编译器设计原理

谌志群　王荣波　黄孝喜

2019 年 12 月

内容简介

本书系统介绍了编译器构造的基本原理和一些经典实现技术。主要内容包括形式文法和形式语言理论、基于有限自动机的词法分析技术、自顶向下和自底向上的语法分析技术、基于语法制导翻译的语义分析和中间代码生成、中间代码优化、目标代码运行时刻环境的组织、目标代码生成等。本书在内容选取上，理论部分深入浅出，技术与算法部分简明扼要，为帮助读者理解，特别重视实例的选取和剖析。为适应“新工科”建设要求，本书专门讨论了编译技术在实际工程领域的应用，设计了几个与新兴产业紧密结合的工程案例。附录部分给出了一个简单模型语言编译器实例，读者可通过阅读编译器源代码，对编译器实现有更深刻理解。

本书可作为计算机相关本科专业编译原理与编译技术的教材，也可以供其它专业学生及工程技术人员参考。

前　言

近年来，随着人工智能、物联网、云计算、区块链等新兴信息技术的快速发展和广泛应用，计算系统呈现出新的特征，形成了嵌入式计算、移动计算、并行计算、服务计算等多计算平台并存和融合的态势。新形势下对计算机类专业人才培养的要求也逐渐从传统的“程序设计能力”培养向更为重要的“系统设计能力”培养转变。系统设计能力体现在能深刻理解计算机系统内部各软/硬件部分的关联关系和逻辑层次，了解计算机系统呈现的外部特征及与人和现实世界的交互模式，开发以计算机技术为核心的高效应用系统。

2010 年，教育部高等学校计算机类专业教学指导委员会（以下简称教指委）成立“计算机类专业系统能力培养研究组”，开始组织实施计算机类专业学生系统能力培养的研究和实践。2016 年，教指委启动第一批系统能力培养改革试点校申报工作，遴选出包括清华大学、浙江大学在内的 8 所示范校和哈尔滨工业大学等 43 所试点校。经过几年的探索实践，“使学生具备设计实现一个 CPU、一个操作系统、一个编译器的能力是系统能力培养的主要目标”逐步成为计算机专业教育领域的共识。“编译原理”作为一门计算机专业基础课，是系统能力培养体系中不可或缺的组成部分。

本书作为计算机及相关专业本科生“编译原理”课程的教材，目的是讲授编译器设计的基本原理和实现编译器的主要方法。全书共分十章，第一章“编译器概述”概要介绍高级程序设计语言的发展史和几种语言翻译程序，讨论了编译过程的主要步骤及每个步骤需完成的主要功能，介绍了编译器结构及编译器构造的主要方法；第二章“形式文法和形式语言”首先通过与自然语言的类比，介绍形式文法和形式语言与自然语言的区别，然后介绍了形式文法和形式语言的定义及相关的几个重要概念，如推导、归约、句型、句子、二义性等。在简要介绍形式文法和形式语言分类的基础上，详细介绍了上下文无关文法句型的分析方法；第三章“词法分析”主要介绍了正规表达式和有限自动机这两个数学模型，讨论了正规表达式、有限自动机和正规文法作为正规语言描述工具的等价性，重点是正规表达式到有限自动机的转换，非确定有限自动机的确定化和最小化。简要介绍了词法分析器生成工具——LEX；第四章“语法分析”概述了自顶向下和自底向上两类语法分析方法的特点，详细介绍了自顶向下的 LL（1）分析和自底向上的 LR 分析方法。LL（1）分析部分重点讨论了递归下降子程序法和非递归的预测分析器实现方法。LR 分析部分重点讨论了 LR 分析器模型及几种 LR 分析表的构造方法。简要介绍了语法分析器生成工具——YACC；第五章“语法制导翻译技术”介绍了综合属性、继承属性、语法制导定义、依赖图、属性计算顺序等几个概念，详细讨论了基于 S-属性定义的自底向上属性计算方法和基于 L-属性定义的深度优先属性计算方法，这两类方法是实现语义分析和中间代码生成的主要方法；第六章“语义分析与中间代码生成”主要介绍了基于语法制导翻译技术的语义分析和中间代码生成的实现方法，包括说明语句的处理、中间表示形式、赋值语句的翻译、布尔表达式和控制流语句的翻译等；第七章“代码优化”简要介绍了代码优化的定义与分类，重点介绍与机器无关的针对中间代码的优化技术，包括针对基本块的局部优化、针对循环体的循环优化等；第八章“目标代码运行时刻环境的组织”详细讨论了目标代码运行时刻内存空间的管理与分配，介绍了运行时刻内存

空间的典型划分，分别介绍了静态存储分配、栈式存储分配和堆式存储分配这 3 种内存空间分配策略。针对几种典型高级语言的语言结构，分别介绍了非局部名字的访问策略；第九章“目标代码生成”简要介绍了目标代码生成的主要任务，以示例方式讨论了静态存储分配和栈式存储分配的代码实现方法。介绍了下次引用信息和活跃信息的获取，及寄存器描述器和地址描述器的作用，在此基础上介绍了一个针对基本块的简单代码生成算法；第十章“编译技术应用”介绍了几个编译技术应用实例；附录 A“SMini——一个简单模型语言编译器”定义了一个简单模型语言 S 语言，介绍了 S 语言编译器的结构和实现方法，供学习者参考。限于篇幅本书不可能深入介绍构造一个产品级编译器涉及的所有算法和技术，但是通过学习本书，读者可掌握开发编译器必需的基础知识和一些构造编译器的经典算法，学会建立一个相对简单的模型语言的编译器。

美国哥伦比亚大学的 Alfred V.Aho 教授在其经典名著《编译器：原理、技术与工具》的前言部分写道“编写编译器的原理和技术具有十分普遍的意义，以至于在每个计算机科学家的研究生涯中，本书中的原理和技术都会反复被用到”。对于学习“编译原理”这门课的学生来说，除了可以通过课程学习掌握一些具体的基本理论、技术和算法之外，更重要的是能培养学生具备计算机学科通用的问题求解和系统设计能力，如问题抽象与形式化描述、复杂问题算法分析与设计、自顶向下逐步求精、自底向上分步求解、软硬件协同设计等。有了这些问题求解方法和系统设计能力将使得学生在计算机专业领域内“具备持续竞争力”。

编者

2019 年 12 月于杭州

目　　录

第一章　编译器概述 1

1.1 程序设计语言发展史 1
1.2 语言翻译器 3
1.3 编译器结构 4
1.4 编译器构造方法 9
1.5 小结 11
习题 12

第二章　形式文法和形式语言 13

2.1 形式语言与自然语言 13
2.2 文法和语言的形式定义 14
2.2.1 一个自然语言的例子 14
2.2.2 字母表和符号串 16
2.2.3 语言的非形式定义 17
2.2.4 语言的运算 18
2.2.5 语言的描述 19
2.2.6 文法的形式定义 19
2.2.7 推导与归约 23
2.2.8 语言与文法 24
2.3 文法和语言的分类 27
2.4 上下文无关文法的句型分析 29
2.4.1 用上下文无关文法描述高级语言 29
2.4.2 句型推导与分析树 31
2.4.3 句子、文法和语言的二义性 34
2.4.4 二义文法的改造 36
2.5 小结 38
习题 38

第三章　词法分析 42

3.1 词法分析程序的设计 42
3.2 单词的描述——正规表达式 45
3.3 单词的识别——有限自动机 47
3.3.1 有限自动机的定义 47
3.3.2 NFA 到 DFA 的转换 51
3.3.3 DFA 的最小化 54
3.4 正规表达式与有限自动机的等价性 60
3.5 词法分析程序的自动构造工具 64
3.6 小结 66
习题 67

第四章　语法分析 71

4.1 语法分析概述 71
4.2 自顶向下语法分析方法 73
4.2.1 不确定的自顶向下分析 73

4.2.2 确定的自顶向下分析 *75*
4.2.3 非 LL（1）文法到 LL（1）文法的等价变换 *81*
4.2.4 无回溯递归下降分析法 *87*
4.2.5 非递归预测分析器 *90*
4.2.5 预测分析中的错误处理 *94*
4.3 自底向上语法分析——LR 分析 96
4.3.1 自底向上语法分析的关键——识别句柄 *96*
4.3.2 自底向上语法分析的实现方法——移进-归约法 *97*
4.3.3 LR 分析器模型 *99*
4.3.4 构造 LR（0）分析表 *102*
4.3.5 构造 SLR（1）分析表 *112*
4.3.6 LR（1）和 LALR（1）分析表的构造 *115*
4.4 语法分析程序的自动构造工具 121
4.5 小结 123
习题 124

第五章 语法制导翻译技术 129

5.1 语义分析概述 129
5.2 语法制导定义 129
5.3 S-属性定义及其自底向上的属性计算 135
5.4 L-属性定义及其深度优先的属性计算 137
5.5 小结 142
习题 142

第六章 语义分析与中间代码生成 145

6.1 类型检查 145
6.2 说明语句的处理 148
6.3 中间语言 155
6.4 赋值语句的翻译 159
6.5 布尔表达式和控制流语句的翻译 165
6.6 回填技术 176
6.7 小结 183
习题 184

第七章 代码优化 185

7.1 代码优化概述 185
7.2 基本块与局部优化 188
7.3 控制流分析与循环优化 194
7.4 数据流分析与全局优化 199
7.5 小结 200
习题 201

第八章 目标代码运行时刻环境的组织 203

8.1 目标代码运行时刻环境 203
8.2 源语言相关问题讨论 204
8.3 运行时刻内存空间的组织 207
8.4 运行时刻内存空间分配策略 209

8.4.1 静态存储分配 *209*
8.4.2 栈式存储分配 *211*
8.4.3 堆式存储分配 *213*
8.5 对非局部名字的访问 214
8.5.1 程序设计语言的作用域规则 *214*
8.5.2 分程序结构的处理 *216*
8.5.3 无嵌套过程语言的处理 *217*
8.5.4 有嵌套过程语言的处理 *218*
8.6 小结 222
习题 223

第九章 目标代码生成 224

9.1 代码生成器概述 224
9.2 运行时刻内存空间管理的实现 227
9.3 一个简单的代码生成器 233
9.3.1 下次引用信息和活跃信息 *233*
9.3.2 寄存器描述器和地址描述器 *235*
9.3.3 简单代码生成算法 *235*
9.4 小结 240
习题 241

第十章 编译技术应用 242

10.1 DFA 在网上购物平台中的应用 242
10.2 广义 LR 分析方法在自然语言语法分析中的应用 244
10.3 属性文法在模式识别中的应用 247

附录 A SMINI——一个简单模型语言编译器 252

一、S 语言简介 252
二、假想目标机及其指令集 253
三、SMINI 设计与实现 257
四、SMINI 操作与编译示例 259

第一章　编译器概述

编译器（Compiler）是计算机系统的重要组成部分，是一种重要的系统软件，用于将高级程序设计语言编写的程序翻译为计算机硬件可以执行的机器语言程序。一个高级程序设计语言可以通过该语言的“语言参考手册（Language Reference Manual）”（或语言的规格说明书）来定义，但是要执行一个高级程序设计语言编写的程序则必须有编译器的参与。没有编译器，高级程序设计语言只存在于纸面上，不能用来解决任何现实中的问题。本章首先概述程序设计语言的发展史，接着讨论几种高级语言翻译器，并详细介绍编译过程包含的步骤及编译器的一般结构，最后讨论构造编译器的常用方法。

1.1 程序设计语言发展史

程序设计语言（Programming Language）是人向计算机传达意图的载体，是程序员编写程序的工具。程序设计语言是人造的符号语言，具有严格定义的语法和语义规则。程序设计语言分为**低级语言**和**高级语言**。低级语言是面向机器的，又分为**机器语言**（Machine Language）和**汇编语言**（Assembly Language）。高级语言是面向人（程序员）的，更接近数学语言和自然语言。几乎在现代计算机问世的同时，就出现了最早的程序设计语言。随着计算机硬件系统的不断发展和计算机系统性能的持续提升，通过计算机解决问题的范式不断进化，程序设计语言也在不断演进。程序设计语言经历了“面向机器”、“面向过程”、“面向对象”、“面向问题”等几个发展阶段。高级程序设计语言的发展史就是编译器的发展史。

- **20 世纪 40~50 年代——机器语言、汇编语言应运而生**

1946 年 2 月 14 日世界上第一台现代通用计算机 ENIAC 在美国问世。这台计算机的运算速度比当时最快的电动机械计算机快 1000 倍（每秒 5000 次加法或 400 次乘法）。1946 年冯·诺依曼提出采用二进制作为数字计算机的数制基础，计算机通过执行预先编制和存储在计算机内存中的程序来进行工作。1951 年第一台按冯·诺依曼原理制成的通用电子数字计算机 UNIVAC-I 研制成功，这台机器可执行用二进制形式的机器语言编写的程序。由于用二进制码（0、1 数字串）编写程序极易出错，人们引入助记符和符号数来代替二进制操作码和操作数地址，助记符通常采用英文单词（或其缩写），便于记忆，可读性好，这就是汇编语言。汇编语言程序需要通过汇编器翻译为机器代码之后才能执行。虽然汇编语言指令和机器语言指令基本上是一一对应的，也即汇编语言也属于低级语言，但是汇编语言开启了“源代码—翻译器—目标代码”这一语言使用范式，成为编译思想发展的源头。

- **20 世纪 50~60 年代——早期高级语言出现**

1954 年来自 IBM 的 John W. Backus 研制成功世界上第一个脱离机器的高级语言 FORTRAN I。FORTRAN I 的编译器用汇编语言编写，耗费 18 个人年完成。在那之后 FORTRAN 语言又多次升级。FORTRAN 语言有变量、表达式、赋值、调用、输入、输出等

概念；有条件比较、顺序、选择、循环控制；有满足科学计算的整数、实数、复数和数组，以及为保证运算精度的双精度等数据类型。FORTRAN 的出现，使得当时科学计算的生产力提高了一个数量级，奠定了高级语言的地位。在这一时期出现的高级语言还有 ALGOL、COBOL、SIMULA、LISP 等。

- **20 世纪 60~70 年代——结构化程序设计语言发展**

20 世纪 60 年代软件发展史上著名的“软件危机”引发了结构化程序设计语言的研发热潮。结构化程序设计语言采用面向过程的编程方法，其主要特征是有结构化控制结构，包括顺序、分支、循环等控制结构，有全局变量、局部变量、作用域和可见性等概念。有丰富的数据类型，除基本类型之外，用户还可以定义结构类型、枚举类型、指针类型等。结构化程序设计语言的典型代表是 1971 年出现的 Pascal 语言。著名的 C 语言也是结构化程序设计语言。C 语言简洁、灵活，源程序由多个函数构成，可以分别编译，利于开发大型软件。

- **20 世纪 70~80 年代——面向对象语言兴起**

20 世纪 70 年代，软件工程界提出一种新的软件构造方法，即面向对象方法。这种方法追求的目标是使软件开发过程与人们在现实世界中解决问题的过程尽可能接近。它把现实世界中的实体抽象为一个个对象，对象由属性和可以改变属性的方法构成，对象之间通过消息相互通信。为适应这一软件构造方法，结构化程序设计语言纷纷借鉴对象思想推出新版本，以支持面向对象程序设计，如 Objective-C、Object Pascal、C++等。第一个纯粹的面向对象语言是 1980 年正式发布的 Smalltalk。

- **20 世纪 90 年代至今——网络编程语言流行**

20 世纪 90 年代，随着计算机网络尤其是互联网的快速普及，迫切需要能在网络结点之间传递信息的编程语言。由于网络结点平台的多样性，网络编程语言首先应满足平台无关性。SUN 公司在 1995 年底发布的 Java 语言具有平台无关、可移植、面向对象、安全、分布式、高性能、多线程等特点，成为最重要的网络编程语言。广泛流行的网络编程语言还有 PHP、Python、Perl、C#等。另外一类重要的网络编程语言是脚本语言。脚本语言具有类型、变量、关键字、表达式、分支和循环、过程调用等小型语言的特点，能进行面向过程和面向对象的编程，可嵌入在网页中以实现客户端或服务器端的动态效果。典型的脚本语言包括 JavaScript、JScript、VBScript 等。

据统计，全世界出现过的程序设计语言有 2000 种以上，但得到广泛使用的语言远没有这么多。根据著名的 TIOBE 编程语言排行榜，2019 年 11 月世界范围最受程序员青睐的 10 种编程语言及其使用率分别为：（1）Java(16.246%)；（2）C(16.037%)；（3）Python(9.842%)；（4）C++(5.605%)；（5）C#(4.316%)；（6）Visual Basic.NET(4.229%)；（7）JavaScript(1.929%)；（8）PHP(1.720%)；（9）SQL(1.690%)；（10）Swift(1.653%)。这个排行榜每个月发布一次，反映程序设计语言最新的流行度变化。

1.2 语言翻译器

语言翻译器是指能够将一种语言（源语言）编写的程序（源程序）转换成等价的另一种语言（目标语言）编写的程序（目标程序）的程序，其功能见图 1-1。

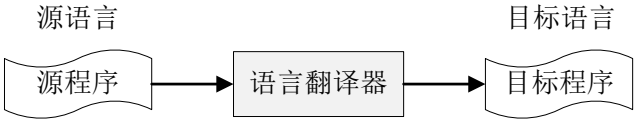

图 1-1 语言翻译器的功能

计算机系统的核心是中央处理器（CPU），每个 CPU 均配置了特有的指令系统。机器语言程序就是由指令系统中的指令的序列构成。本质上，计算机系统只能识别和执行指令系统中的指令。指令的一般格式为：

操作码　　操作数 1　　操作数 2

其中操作码和操作数均是二进制码，如：

00000100　10100001　00101110

其中 00000100 约定为“加”运算，指令的功能是将后面的两个操作数 10100001 和 00101110 相加。如果用汇编语言指令来实现，上述指令可以写为：

ADD　AX，2EH

其中，ADD 是“加”运算助记符，AX 和 2EH 是符号数。显然汇编语言指令相对于机器指令来说，可读性较好，但是计算机不能直接执行汇编指令，需要通过一个语言翻译器将其翻译为机器指令才行。能够将汇编语言程序翻译为机器语言程序的语言翻译器称为**汇编器**，其功能见图 1-2。

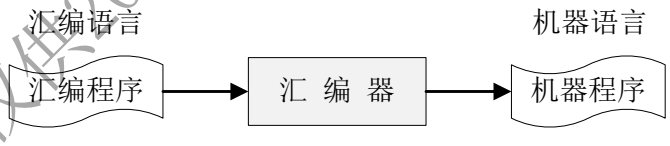

图 1-2 汇编器的功能

如果源语言是高级语言，目标语言是低级语言，那么这个语言翻译器就称为**编译器**，其功能见图 1-3。

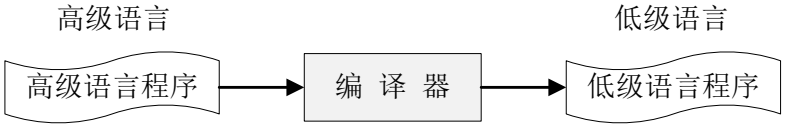

图 1-3 编译器的功能

编译是实现高级语言的一种方式，程序员用高级语言编写好源程序之后，提交给编译器。编译器将源程序翻译为目标代码模块，通过链接、加载，成为内存中可执行的目标程

序。下次程序运行时，不需要再次编译，只需要直接调用目标代码就可以。大部分高级语言都采用编译方式。

高级语言的另一种实现方式是解释。程序运行时，**解释器**接受用户的输入，直接执行源程序中指定的操作，即一边翻译一边执行，然后输出运行结果，解释器的功能见图 1-4。解释器不生成可重复执行的目标代码模块，下次程序运行时，需要再次解释执行。大部分交互式语言、脚本语言、查询命令语言采用解释方式来实现，如：BASIC、Python、JavaScript、SQL等。

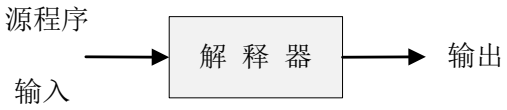

图 1-4 解释器的功能

目前流行的 Java 语言比较特殊，它采用一种编译和解释相结合的混合翻译方式，以支持跨平台性。首先 Java 源程序被编译为一个称为字节码的中间表示形式，然后由一个虚拟机来对字节码做解释执行，其翻译方式如图 1-5 所示。字节码可以在不同平台之间进行传输，只要目标平台上有 Java 虚拟机，就可以运行字节码。

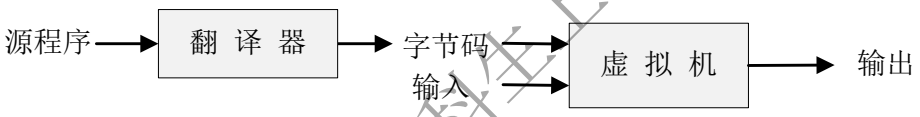

图 1-5 Java 语言的翻译方式

1.3 编译器结构

编译过程与自然语言之间的翻译过程有相似之处。比如要将一个汉语句子“她把一束花放在桌上。”翻译为英语句子。首先要将这个汉语句子中的词语解析出来，并确定其词性、含义等属性，即做词法分析。经过词法分析之后，句子变成一个词语的序列“‘她’、‘把’、‘一束’、‘花’、‘放’、‘在’、‘桌’、‘上’、‘。’”。然后做句法分析，分析这句话的语法结构，并用某种形式化方法，如句法树来描述该句子的语法结构。接着是做语义分析，获取句子中各个成分之间的格关系或依存关系。在经过词法分析、语法分析和语义分析之后，汉语句子被转化成一种机器内部的句法-语义表示。然后是译文生成阶段，首先根据双语词典等知识库得到汉语词语的目标译词，由于汉语词语在英语中往往对应多个单词，所以需要解决义项排歧问题。上述例子对应的正确英文目标译词是“‘she’、‘put’、‘a bunch of’、‘flower’、‘on’、‘table’ ‘.’”。最后，经过形态的调整，如单词词尾的变化、主谓语一致问题解决等处理，把该汉语句子翻译为完整的英语句子“She puts a bunch of flowers on table.”。

编译器的工作从输入高级语言源程序开始，到输出目标代码结束，与自然语言之间的翻译很相似。整个编译过程非常复杂，从宏观上看编译过程分为“分析”和“综合”两个大的阶段，这就是编译的**分析-综合模型**。分析阶段负责对源程序做多层次的分析，判断源程序是否有错误，同时获取源程序的词法、语法、语义信息，并将这些信息存放到创建的符号表

中。如果源程序不存在错误，分析阶段通常会把源程序转换为一个中间表示形式（一般采用中间代码形式）。综合阶段负责将中间表示转换为最终的目标代码。为获得高质量的目标代码，可以针对中间代码和目标代码做优化，在生成目标代码和做代码优化过程中都要用到符号表中的信息。

如果将编译过程划分得更细一点，可以将编译分成 6 个步骤，即词法分析、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，见图 1-6。

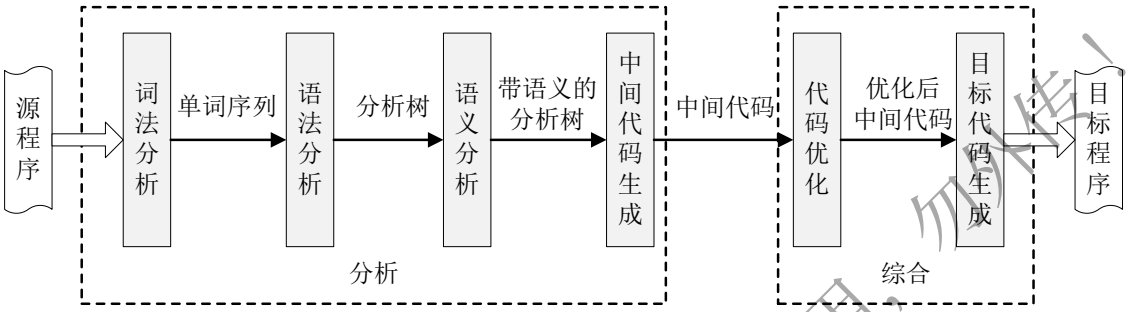

图 1-6 编译过程的步骤划分

下面通过翻译一个 C 语言源程序的片段（一个赋值语句）来展示编译过程每个步骤的工作细节。

【例 1-1】：翻译如下 C 语言的赋值语句：

$$p = i + r * 60 \qquad (1.1)$$

其中，p、i、r 为程序员定义的单精度实型变量，即 float p, I, r ; 。

（1）词法分析

编译的第一个步骤是词法分析。词法分析程序首先读入字符流形式的源程序，然后扫描、切分源程序字符流，从中识别出一个个的单词。空格、回车、程序注释等首先被略去以获得源程序的有效部分。从源程序有效部分识别单词的依据是源语言的构词规则，单词可以是保留字、标识符、运算符、分界符、常数等。

对于（1.1）可识别出如下 7 个单词：

1）p：标识符 id_1；

2）=：赋值运算符；

3）i：标识符 id_2；

4）+：加法运算符；

5）r：标识符 id_3；

6）*：乘法运算符；

7）60：常数。

经过词法分析之后，源程序字符流转化为一个单词的序列。

（2）语法分析

词法分析之后的第二个步骤是语法分析，即在词法分析的基础上，根据源语言的语法规则，判断源程序在语法上是否合法。如果合法则对源程序单词序列进行层次分析，并输出分析树或者语法树反映源程序的语法结构。通常采用巴科斯-瑙尔范式 BNF（ Backus-Naur Form 的缩写）作为语法规则的表示方式。赋值语句的语法规则包含以下几条：

1）<赋值语句> ::= <标识符>“=”<表达式>

2）<表达式> ::= <表达式>“+”<表达式>

3）<表达式> ::= <表达式>“*”<表达式>

4）<表达式> ::= “(”<表达式>“)”

5）<表达式> ::= <标识符>

6）<表达式> ::= <整数>

7）<表达式> ::= <实数>

图 1-7 和 1-8 分别是（1.1）的分析树和语法树。图 1-7 的分析树反映了赋值语句的层次结构和运算顺序，即变量 r 和常数 60 首先做乘运算，变量 i 和 r * 60 的运算结果再做加运算，运算结果最后再赋给变量 p。图 1-8 的语法树同样反映了赋值语句的层次结构和运算顺序，但是结点数更少。可以说语法树是分析树的一种浓缩形式。

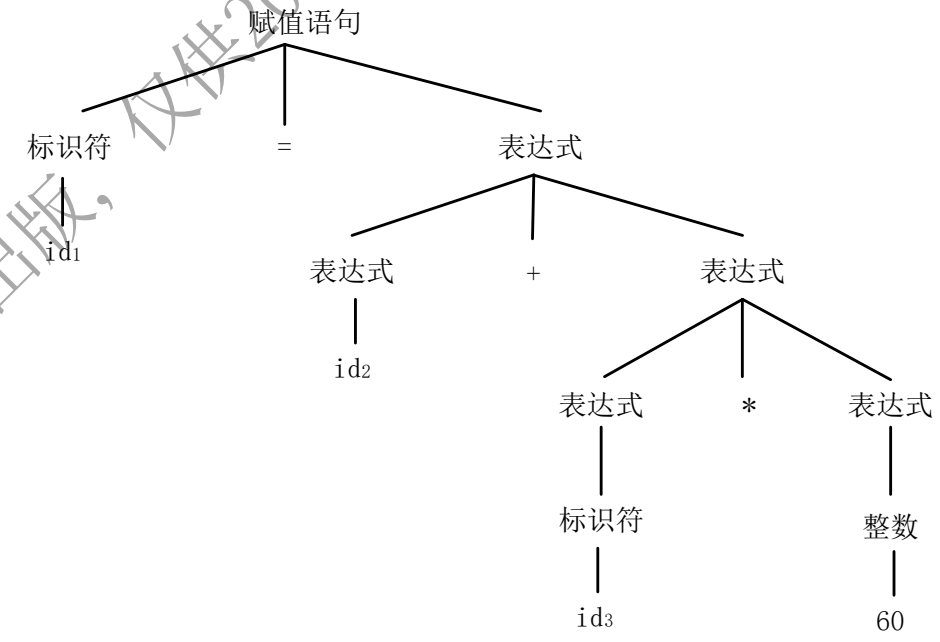

图 1-7 （1.1）的分析树

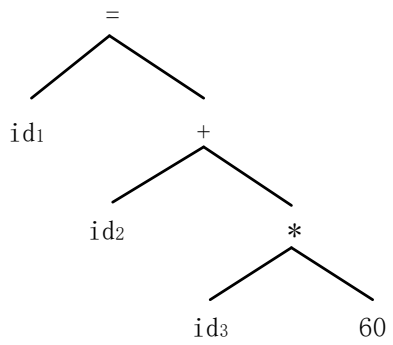

图 1-8 （1.1）的语法树

（3）语义分析

语义分析阶段使用语法树和符号表中的信息来检查源程序在语义上是否合法，即源程序的各个组成部分组合在一起是否有意义。这个阶段同时要分析源程序的说明部分来收集源程序中定义的标识符（尤其是变量）的类型等信息并把这些信息存放到符号表中，以便在综合阶段使用这些信息。语义分析阶段的一个重要任务是类型检查，即检查语句中的数据类型是否合法，例如表达式中的运算对象在类型上是否一致或者相容、数组的下标是否是整数等，如一个二目算术运算符可以要求其两个运算对象的类型必须一致（同时为整型或者同时为实型）。某些高级语言允许类型转换，如果一个二目运算符应用于一个整型数和一个实型数，那么编译器首先需要将其中的整型数转换为实型数再来做这个二目运算。例子（1.1）中，r 是单精度实型变量，60 是整数，两者类型不一致。在语义分析阶段检测到这个类型不一致之后，可以在分析树中增加一个显性的结点 int-to-float，用于将 60 转换为实型数，如图 1-9 所示。

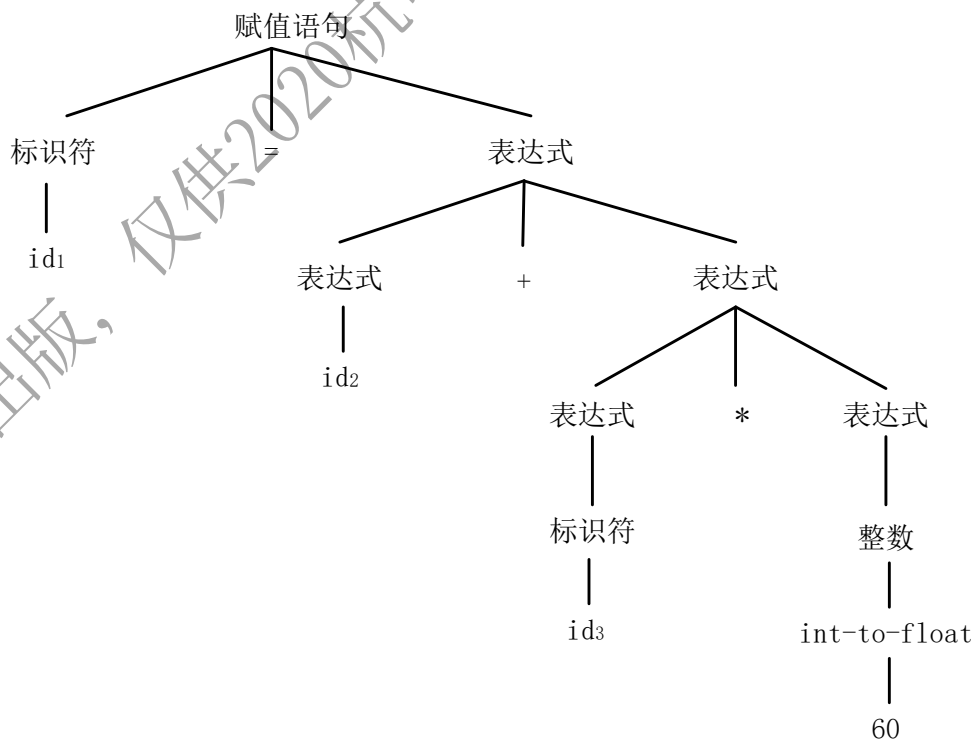

图 1-9 带语义结点的分析树

（4）中间代码生成

前 3 个分析步骤完成之后，如果没有发现错误，说明源程序在词法、语法、语义层面都是正确的。这时编译器通常会将源程序转化为一个低级或者类机器语言的中间表示。这个中间表示一般来说是平台无关的，并且易于生成和易于翻译成目标代码。中间表示形式可以是树形或者图形的，但更多的是采用中间代码形式。三地址代码就是一种常见的中间代码，它的每条指令一般具有三个运算分量，其中两个是运算对象，一个是运算结果。例（1.1）翻译得到的中间代码序列如下：

$$t_1 = \text{int-to-float}\ (60)$$

$$t_2 = id_3 * t_1 \qquad (1.2)$$

$$t_3 = id_2 + t_2$$

$$id_1 = t_3$$

（5）代码优化

优化代码的目的是获得高质量的目标代码。“高质量”体现在最终生成的目标代码运行更快、所需的运行内存空间更小。代码优化分为针对中间代码的优化和针对目标代码的优化，其中针对中间代码的优化由于其平台无关性，更具有普遍意义。考察（1.2）第一条代码中的运算 int-to-float (60)，其功能是将 60 转换为实数，可以在编译时刻一劳永逸地完成，而不必在运行时刻每次都去做。t_3 的作用是作为中介将 $id_2 + t_2$ 的运算结果赋给 id_1，由于 t_3 只在这里使用了一次，可以将 $id_2 + t_2$ 的运算结果直接赋给 id_1，而不必通过 t_3。对（1.2）优化之后的代码如下：

$$t_1 = id_3 * 60.0 \qquad (1.3)$$

$$id_1 = id_2 + t_1$$

（6）目标代码生成

目标代码生成的责任是将优化之后的中间代码映射为目标代码。目标代码的形式包括绝对机器指令、可重定位的机器指令、汇编语言代码等。目标代码生成需要为源程序中定义的变量分配寄存器，为每条中间代码选择合适的机器指令，包括确定指令的操作码及操作数的编址方式。优化后的中间代码序列（1.3）可映射为如下汇编语言形式的目标代码：

```
MOVF   id3 , R2
MULF   #60.0 , R2                    (1.4)
MOVF   id2 , R1
ADDF   R2 , R1
MOVF   R1 , id1
```

以上各指令中的 F 表示处理的是单精度实型数。（1.4）中的指令首先将实型变量 id_3 装载到寄存器 R_2 中，然后 R_2 中的数与实数 60.0 做乘法运算，结果仍然存放在 R_2 中。接着将

实型数 id_2 装载到寄存器 R_1 中，R_2 中的数与 R_1 中的数做加法运算，结果存放到 R_1 中。最后将 R_1 中的值复制到 id_1 对应的内存单元中。

编译过程中，特别是在综合阶段，需要使用源程序中定义的名字及其属性值。名字有变量名、常量名、过程（函数）名等。变量涉及的属性包括类型、存储分配信息、嵌套深度、作用域等；常量涉及的属性包括类型、具体的值等；过程涉及的属性有参数数量和类型、参数传递方法、返回值的类型等。名字及其属性信息存放在符号表中，相应有变量表、常量表、过程（函数）表等，每个名字在符号表中均有一个记录条目（称为表项）。一般来说，在分析阶段创建各类符号表，并获取名字的属性信息，在综合阶段使用符号表中的信息。对符号表的操作贯穿整个编译过程，编译器中包含专门的符号表管理模块完成对应的工作。

编译器另一重要组成部分是错误处理模块。错误处理机制是编译器可用性和用户友好性的重要体现。对错误处理程序的基本要求是能发现源程序中的错误并指明错误所处的位置。源程序的错误又分为词法错误（如单词拼写有误）、语法错误（如表达式中括号不配对）、语义错误（如运算对象的类型不兼容）等。一个用户友好的编译器应能判断源程序的错误类型并给出尽可能详细的错误提示，以便程序员能快速改正源程序中的错误。有些编译器还有自动纠错功能，对于一些显而易见的错误可以直接予以纠正。

编译器通常分为相对独立的**前端**系统和**后端**系统。前端完成分析的功能，即对源程序进行词法分析、语法分析、语义分析，并将合法的源程序转换成中间代码；后端完成综合的功能，即对中间代码进行优化并生成目标代码。前端面向源语言并在很大程度上独立于目标机器，后端面向中间代码，独立于源语言但依赖目标机器。不同的目标机器采用不同的指令系统，最后的目标代码依赖于具体的指令系统。开发时将编译器分成耦合度低的前端和后端，有利于提高开发编译器的效率和可移植性。如“同一前端”+“不同后端”可以得到同一高级语言在不同机器上的多个编译器；而“不同前端”+“同一后端”可以得到几个不同语言在同一机器上的多个编译器。图 1-10 是“多个前端”+“多个后端”组合得到多个语言在多个平台上的多个编译器的示例。需要注意的是，这“多个前端”和“多个后端”必须采用同一种中间表示形式。

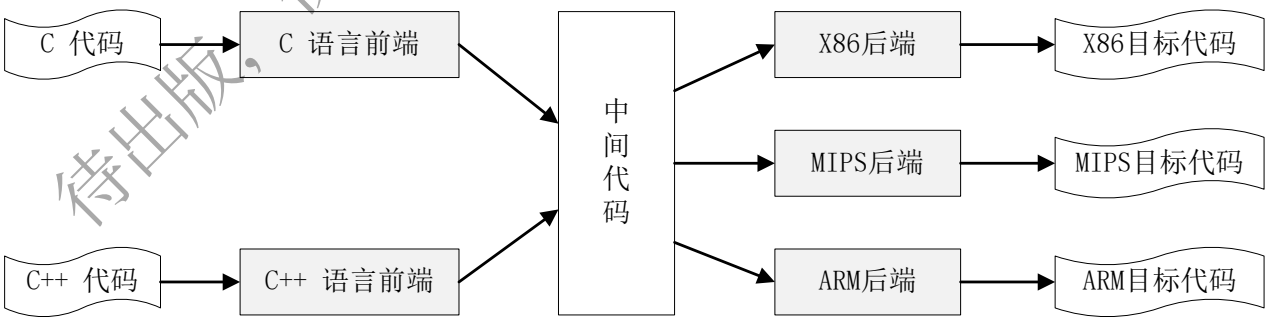

图 1-10 “多个前端”+“多个后端”的多编译器示例

1.4 编译器构造方法

在实现编译器的时候，可以将多个连续的编译步骤组合为一遍。所谓**遍**，是指对源程序或源程序中间表示的一次从头到尾的扫描，同时完成多个编译步骤的功能。每一遍需要从外

部存储器中读入一个文件，完成一定的编译工作之后，又要以文件形式将处理结果写到外部存储器中。后一遍的输入是前一遍的输出，第一遍的输入是源程序，最后一遍的输出是目标代码。编译器可以通过一遍或者多遍来组织，具体采用几遍需要考察源语言的特点、设计编译器的目的、运行编译器的机器的性能等因素。需要指出的是，由于每一遍需要访问两次外部存储器，如果遍数过多，将影响编译的时间效率，而遍数少的话对运行编译器的机器的内存容量有较高要求。

编译器是一个系统软件，构造编译器是一个复杂的系统工程，从零开始手工编写一个编译器工作量巨大。为提高效率，在构造编译器的时候可以利用现代的软件开发环境，如语言编辑器、调试器、版本管理、软件测试工具等。除了这些通用的软件开发工具，由于编译器的特殊性，开发人员还可采用一些特有的开发方法。

设计和实现一个编译器，不仅要考虑源语言和目标语言，还要考虑开发这个编译器采用的语言。世界上第一个通用的高级程序设计语言是 1954 年出现的 FORTRAN 语言，其编译器用汇编语言编写。从那开始到 20 世纪 60 年代，几乎所有的编译器都是用机器语言或者汇编语言编写的。用低级语言开发编译器，存在开发周期长、不易调试、源码可读性差、可扩充性和可维护性差、可靠性低等问题。20 世纪 70 年代，开始采用高级语言来编写编译器，同时多种编译器自动开发工具被推出，如 BELL 实验室推出的 LEX 和 YACC，大大提高了开发编译器的效率，同时也提高了编译器的可靠性和可移植性。

构造编译器主要有如下几条途径：

（1）自展技术

自展技术是编译器的一种渐进式实现方案，其基本思想是先用目标机器上的汇编语言或者机器语言来编写源语言 L 的一个子集 L_1 的编译器，然后用 L_1 作为编写语言实现 L 的一个较大子集 L_2 的编译器，再用 L_2 作为编写语言，实现一个更大的子集 L_3 的编译器，不断重复这一过程直到最后实现源语言 L 的编译器，如图 1-11 所示。在这个过程中，只有实现 L_1 编译器时是采用低级语言，其它部分采用的都是高级语言。

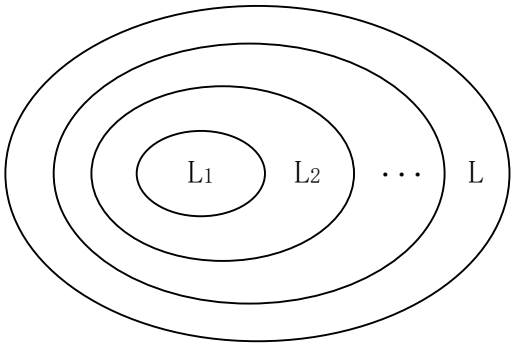

图 1-11 编译器实现的自展技术

（2）用已实现的高级语言开发其它高级语言的编译器

例如：用 C 语言编写 C++语言的编译程序，然后用 C 语言编译器进行编译，就得到 C++语言的编译器。这是目前最常见的编译器构造方法。

（3）利用构造工具实现编译器

编译器构造工具主要包括：

1）词法分析器生成器：可以根据描述源语言单词结构的正规表达式生成词法分析器源代码，如 LEX。

2）语法分析器生成器：可以根据源语言的语法规则自动生成语法分析器源代码，如 YACC。

3）语法制导翻译器生成器：可以生成用于遍历分析树并生成中间代码的程序。

4）代码生成器生成器：能根据翻译规则（从中间代码映射到目标代码）输出一个目标代码生成器的源代码。

通常情况下，编译器的运行平台就是其编译源程序后得到的目标代码运行的平台。但有时候需要运行目标代码的平台无法运行编译器这样一个大型系统软件，解决的方法就是**交叉编译**。所谓交叉编译就是在一个物理平台上（称为主机平台）生成另一个物理平台上（称为目标平台）的目标代码。源程序的编写和编译均在主机平台上完成，目标代码的运行则是在目标平台上。

早期的编译器均是专门针对某一种高级语言和某一个目标机器而编写的。随着以嵌入式系统为代表的高性能体系结构的飞速发展，相继出现了支持多种源语言和多个目标机器的编译系统，如可重定向编译程序 GCC（GNU compiler collection）。

1.5 小结

编译器是一个语言翻译器，可用于将高级语言书写的程序翻译为等价的低级语言书写的程序。世界上第一个高级语言是 FORTRAN 语言，其编译器用汇编语言编写。高级语言分为面向过程的结构化编程语言、面向对象的编程语言、网络编程语言等。高级语言和编译技术是同步演进的，没有编译器对高级语言的实现，高级语言无法解决实际问题。语言翻译器包括汇编器、编译器、解释器等。汇编器可以将汇编语言程序翻译为机器语言程序，汇编语言与机器语言都属于低级语言。与编译器类似，解释器也可以将高级语言程序翻译为低级语言程序，在翻译的时候同样要对高级语言程序进行词法、语法和语义分析。与编译器不同之处在于，解释器是一边翻译一边执行，并且不保存目标代码，在下次执行时解释器需要再次翻译源程序。源程序的编译过程非常复杂，从宏观上看编译过程分为综合与分析两个阶段，它们之间的界面是源程序的中间表示。对应地，编译器通常分为相对独立的的前端和后端，前端的功能是将源程序转换为某种形式的中间表示，后端则负责将该中间表示转换为目标代码。从逻辑上可以将整个编译过程分为词法分析、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等 6 个步骤。符号表管理和出错处理也是编译过程的重要组成部分。

构造编译器时往往将多个编译步骤组合在一起作为一“遍”来实现。编译器翻译源程序时是采用“几遍”编译的是考察一个编译器的重要指标。最早的编译器用低级语言开发，开发效率低下，自展技术的引入可以部分解决这个问题。随着高级语言的发展，用一种高级语言开发另一种高级语言的编译器已成为编译器开发的主流方法。为进一步提高编译器的开发效率，多种编译器辅助开发工具被研发出来并得到广泛应用。本章要求掌握几个与编译器相关的概念，重点是理解编译过程的 6 个步骤及每个步骤需完成的主要功能。后续章节的内容基本上是围绕编译过程的 6 个步骤展开的。

习题

1.1 简要概述计算机程序设计语言的发展历程。

1.2 试描述高级程序设计语言需要翻译程序的原因。

1.3 用自己的语言说明翻译程序中编译和解释的差别。

1.4 编译程序具有哪些特点？

1.5 编译程序的逻辑过程包含哪几个组成部分？ 请分别阐述各个组成部分的主要任务（输入、输出、操作）。

1.6 试简要描述编译程序的结构和组织方式。

1.7 通过文献查阅，调研各种编译器生成工具，并挑选其中一个自己感兴趣的工具进行介绍。