考问题和交流思想的工具。在计算机应用领域，程序设计语言则用来描述要解决问题的算法，并充当了人与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将能提高计算机软件的开发效率和对问题的表达能力。

在计算机发展初期（1951— 1958年），人们直接使用机器语言编写程序，即用二进制代码直接编写程序（指令和数据），且为它们分配存储空间。机器语言是第一代语言(1GL)，它很不直观，难写，难读，且容易出错。出错后又难于检查，即使查出了错误也难于修改。程序设计和调试程序所花的时间往往比计算机实际运行的时间多出数百倍，甚至数千倍以上。

在1959—1964年，出现了第二代语言（2GL）—汇编语言。人们用助记符取代二进制指令代码；用符号代表存储单元的地址和常数；并允许使用宏指令对应一组机器指令，以完成一些特定的功能。用汇编语言编制的程序不能在计算机上直接运行，必须由汇编程序（Assembler）将汇编语言的源程序翻译成机器语言的目标代码，然后才能运行。汇编程序为源程序的指令和数据分配存储空间；将指令的助记符翻译成其目标代码，并代人分配的存储单元的地址。

机器语言和汇编语言是低级语言，它们是面向机器的语言。低级语言使用起来仍不方便，且程序设计的效率很低，不便于交流和软件移植。为了进一步提高语言的描述能力，方便用户，人们参照数学语言设计了一类便于描述算法的语言，如ALGOL，FORTRAN等。目前，市场上已有数百种高级程序设计语言，它们在应用上有各自不同的侧重面。例如，FORTRAN适用于数值计算，COBOL便于事务处理，C语言则组合了结构程序设计语言和汇编语言的先进特征，利于系统软件开发。

高级语言（ 1975—1970 年）是第三代程序设计语言，因为这类语言完全摆脱了机器指令的约束，用它编制的程序更接近于自然语言和习惯上对算法的描述，因而又称之为面向用户的语言。

高级语言与低级语言相比，它有下述优点：

i ）**独立于计算机，移植性好**。因为用高级语言研制软件，不依赖于具体计算机的实现的细节，所以高级语言方便了用户，促进了计算机的使用与推广。此外，在一般情况下，使用高级语言编制的程序可以不加修改地或进行很少量的修改（主要是数据类型及其长度）就能在不同计算机上编译连接，然后就能运行。

ii ）**自动分配存储空间**。用户在编制程序时仅定义变量名，而由编译程序为它们分配存储空间和进行存储空间的管理。用户也不必知道各种类型数据在外部格式和内部格式之间的转换以及它们在内存中如何存储的实现细节。这些任务都由编译程序和系统进行处理，这大大简化了软件的开发过程。

iii ）**丰富的数据结构和控制结构**。例如， C 语言允许用户定义结构数据类型及其数组，使用结构数据能更好地表达数据本身及其之间的内在关系。控制结构有条件语句、循环语句、多分支选择语句（ case 语句）和函数调用等。这些结构改善了程序的风格，便于采用结构化程序设计技术和模块化方法来开发程序，降低了程序的复杂性，提高了程序的可靠性和软件开发效率，从而也降低了开发费用。

iv ）**高级语言更接近于数学语言和工程语言，因而比较直观、自然和易于理解**。高级语言程序易读，易写，易于交流和维护。由于易于理解，因而有利于防止程序出错和便于验证其正确性，一旦发现错误也易于修改。

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，提高了软件的开发效率、可靠性和可维护性；较大幅度地缩短了软件的开发周期，降低了软件的开发成本和维护费用。这在计算机软件发展史上是一次具有重大意义的进展。

前三代语言都属于过程型语言（ Procedural language ），它们描述了要解决问题的算法，即具体的操作步骤。从 1971 年开始，出现了另一类超高级语言（ Very—high languages ），它们是第四代语言（ fourth—generation language （ 4GL ）），是面向非过程的语言（ Nonprocedural—oriented language ）。例如，用于数据库查询的 SQL 语言，报表产生器，等等，它们是专门用于描述某个应用领域问题的专用语言。另一类是面向对象的语言（如 C++语言）和计算机辅助软件工程 CASE （ Computer—aided software enginerring ）。它们是面向人的语言（ Human—oriented language ）。第四代语言的另一个杰出代表是 Prolog （ Programming in logic ），它是一种智能逻辑推理语言。

众所周知，用高级语言编制的程序，计算机是不能立即执行的。因为计算机只能识别和执行机器指令，所以高级语言的源程序必须通过一个“翻译程序”的加工处理，使之转换为等价的机器语言程序，计算机才能执行之。通常称这种翻译程序为编译程序或编译器（ Compiler ）。

编译程序是计算机系统软件的重要组成部分。本书介绍编译程序的结构、设计的一般原理及其构造技术。

1.2 编译过程概述

翻译程序／翻译器（ Translator ）扫描输入的源程序，然后将该源程序变换为与其等价的目标程序。源程序是用汇编语言或高级程序设计语言编制的，而目标程序则是用目标语言表示的。

如果翻译的源程序是汇编语言源程序，且目标语言是机器语言，那么，该翻译程序就是汇编程序／汇编器（ Assembler ）。

倘若翻译程序将一个用高级程序设计语言编制的源程序翻译为目标机器的机器语言或汇编语言的目标程序，则称该翻译程序为编译程序／编译器。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，数据也是在目标程序运行时处理，目标程序可以重复运行多次。图 1.2 说明了高级语言源程序的编译和运行过程。

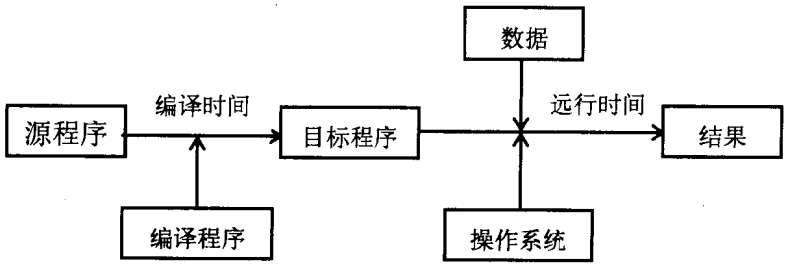


图 1.2 高级语言程序的编译和运行

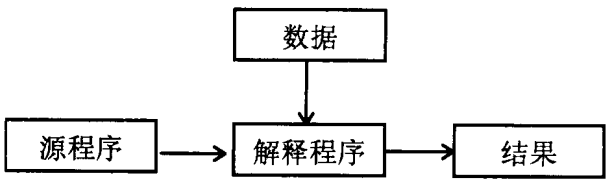


图 1.3 源程序解释过程图

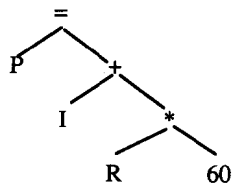


图 1.4 语句 P=I+R*60 的语法树

另一类翻译程序是解释程序，它同时处理源程序和数据。解释程序解释源程序并执行之，但不为其生成目标程序。例如， BASIC 解释程序就是一例，图 1.3 说明了解释程序的执行过程。解释程序通常从源程序中读入一条源语句，对其进行语法分析并生成其中间代码，然后解释并执行该段中间代码程序。例如，对于一条赋值语句，解释程序为其生成一棵语法分析树（ Syntax tree ），如图 1.4 所示，然后使用递归后序遍历算法遍历语法分析树，并执行结点规定的运算或操作。例如，图 1.4 所示的语法分析树的根结点是赋值运算符，则将其右结点变量的值赋值给其左结点标识的变量中；但因其右结点是一棵子树，则应先遍历右子树，以计算其右边的算术表达式的值，并将其返回值存放在 P 标识的存储单元中。解释程序在遍历根结点的右子树时，该右子树的根结点是加法运算符，则应将其左、

右结点代表的值进行加法运算，并将计算的和作为右子树根结点的值返回。这样，解释程序按递归后序遍历算法遍历该语法分析树，先计算 R*60 表达式的值，再将该表达式 R*60 的值和 I 的值进行加法运算，最后将表达式 I + R*60 的值存入 P 标识的存储单元。

1.2.1 编译程序模型

为一个语言编制编译程序是一个相当复杂的任务，编译过程的复杂性在很大程度上取决于源语言，图 1.5 示例了一个编译程序的基本模型。

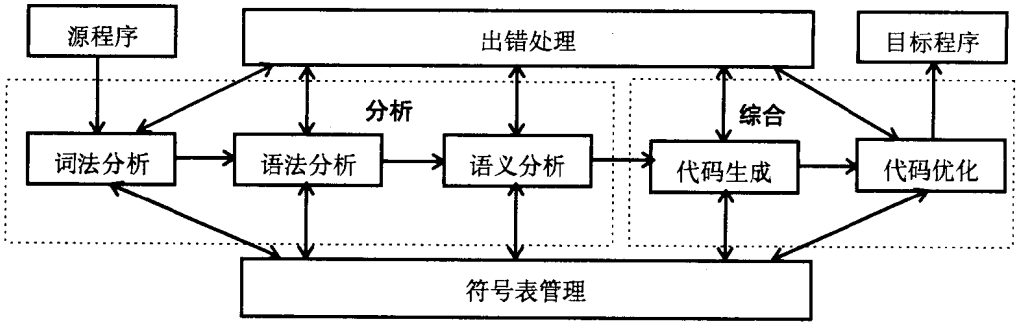


图 1.5 编译程序的结构

编译程序必须完成两个主要任务:

ⅰ）**分析** 分析部分将源程序分解为源程序的各个基本组成部分，揭示源程序的结构和基本数据类型、数据结构，决定它们的语义，并建立源程序的中间形式表示。

ⅱ）**综合** 综合部分接收分析部分产生的中间形式表示，并根据具体的目标计算机系统生成目标代码并对代码进行优化处理。

1.2.2 分 析

在编译期间，分析由三个阶段组成:

ⅰ）**线性分析（词法分析）** 从左到右地读组成源程序的字符流，并分离、识别出一个个记号（ Token ）， Token 是有确定含义的字符序列。

ⅱ）**层次分析** 根据语言的语法规则，由 Token 流形成语言的各类语法单位，如短语、句子、程序段和程序。

ⅲ）**语义分析** 它完成一些语义检查，以保证程序的各部分能有意义地结合在一起。

1．词法分析（ Lexical Analysis ）

一个源程序实际上是由字母、数字、运算符、分隔符和某些专用符号组成的字符串。一个语言的源程序包含了许多该语言的基本的语言结构成份，如关键字、变量名、标号名、常数、运算符和分隔符等等。所以编译程序首先要识别出这些不同类型的基本语言结构成份。例如， FORTRAN 语句

DO 150 I=1,00

由关键字 DO 、标识符 I（变量名）、符号 =（赋初值运算符）、常数 1 和 100 、标号 150

和分割符“,”（逗号）等单词符号组成。这些单词是组成上述 FORTRAN 的 DO 语句的基本符号。单词符号是语言的基本组成成份，是人们理解和编写程序的基本要素。识别和理解这些要素毫无疑问也是翻译的基础。

词法分析器的任务：输入源程序，对构成源程序的字符串进行扫描和分解，识别一个个单词符号。在词法分析阶段，词法分析器按语言的构词规则识别一个个单词符号，形成记号（ Token ）流。 Token 是单词类别的整数编码，它是一类单词别种的代表，通常以一对偶的形式表示，例如 < ID ， I 所在符号表项的地址 > 。其中， ID 是标识符的整型数内部码，标识单词类别；第二项给出读单词 I 在符号表中的入口地址，以标识具体单词 I 。

2 ． 语法分析（ Syntax Analysis ）

层次分析也叫做语法分析。语法分析程序从词法分析程序取得源程序（记号流，即单词串），并将源程序的记号分组形成语法短语。语法分析所依循的是语言的语法规则。源程序的语法短语常用分析树（ Parse tree ）表示。

程序的层次结构通常使用递归的语法规则表示。例如，表达式的语法规则可定义为：

i ）标识符是一个表达式;

ii ）数是一个表达式;

iii ）如果 e1 和 e2 都是一个表达式，则

e_1+e_2

e_1*e_2

e_1/e_2

(e_1)

也都是表达式。

类似地，赋值语句可定义为:

ID = e; 注：分号（;）是语句的结束标记。其中， ID 是标识符， e 是表达式。

这样，根据上述定义的语法规则，语法分析器为赋值语句

p =x0+v*60;

产生如图 1.6 所示的分析树。

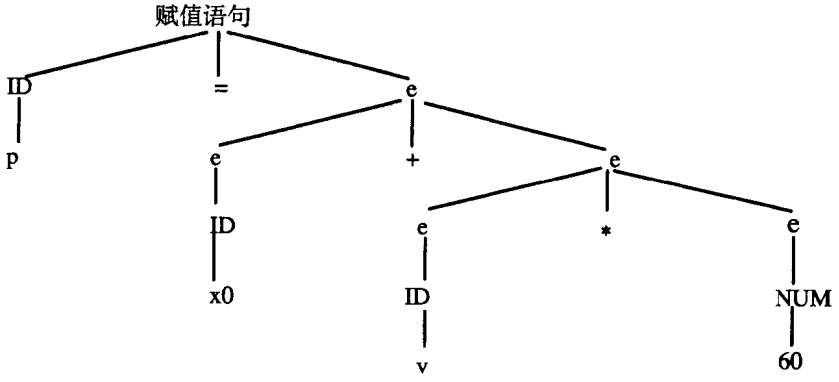


图 1.6 赋值语句 p=x0+v*60 的分析树

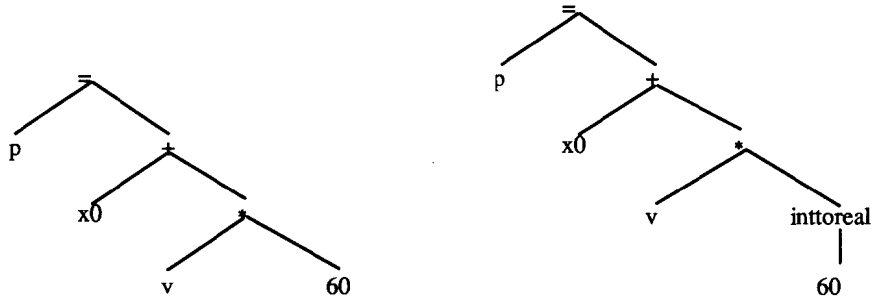
通常，标识符和常数由词法分析器根据标识符和常数的构词规则识别。

3．语义分析（Semantic Analysis）

语义分析器使用语法分析器产生的分析树／语法树，其功能是确定源程序的意义（语义）。在语义分析阶段，检查源程序的语义错误，并为以后的代码生成阶段收集类型等属性信息。它使用语法分析阶段确定的层次结构来标识表达式的运算符、运算对象以及语句。

语义分析的另一个重要任务就是类型检查，例如检查每个运算符的运算对象的类型，判断其是否合法。例如，对于表达式 v*60，语义分析器检查符号表中变量 v 的类型，若 v 是整型变量，则表达式 v*60 有效。倘若 v 是实型，则根据语言是否允许混合类型运算，如果允许，应将整型数 60 转换为实数（ 60.0 ），然后参加运算；否则报告类型不一致的错误信息。

分析树描述了输入的语法结构，为了节省存储空间，大多编译器使用语法树（ Syntax tree ）作为其内部表示。语法树是分析树的压缩形式表示，其中算符作为内部结点（非叶子结点）出现，它的运算对象作为它的子结点。图 1.7 显示了语义分析插入类型转换的 p=x0+v*60 的语法树。



(a) p=x0+v*60 的语法树　　(b) 语义分析插入整型到实型类型转换的语法树

1.7 p=x0+v*60 的语法树

1.2.3 综 合

综合阶段从分析阶段得到带语义的语法树或其它源程序的中间表示，并根据制定的源语言到目标语言的对应关系，进行综合加工处理，从而产生与源程序等价的目标程序。通常，综合也分几个阶段进行:

i ）中间代码生成;

ii ）代码优化;

iii ）目标代码生成。

1．中间代码生成

中间代码产生器（ Intermediate code generator ）对各类不同语法范畴按语言的语义进行初步翻译工作，产生源程序的中间代码表示。“中间代码”是一种结构简单、含义明确的符号系统，它也可以被认为是一个抽象机的程序。中间代码表示应具有下述两个性质:

i ）容易产生;

ii ）容易翻译成目标程序。

例如，许多编译器采用了一种与“三地址指令”非常近似的“四元式”作为中间代码。例如， FORTRAN 语句 p=x0+v*60 可以翻译成下列四元式序列或三地址语句序列:

```
（*， v ， 60.， temp1 ）          ；  temp1=v*60.
（+， x0 ， temp1 ， temp2 ）      ；  temp2=x0+temp1
（=， temp2 ， 0 ， p ）           ；  p=temp2
```

将各类语法范畴翻译成中间代码所依据的是语言的语义规则。一般而言，中间代码是一种独立于具体硬件的符号系统。除四元式外，常用的中间代码形式还有三元式、间接三元式和逆波兰表达式等。

2．代码优化

优化的任务在于对以前阶段产生的中间代码进行加工变换，以期在最后阶段产生更为高效（省时间和空间）的目标代码。优化的主要方面有:

i ）公共子表达式的提取（ Common subexpression elimination ）

例如，下列赋值语句

```
ix=j*k+16
iy=j*k+26*l-m*l
```

j*k 是公共子表达式，可以只计算一次。经优化处理后，产生下列三地址语句序列:

```
temp1=j*k
ix=temp1+16
temp2=26-m
temp3=l*temp2
iy=temp1+temp3
```

ii ）循环优化（ Loop optimization ）

考虑下列循环语句:

```
for(k=1; k<101; k++) {
    m=i+10*k;
    n=j+10*k;
    }
```

中间代码产生器生成下列三地址语句序列:

```
1       k=1
2       if(k>100)goto 9
3       temp1=10*k
4       m=i+temp1
5       temp2 = 10*k
6       n=j+temp2
7       k=k+1
8       goto  2
9                                  ;注: 后续语句的四元式的序号
```

经循环优化处理后，产生下列三地址语句序列:

```
1       m=i
2       n=j
3       k=1
4       if(k>100)goto  9
5       m=m+10
6       n=n+10
7       k=k+1
8       goto  4
9                                   ;注: 后续语句的四元式的序号
```

对于优化前的三地址语句序列，在循环中执行了 300 次加法和 200 次乘法运算；优化后的中间代码仅执行了 300 次加法运算，明显地改善了程序的运行效率。

循环优化对目标程序的质量有至关重要的影响。

iii ）算符归约（ Reduction in strength ）

在计算机系统中，计算一个实数的幂次方（如 x^2 ）比乘法运算（ x*x ）要慢得多；同样，乘法／除法运算又要比加法和移位运算慢了多。所以，在某些情况下，可用乘法运算（ x*x ）取代幂运算（ x^2 ）；用加法和移位运算实现简单的乘法运算，如用（ x<<1+x)<<1 完成 6*x 的乘法运算，用 x+x 取代 2.0*x 的运算。

iv ）全局优化（ Global optimization ）

上述几种优化是局部优化（ Local optimization ），更复杂的代码优化是全局优化，则需使用数据流分析技术。

大部分编译器完成不同程度的优化，能实现大多数优化的，叫做“优化编译器”，但编译时间相当一部化消耗在优化上。简单的优化也可以使目标程序的运行时间大大缩短，而编译速度并没有降低太多。

代码优化遵循程序的等价变换规则。所谓等价，是指不改变程序的运行结果。

3．目标代码生成

编译的最后一个阶段是目标代码生成，图 1.8 说明了代码生成器（ Code generator ）在编译器中的位置。

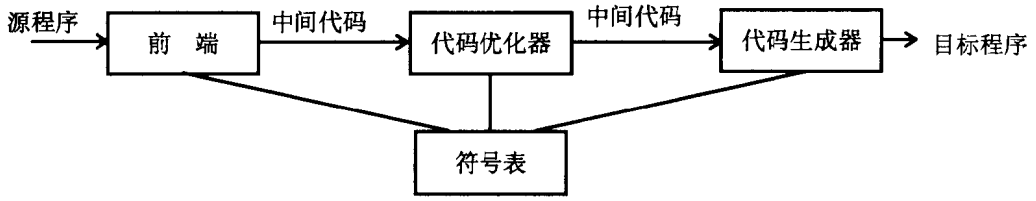


图 1.8　代码生成器的位置

目标代码生成器将中间代码（经优化处理之后）变换成特定机器上的绝对指令代码、可重定位的指令代码或汇编指令代码。它实现了编译器最后的翻译工作，它的工作依赖于硬件系统的体系结构和 CPU 的指令系统。代码生成器输出的代码必须正确，且必须是高质