HZ BOOKS
华章教育

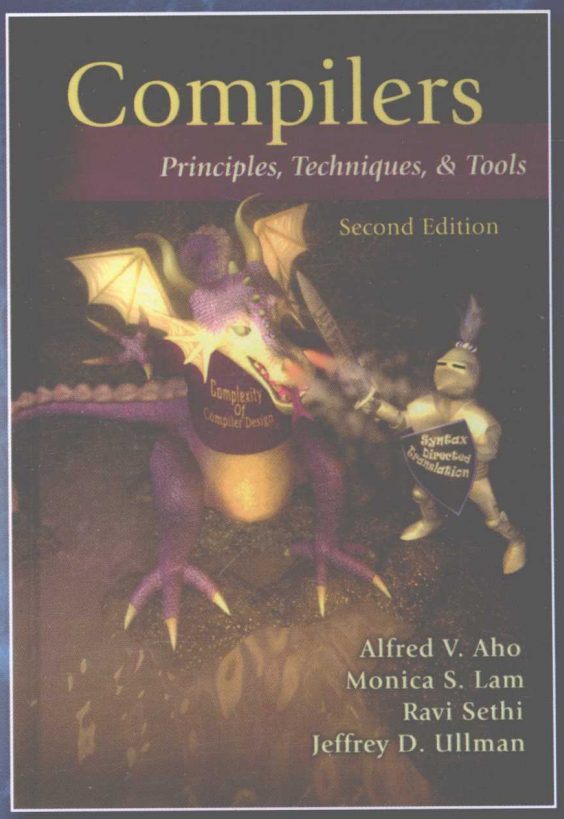
计 算 机 科 学 丛 书

第2版

本科教学版

编译原理

(美) Alfred V. Aho 哥伦比亚大学 Monica S. Lam 斯坦福大学 Ravi Sethi Avaya实验室 Jeffrey D. Ullman 斯坦福大学 著
赵建华 郑滔 戴新宇 南京大学 译



Compilers
Principles, Techniques and Tools
Second Edition

机械工业出版社
China Machine Press

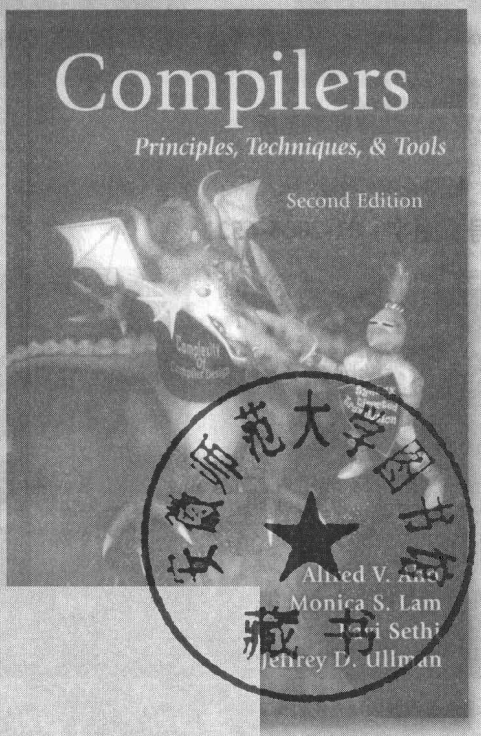
计 算 机 科 学 丛 书

第2版

本科教学版

编译原理

(美) Alfred V. Aho 哥伦比亚大学 Monica S. Lam 斯坦福大学 Ravi Sethi Avaya实验室 Jeffrey D. Ullman 斯坦福大学 著
赵建华 郑滔 戴新宇 南京大学 译



Compilers

Principles, Techniques and Tools

Second Edition

机械工业出版社
China Machine Press

《编译原理》是编译原理课程方面的经典教材，全面、深入地探讨了编译器设计方面的重要主题，包括词法分析、语法分析、语法制导定义和语法制导翻译、运行时刻环境、目标代码生成、代码优化技术、并行性检测以及过程间分析技术，并在相关章节中给出大量的实例。与上一版相比，本书进行了全面修订，涵盖了编译器开发方面最新进展。每章中都提供了大量的实例及参考文献。

本书基于该书第2版进行改编，内容更加精练和实用，体系更加符合国内教学情况，适合作为高等院校计算机及相关专业本科生的编译原理课程的教材，也是广大研究人员和技术人员的极佳参考读物。

本书版权登记号：图字：01-2006-6521

图书在版编目（CIP）数据

编译原理　第2版：本科教学版/（美）阿霍（Aho, A. V.）等著；赵建华等译. —北京：机械工业出版社，2009.5

（计算机科学丛书）

书名原文：Compilers: Principles, Techniques and Tools, Second Edition

ISBN 978-7-111-26929-8

Ⅰ. 编…　Ⅱ. ①阿…　②赵…　Ⅲ. 编译程序－程序设计－高等学校－教材　Ⅳ. TP314

中国版本图书馆 CIP 数据核字（2009）第062532号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：金　纯

北京京北印刷有限公司印刷

2009年5月第1版第1次印刷

184mm×260mm · 26.5印张

标准书号：ISBN 978-7-111-26929-8

定价：55.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章分社较早意识到“出版要为教育服务”。自1998年开始，华章分社就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章分社欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

联系电话：（010）88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037

华章教育

改编者序

构造编译器的原理和技术是计算机科学技术领域中一个非常重要的组成部分，指导人们构造能够生成正确、高效的代码的编译器。现在的绝大部分软件都是使用高级程序设计语言编写的，需要使用编译器来得到可运行代码，因此编译原理和技术对于构造正确、可靠、高效的软件是非常重要的。经过了50年的研究发展，编译技术已经使得人们可以为各种高级编程机制生成高效的代码，使得人们可以使用更加抽象的语言来编写高效的软件。但硬件技术的进步仍然对编译技术提出了新的挑战。比如多核CPU的广泛应用要求更优秀的程序分析技术和并行编译器。因此，编译原理和技术在将来仍然是一个重要的研究课题。

Aho.等人编写的《编译原理》是一本经典的教材。这本书不仅包含了编译器构造的基本原理和技术，还包含了很多和编译相关的高级技术。对于专业技术人员来说，这是一本很全面的参考书目。但是书中的很多内容超出了本科教学的要求，不符合中国的本科教材的习惯。因此，出版社委托我们对这本书进行改编，主要的工作是删减一些不需要在本科教学过程中讲授的内容。保留下的内容包括词法分析、语法分析、语义分析、中间代码生成，以及运行时刻环境、优化和代码生成方法的基本技术。

我们删去了原书的第十章、第十一章和第十二章。这三章的内容是关于并行性和程序分析的高级议题，一般不对本科生讲授。此外，我们对原书第九章机器无关优化的内容进行了删减，保留了一些基本的数据流优化算法。我们还删减了一些高级的算法和技术，包括运行时刻环境中的短停顿垃圾收集算法、类型检查中的类型推导和合一算法，高效构造DFA算法等。另外，我们还删去了一些与实现细节有关的技术，比如词法分析中缓冲区的管理、语法分析中LR分析表的压缩技术等。删去了这些高级内容之后，保留部分已经可以在一个学期的本科生课程中讲完。当然，考虑到不同学校有不同的专业要求，任课教师仍然可以考虑舍弃一些内容，比如第八章中关于代码生成的高级议题。

编译原理是一门比较难学的课程。主要原因在于它包含了很多理论性的东西，抽象程度比较高，而且还包含了很多复杂的算法和用于编译器构造的抽象数学概念。我建议学生学习的时候可以先阅读本书的第二章。第二章的内容可以帮助大家了解编译器的基本构造和功能，然后在学习后续各章节的时候加深理解。自己动手编写一个小型语言的编译器也是一个很好的学习方法。使用Yacc和Lex等工具之后，编写一个这样的编译器并不需要很大的工作量，却可以有效帮助大家深入理解各种编译技术。

对编译的基本原理和技术有所了解之后，如果读者还希望进一步深入学习，我建议大家购买完整版的《编译原理》来阅读。

译者

2009年4月

前言

从本书的1986版出版到现在，编译器设计领域已经发生了很大的改变。随着程序设计语言的发展，提出了新的编译问题。计算机体系结构提供了多种多样的资源，而编译器设计者必须能够充分利用这些资源。最有意思的事情可能是，古老的代码优化技术已经在编译器之外找到了新的应用。现在，有些工具利用这些技术来寻找软件中的缺陷，以及(最重要的是)寻找现有代码中的安全漏洞。而且，很多"前端"技术——文法、正则表达式、语法分析器以及语法制导翻译器等——仍然被广泛应用。

因此，本书先前的版本所体现的我们的价值观一直没有改变。我们知道，只有很少的读者将会去构建甚至维护一个主流程序设计语言的编译器。但是，和编译器相关的模型、理论和算法可以被应用到软件设计和开发中出现的各种各样的问题上。因此，我们会关注那些在设计一个语言处理器时常常会碰到的问题，而不考虑具体的源语言和目标机器究竟是什么。

使用本书

下面是各章的概要介绍：

- 第1章给出一些关于学习动机的资料，同时也将给出一些关于计算机体系结构和程序设计语言原则的背景知识。
- 第2章会开发一个小型的编译器，并介绍很多重要概念。这些概念将在后面的各章中深入介绍。这个编译器本身将在附录中给出。
- 第3章将讨论词法分析、正则表达式、有穷状态自动机和词法分析器的生成器工具。这些内容是各种文本处理的基础。
- 第4章将讨论主流的语法分析方法，包括自顶向下方法(递归下降法、LL技术)和自底向上方法(LR技术和它的变体)。
- 第5章将介绍语法制导定义和语法制导翻译的基本思想。
- 第6章将使用第5章中的理论，并说明如何使用这些理论为一个典型的程序设计语言生成中间代码。
- 第7章将讨论运行时刻环境，特别是运行时刻栈的管理和垃圾回收机制。
- 第8章将主要讨论目标代码生成技术。该章会讨论基本块的构造，从表达式和基本块生成代码的方法，以及寄存器分配技术。
- 第9章将介绍代码优化技术，包括流图、数据流分析框架以及求解这些框架的迭代算法。

哥伦比亚大学、哈佛大学、斯坦福大学已经开设了讲授本书内容的课程。哥伦比亚大学定期开设一门关于程序设计语言和翻译器的课程，使用了本书前8章的内容。该课程常年面向高年级本科生/一年级研究生讲授，这门课程的亮点是一个长达一个学期的课程实践项目。在该项目中，学生分成小组，创建并实现一个他们自己设计的小型语言。学生创建的语言涉及多个应用领域，包括量子计算、音乐合成、计算机图形学、游戏、矩阵运算和很多其他领域。在构建他们自己的编译器时，学生们使用了很多种可以生成编译器组件的工具，比如ANTLR、Lex和Yacc；他

们还使用了第2章和第5章中讨论的语法制导翻译技术。

斯坦福大学开设了一门历时一个学季的入门课程，大致涵盖了本书第1章到第8章的内容，同时还会简介本书第9章中全局代码优化的相关内容。

预备知识

学习本书的读者应该拥有一些“计算机科学的综合知识”，至少学过两门程序设计课程，以及数据结构和离散数学的课程。具备多种程序设计语言的知识对学习本书会有所帮助。

练习

本书包含内容广泛的练习，几乎每一节都有一些练习。我们用感叹号来表示较难的练习或练习中的一部分。难度最大的练习有两个感叹号。

万维网上的支持

在本书的主页(http://dragonbook.stanford.edu)[⊖]上可以找到本书已知错误的勘误表以及一些支持性资料。我们希望将我们讲授的每一门与编译器相关的课程的可用讲义(包括家庭作业、答案和练习等)都提供出来。我们也计划公布由一些重要编译器的作者撰写的关于这些编译器的描述。

致谢

本书封面由 Strange Tonic Productions 的 S. D. Ullman 设计。

Jon Bentley 针对本书的初稿中的多章内容与我们进行了广泛深入的讨论。我们收到了来自下列人员的有帮助的评价和勘误：Domenico Bianculli、Peter Bosch、Marcio Buss、Marc Eaddy、Stephen Edwards、Vibhav Garg、Kim Hazelwood、Gaurav Kc、Wei Li、Mike Smith、Art Stamness、Krysta Svore、Olivier Tardieu 和 Jia Zeng。我们衷心感谢这些人的帮助。当然，书中还可能有错漏之处，希望得到指正和反馈。

另外，Monica 希望能够向她在 SUIF 编译器团队的同事表示感谢，感谢他们在18年的时间里给予她的支持和帮助，他们是：Gerald Aigner、Dzintars Avots、Saman Amarasinghe、Jennifer Anderson、Michael Carbin、Gerald Cheong、Amer Diwan、Robert French、Anwar Ghuloum、Mary Hall、John Hennessy、David Heine、Shih-Wei Liao、Amy Lim、Benjamin Livshits、Michael Martin、Dror Maydan、Todd Mowry、Brian Murphy、Jeffrey Oplinger、Karen Pieper、Martin Rinard、Olatunji Ruwase、Constantine Sapuntzakis、Patrick Sathyanathan、Michael Smith、Steven Tjiang、Chau-Wen Tseng、Christopher Unkel、John Whaley、Robert Wilson、Christopher Wilson 和 Michael Wolf。

A. V. A., Chatham NJ
M. S. L., Menlo Park CA
R. S., Far Hills NJ
J. D. U., Stanford CA
2006年6月

⊖ 读者可以从 http://infolab.stanford.edu/~ullman/dragon/errata.html 找到本书的勘误表，并可以从 http://infolab.stanford.edu/~ullman/dragon.html 处找到本书的一些支持资料。

目 录

第1章 引 论

程序设计语言是向人以及计算机描述计算过程的记号。如我们所知，这个世界依赖于程序设计语言，因为在所有计算机上运行的所有软件都是用某种程序设计语言编写的。但是，在一个程序可以运行之前，它首先需要被翻译成一种能够被计算机执行的形式。

完成这项翻译工作的软件系统称为*编译器*(compiler)。

本书介绍的是设计和实现编译器的方法。我们将介绍用于构建面向多种语言和机器的翻译器的一些基本思想。编译器设计的原理和技术还可以用于编译器设计之外的众多领域。因此，这些原理和技术通常会在一个计算机科学家的职业生涯中多次被用到。研究编译器的编写将涉及程序设计语言、计算机体系结构、形式语言理论、算法和软件工程。

在本章中，我们将介绍语言翻译器的不同形式，在高层次上概述一个典型编译器的结构，并讨论了程序设计语言和硬件体系结构的发展趋势。这些趋势将影响编译器的形式。我们还将介绍关于编译器设计和计算机科学理论的关系的一些事实，并给出编译技术在编译领域之外的一些应用。最后，我们将简单论述在我们研究编译器时需要用到的重要的程序设计语言概念。

1.1 语言处理器

简单地说，一个编译器就是一个程序，它可以阅读以某一种语言(源语言)编写的程序，并把该程序翻译成为一个等价的、用另一种语言(目标语言)编写的程序，参见图1-1。编译器的重要任务之一是报告它在翻译过程中发现的源程序中的错误。

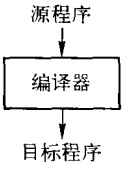


图1-1 一个编译器

如果目标程序是一个可执行的机器语言程序，那么它就可以被用户调用，处理输入并产生输出。参见图1-2。

解释器(interpreter)是另一种常见的语言处理器。它并不通过翻译的方式生成目标程序。从用户的角度看，解释器直接利用用户提供的输入执行源程序中指定的操作。参见图1-3。

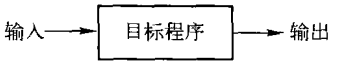


图1-2 运行目标程序

在把用户输入映射成为输出的过程中，由一个编译器产生的机器语言目标程序通常比一个解释器快很多。然而，解释器的错误诊断效果通常比编译器更好，因为它逐个语句地执行源程序。

例1.1 Java语言处理器结合了编译和解释过程，如图1-4所示。一个Java源程序首先被编译成一个称为*字节码*(bytecode)的中间表示形式。然后由一个虚拟机对得到的字节码加以解释执行。这样安排的好处之一是在一台机器上编译得到的字节码可以在另一台机器上解释执行。通过网络就可以完成机器之间的迁移。

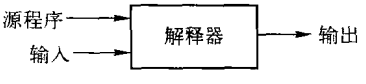


图1-3 一个解释器

为了更快地完成输入到输出的处理，有些被称为*即时*(just in time)编译器的Java编译器在运行中间程序处理输入的前一刻首先把字节码翻译成为机器语言，然后再执行程序。 □

如图1-5所示，除了编译器之外，创建一个可执行的目标程序还需要一些其他程序。一个源

程序可能被分割成为多个模块，并存放于独立的文件中。把源程序聚合在一起的任务有时会由一个被称为预处理器(preprocessor)的程序独立完成。预处理器还负责把那些称为宏的缩写形式转换为源语言的语句。

然后，将经过预处理的源程序作为输入传递给一个编译器。编译器可能产生一个汇编语言程序作为其输出，因为汇编语言比较容易输出和调试。接着，这个汇编语言程序由称为汇编器(assembler)的程序进行处理，并生成可重定位的机器代码。

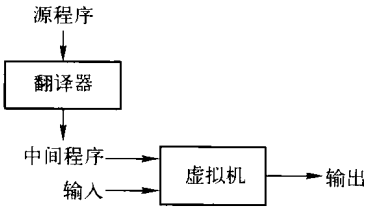


图 1-4 一个混合编译器

大型程序经常被分成多个部分进行编译，因此，可重定位的机器代码有必要和其他可重定位的目标文件以及库文件连接到一起，形成真正在机器上运行的代码。一个文件中的代码可能指向另一个文件中的位置，而链接器(linker)能够解决外部内存地址的问题。最后，加载器(loader)把所有的可执行目标文件放到内存中执行。

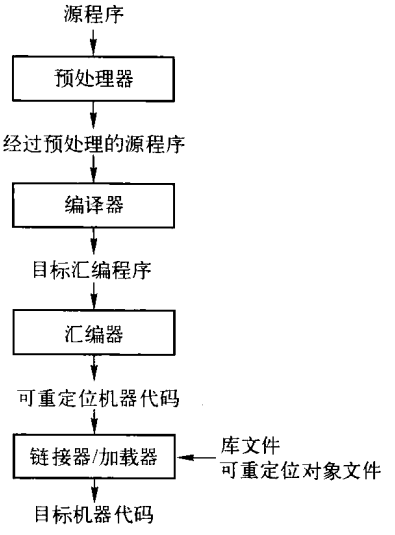


图 1-5 一个语言处理系统

1.1 节的练习

练习 1.1.1：编译器和解释器之间的区别是什么？

练习 1.1.2：编译器相对于解释器的优点是什么？解释器相对于编译器的优点是什么？

练习 1.1.3：在一个语言处理系统中，编译器产生汇编语言而不是机器语言的好处是什么？

练习 1.1.4：把一种高级语言翻译成为另一种高级语言的编译器称为源到源(source-to-source)的翻译器。编译器使用 C 语言作为目标语言有什么好处？

练习 1.1.5：描述一下汇编器所要完成的一些任务。

1.2 一个编译器的结构

到现在为止，我们把编译器看作一个黑盒子，它能够把源程序映射为在语义上等价的目标程序。如果把这个盒子稍微打开一点，我们就会看到这个映射过程由两个部分组成：分析部分和综合部分。

分析(analysis)部分把源程序分解成为多个组成要素，并在这些要素之上加上语法结构。然后，它使用这个结构来创建该源程序的一个中间表示。如果分析部分检查出源程序没有按照正确的语法构成，或者语义上不一致，它就必须提供有用的信息，使得用户可以按此进行改正。分析部分还会收集有关源程序的信息，并把信息存放在一个称为符号表(symbol table)的数据结构中。符号表将和中间表示形式一起传送给综合部分。

综合(synthesis)部分根据中间表示和符号表中的信息来构造用户期待的目标程序。分析部分经常被称为编译器的前端(front end)，而综合部分称为后端(back end)。

如果我们更加详细地研究编译过程，会发现它顺序执行了一组步骤(phase)。每个步骤把源程序的一种表示方式转换成另一种表示方式。一个典型的把编译程序分解成为多个步骤的方式如图 1-6 所示。在实践中，多个步骤可能被组合在一起，而这些组合在一起的步骤之间的中间表示不需要被明确地构造出来。存放整个源程序的信息的符号表可由编译器的各个步骤使用。

有些编译器在前端和后端之间有一个与机器无关的优化步骤。这个优化步骤的目的是在中

间表示之上进行转换，以便后端程序能够生成更好的目标程序。如果基于未经过此优化步骤的中间表示来生成代码，则代码的质量会受到影响。因为优化是可选的，所以图 1-6 中所示的两个优化步骤之一可以被省略。

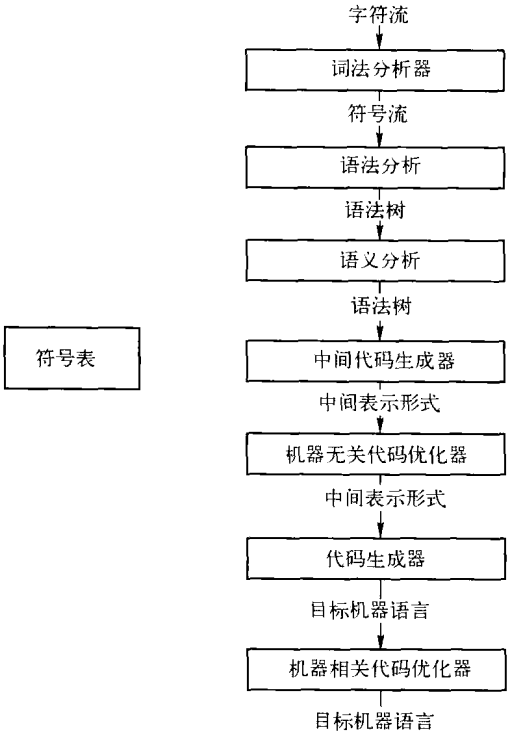


图 1-6　一个编译器的各个步骤

1.2.1　词法分析

编译器的第一个步骤称为词法分析(lexical analysis)或扫描(scanning)。词法分析器读入组成源程序的字符流，并且将它们组织成为有意义的词素(lexeme)的序列。对于每个词素，词法分析器产生如下形式的词法单元(token)作为输出：

〈token-name，attribute-value〉

这个词法单元被传送给下一个步骤，即语法分析。在这个词法单元中，第一个分量 token-name 是一个由语法分析步骤使用的抽象符号，而第二个分量 attribute-value 指向符号表中关于这个词法单元的条目。符号表条目的信息会被语义分析和代码生成步骤使用。

比如，假设一个源程序包含如下的赋值语句

```
position = initial + rate * 60        (1.1)
```

这个赋值语句中的字符可以组合成如下词素，并映射成为如下词法单元。这些词法单元将被传递给语法分析阶段。

1）position 是一个词素，被映射成词法单元〈**id**，1〉，其中 **id** 是表示标识符(identifier)的抽象符号，而 1 指向符号表中 position 对应的条目。一个标识符对应的符号表条目存放该标识符有关的信息，比如它的名字和类型。

2）赋值符号 = 是一个词素，被映射成词法单元〈 = 〉。因为这个词法单元不需要属性值，所以我们省略了第二个分量。也可以使用 **assign** 这样的抽象符号作为词法单元的名字，但是为了标记上的方便，我们选择使用词素本身作为抽象符号的名字。

3）initial 是一个词素，被映射成词法单元〈**id**，2〉，其中 2 指向 initial 对应的符号表条目。

4）+ 是一个词素，被映射成词法单元〈 + 〉。

5）rate 是一个词素，被映射成词法单元〈**id**，3〉，其中 3 指向 rate 对应的符号表条目。

6）* 是一个词素，被映射成词法单元〈 * 〉。

7）60 是一个词素，被映射成词法单元〈60〉[⊖]。

分隔词素的空格会被词法分析器忽略掉。

图 1-7 给出经过词法分析之后，赋值语句 1.1 被表示成如下的词法单元序列：

〈**id**，1〉〈 = 〉〈**id**，2〉〈 + 〉〈**id**，3〉〈 * 〉〈60〉　　(1.2)

在这个表示中，词法单元名 = 、+ 和 * 分别是表示赋值、加法运算符、乘法运算符的抽象符号。

⊖ 从技术上讲，我们应该为语法单元 60 建立一个形如〈**number**，4〉的词法单元，其中 4 指向符号表中对应于整数 60 的条目。但是我们要到第 2 章中才讨论数字的词法单元。第 3 章将讨论建立词法分析器的技术。

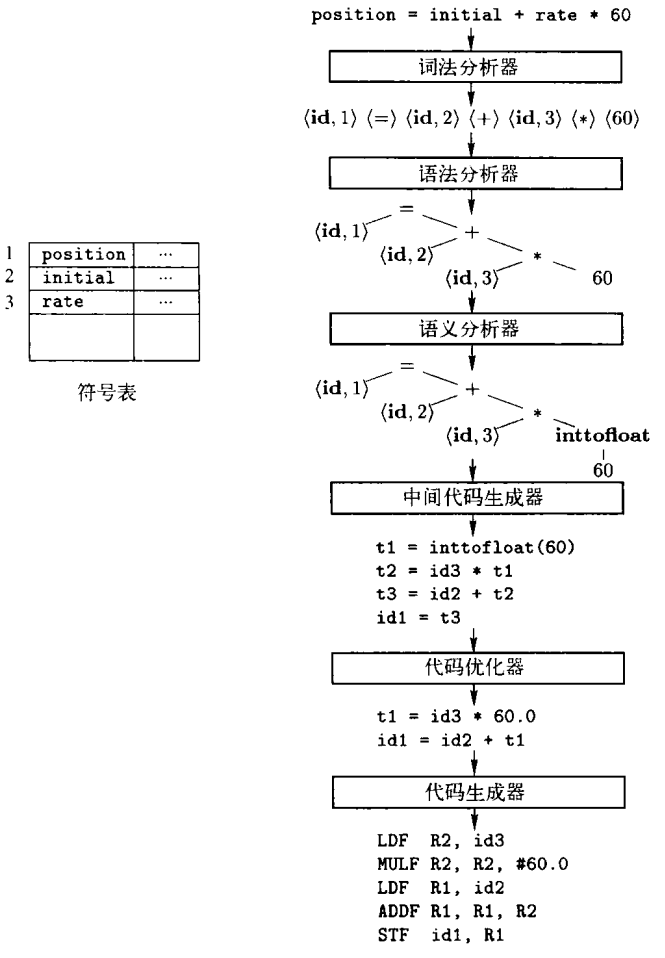


图 1-7 一个赋值语句的翻译

1.2.2 语法分析

编译器的第 2 个步骤称为*语法分析*(syntax analysis)或*解析*(parsing)。语法分析器使用由词法分析器生成的各个词法单元的第一个分量来创建树形的中间表示。该中间表示给出了词法分析产生的词法单元流的语法结构。一个常用的表示方法是*语法树*(syntax tree),树中的每个内部结点表示一个运算,而该结点的子结点表示该运算的分量。在图 1-7 中,词法单元流(1.2)对应的语法树被显示为语法分析器的输出。

这棵树显示了赋值语句

```
position = initial + rate * 60
```

中各个运算的执行顺序。这棵树有一个标号为 * 的内部结点,< **id**, 3 > 是它的左子结点,整数 60 是它的右子结点。结点 < **id**, 3 > 表示标识符 `rate`。标号为 * 的结点指明了我们必须首先把 `rate` 的值与 60 相乘。标号为 + 的结点表明我们必须把相乘的结果和 `initial` 的值相加。这棵树的根结点的标号为 =,它表明我们必须把相加的结果存储到标识符 `position` 对应的位置上去。这个运算顺序和通常的算术规则相同,即乘法的优先级高于加法,因此乘法应该在加法之前计算。

编译器的后续步骤使用这个语法结构来帮助分析源程序,并生成目标程序。在第 4 章,我们将使用上下文无关文法来描述程序设计语言的语法结构,并讨论为某些类型的语法自动构造高