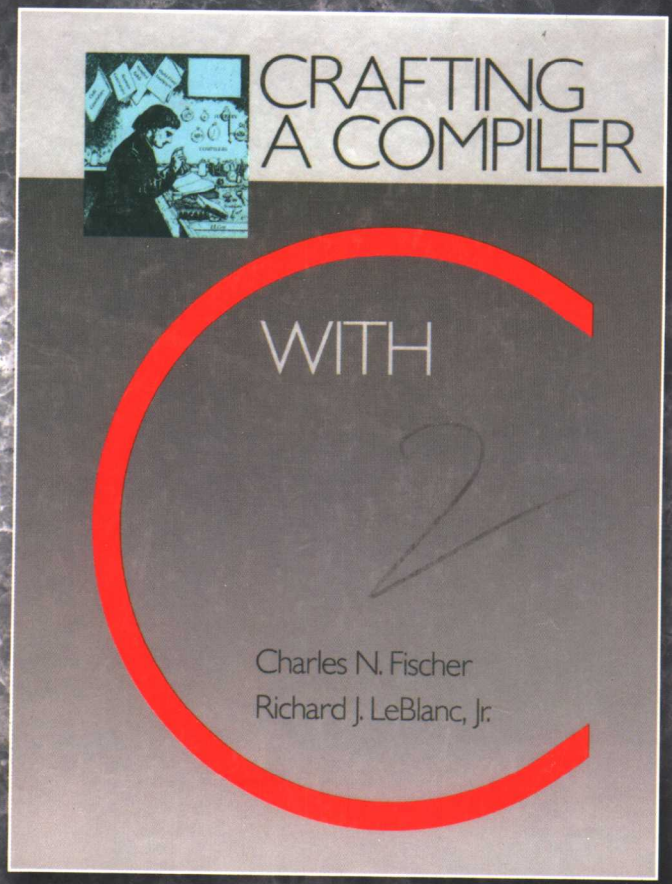
计 算 机 科 学 丛 书

编译器构造
C语言描述

(美) Charles N. Fischer 威斯康星大学麦迪逊分校 Richard J. LeBlanc, Jr. 佐治亚理工学院 著 郑启龙 姚震 译



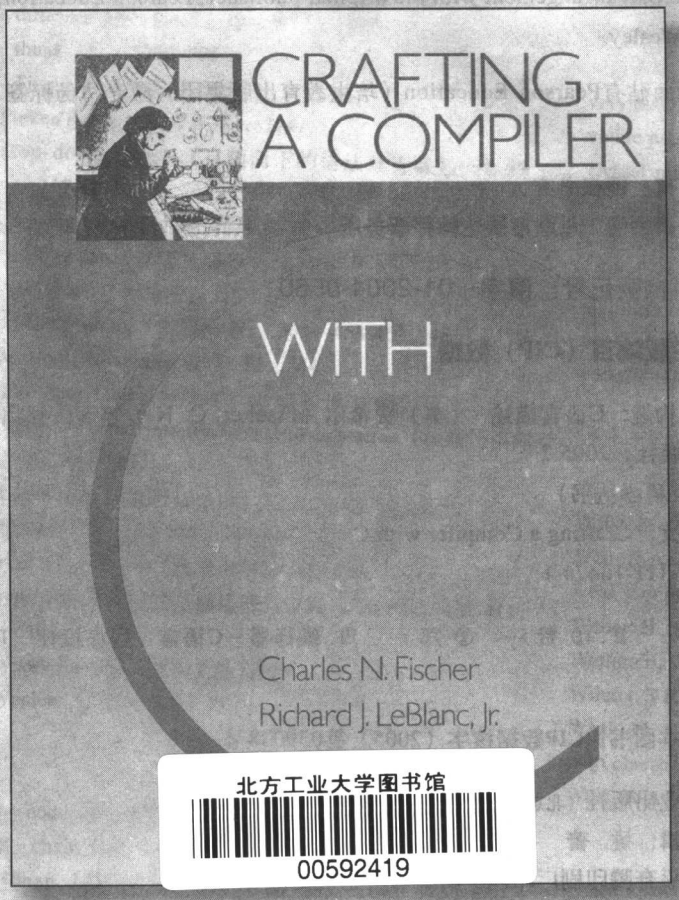
Crafting A Compiler with C

机械工业出版社
China Machine Press

计 算 机 科 学 丛

编译器构造
C语言描述

(美) Charles N. Fischer 威斯康星大学麦迪逊分校 Richard J. LeBlanc, Jr. 佐治亚理工学院 著 郑启龙 姚震 译



Crafting A Compiler with C

机械工业出版社
China Machine Press

本书均衡讲述了编译器设计中的理论与实现两大部分，详细讨论了标准编译器设计的相关主题（如自顶向下和自底向上的语法分析、语义分析、中间表示和代码生成），提供了创新的编译器构造方法，使读者可以从头至尾地学习如何设计一个可用的编译器。

本书是一本优秀的编译器构造方面的教材，适合于高等院校计算机专业的学生和使用C语言的专业程序员。

本书版权登记号：图字：01-2004-0560

图书在版编目（CIP）数据

编译器构造：C语言描述 /（美）费希尔（Fischer, C. N.）等著；郑启龙等译. -北京：机械工业出版社，2005.7

（计算机科学丛书）

书名原文：Crafting a Compiler with C

ISBN 7-111-16474-1

Ⅰ. 编 … Ⅱ. ① 费 … ② 郑 … Ⅲ. 编译器－C语言－程序设计 Ⅳ. ① TN762 ② TP312

中国版本图书馆CIP数据核字（2005）第039038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：姚 蕾

北京昌平奔腾印刷厂印刷 · 新华书店北京发行所发行

2005年7月第1版第1次印刷

787mm × 1092mm 1/16 · 34.25印张

印数：0 001-4 000册

定价：65.00元

凡购本书，如有倒页、脱页、缺页，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，还揭橥了学术的源变，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、从业人员较少的现状下，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图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray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、研究及庋藏。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，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，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，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步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步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的

指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

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，但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。教材的出版只是我们的后续服务的起点。华章公司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：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

译 者 序

当今计算机体系结构正处于从32位向64位跨越的时代，除了硬件本身的革命性设计之外，要想充分发挥64位硬件计算平台的效率并以此提高应用软件的性能，编译器作为现代计算机中最重要的系统软件之一，其作用是无可替代的。因此，编译器的原理与设计应该是计算机专业学生所必须学习和掌握的一门重要的专业基础课。尽管最终从事编译器构造这种浩繁而艰巨的工程的人只是少数，但通过对编译器原理的学习、对其设计技巧的掌握，可以使学生对诸如程序设计语言、数据结构和算法设计与分析等诸多计算机专业基础课程有更加深入的理解与体会。通过对编译器这个沟通软硬件的“翻译”桥梁角色的认识，还可以让学生对计算机软、硬件的相互依存、相互支持的关系有更进一步的领悟与洞察。而这一切的获得往往开始于选择一本好的教材、一本有益的参考书。可以这么说，好的选择是成功道路上坚实的第一步。

由Charles N. Fischer和Richard J. LeBlanc, Jr两位教授合著的本书就是一本这样的书。本书中文版的第一译者最早在11年前就接触了本书的原版，从中受益并采撷了部分知识用于当时的科研工作，之后在中国科学技术大学多次讲授“编译原理与技术”课程时也就一直把它作为重要的不可或缺的教学参考书。

正如本书的书名首先让人想到的，它是一本告诉人们如何用C语言去实际地构造编译器的书，这正是本书的最大特色——完美而均衡地覆盖了编译器的理论和实现的相关议题。本书使读者在阅读深奥而枯燥的理论的同时，又能享受精妙而细致的实现，虽精巧但又不落于旁支细节，既可信手拈来又给人以启迪与思考的空间，在精彩或重要之处作者更是不惜笔墨。另外，本书还是一本编译器构造理论方面的有价值的专著，书中不乏围绕构造一个编译器而进行的系统全面、内容详实的理论介绍。全书正文共分17章，其中包括了从最基本的词法分析到语法分析，再到语义处理、运行时存储环境、代码生成和代码优化，从符号表的组织到程序错误的处理等编译器各个典型的组成部分的周到的分析与讨论；所涉及的程序设计语言从较早期的Algol 60、Modula-2到美国国防部专用的、功能丰富但也十分复杂的Ada语言，从简单的Pascal语言到灵活的C语言，再到时下流行的专业程序员首选的C++语言等。

尽管本书反映的是截止到20世纪90年代初期的研究成果，但它精辟的理论分析和细致的实现描述却有力地影响着后来的编译器的设计与实现。本书译者曾在某国际著名软件公司的商用编译器所生成的目标代码中看到了本书所介绍的优化编译技术的踪影。对于这本内容丰富、理论与实践交融的专著，我们有理由相信它会成为对编译器感兴趣并有着不同需求的读者们的良师益友。

本书中文版的翻译工作由郑启龙和姚震完成。其中：姚震负责完成前言、第1章～第7章、附录A～附录F的翻译工作，郑启龙负责第8章～第17章的翻译工作并校对了全部译稿。

本书是两位译者第一次翻译国外计算机专业著作，由于水平有限，加之时间仓促，其中难免有这样或那样的错误与不当之处，恳请各位读者批评指正。

郑启龙　姚震

2005年4月于

中国科学技术大学

前　言

本书以作者实现编译器和开发编译器构造课程的经验为基础，介绍了编译器构造的实际方法。其宗旨是使读者不仅能够对编译器的所有组件有深入的理解，而且能够对编译器的各组件如何实际组合、构成一个可以工作的编译器有感性认识。我们相信这种理念是本书一个与众不同的特色。因为我们专注于对现代编译器构造技术的介绍，所以强调尽可能地使用编译器工具生成编译器的组件。（附录B～F中所述工具的源代码，可以从出版社网站下载。）

本书和*Crafting a Compiler*一书基本相同，只是其中的算法和伪代码示例使用C而不是Ada语法。由于C语言在编译器课程中广泛使用，因此许多教师认为这样的版本会很有价值。为使所有伪代码即使对于那些不是C语言专家的读者也尽可能易读，我们使用了一些标准C语法的扩展（在下面描述），而且并不总是使用通用的C语言惯用法。由于本书作者均不是熟练的C程序员，因此我们感谢Arnold Robbins对本书做了从Ada语言到C语言的转换工作，并给出很多编辑上的建议。

教科书和参考书

作为教学用书，本书主要面向近15年来我们开发的一种课程组织结构。但是，本书的使用可以非常灵活，已经用于从为期10周的3学分高年级本科生课程到为期3个月的6学分研究生课程教学中。本书也可作为一本有价值的专业参考书，因为它完整覆盖了对编译器编写者和设计者有着重要实际意义的技术。

课程设计方法

本书全面地覆盖了与构造编译器相关的理论主题。而且，一个密切相关的课程设计实现也是我们课程组织的重要组成部分。因此，本书也是面向课程设计的。附录A包含一个称为 Ada/CS的语言定义，它主要是Ada程序设计语言的一个重要子集。但出于教学目的，这里介绍的Ada/CS并非Ada程序设计语言的一个严格子集。针对使用本书的课程，我们建议的课程设计可以涉及部分或完整的Ada/CS实现，具体情况要根据课程的长度、层次以及授课教师的要求来定。

本书第2章介绍并讨论一个针对非常简单的Ada/CS子集的递归下降编译器。将上述课程设计实施为该编译器的扩充，可以让学生即使在短到一个学期的课程中也能完成一个较大的课程设计。在时间充裕的情况下，这种扩充方法同样有价值。要求学生阅读并扩充一个实际的程序，可使他们获得在许多计算机科学课程中难以得到的重要经验。这也教会他们如何将编译器的各部分组合起来，而这种知识是难以用其他任何方式来讲授的。

C程序设计语言

本书中的示例伪代码采用基于ANSI C的语法编写。从PC机到大型机直到超级计算机，C语言在许多计算环境中都是流行的语言，它小巧且富于表达力、高效，同时通常具有很好的可移植性。C语言最近已成为美国国家标准（ANSI 1989）。支持标准C语法的编译器也已广泛可用，而且将会更加普及。因此，我们选择为示例程序使用ANSI语法而不是在Kernighan and Ritchie（1978）中描述的更广为人知的语法。而Kernighan和Ritchie的新书（1988）则是ANSI C的优秀参考书。

为了将算法尽可能以最易读的方式描述（而不关注语法细节），我们以几种方式扩充了C语言。首先，在正文的几个地方使用了*匿名联合*（anonymous union），该特性借用自C++（Stroustrup 1986）。一个匿名联合看起来像这样：

```
struct somestruct {
    int elem1;
    union {
        float  f2;
        int    i2;
        double d2;
        long       l2;
    };
    long   elem3;
} s;
 . . .
s.elem1 = 10;
s.f2 = 10.0
 . . .
```

注意：该结构的**union**成员没有名字，联合中的元素作为结构的元素被直接引用。在传统的C语言中，同样的行为通常通过宏预处理器获得：

```
struct somestruct {
    int elem1;
    union {
        float  u_f2;
        int    u_i2;
        double u_d2;
        long       u_l2;
    } u;
    long   elem3;
} s;
#define f2 u.u_f2
#define i2 u.u_i2
#define d2 u.u_d2
#define l2 u.u_l2
 . . .
s.elem1 = 10;
s.f2 = 10.0
 . . .
```

其次，从第10章开始，使用*匿名结构*（anonymous structure）作为匿名联合的成员。实际编程中，这些结构需要通过上面描述的预处理器方案实现。

最后，在许多高层伪代码示例中，使用下列*构造函数*（constructor）机制来创建结构表达式：

```
struct something {
    int elem1;
    char *elem2;
} v;
 . . .
v = (something) {
        .elem1 = 1;
        .elem2 = "string";
    };
```

尽管这种结构不能使用宏来代替，但其含义显而易见。

一般情况下，在使用高层伪代码而不是正规C语言时，我们将这些图标记为“算法”而不是“程序”。

与*Crafting a Compiler*一样，在使用本书的课程中不必使用任何特定的语言来实现课程设计。不论选择哪种实现语言，伪代码都是极佳的设计描述手段。此外，我们提供的词法分析器和语法分析器生成工具生成的是表格而不是程序，因此它们可用于任何语言环境中。为了那些使用C语言实现课程设计的读者，我们也讨论Lex和Yacc的使用。

Ada的角色

我们所建议的课程设计和对语言特性的讨论都基于Ada语言，因为它事实上包含了在语义分析部分

我们希望讨论的所有语言特性。假如选择任何其他语言作为基础（例如Modular-2），就必须描述很多扩充以讨论编译这些东西（比如exit语句、异常处理和操作符重载）的技术。

使用本书的学生当然不必熟悉Ada语言。附录A中的Ada/CS介绍可以作为在语义分析部分所讨论的Ada语言特性的教程。在适当的情况下，我们也考虑从其他语言（包括Modula-2、Pascal、C、ALGOL-60、ALGOL-68以及Simula）中抽取的语言特性的翻译。

各章描述

本书作为入门课程，可以讲授第1～4章、第5章或第6章、第7章以及第8～13章和第15章的部分内容。

本书作为高级课程，可以包含讲述语法分析的各章（第5～6章）的内容，以及第8～13章和第15章加上第14、16和17章的高级主题部分。

第1章 绪论

在本书的一开始概述编译过程，强调从一组组件构造编译器的概念，介绍使用工具生成其中一些组件的思想。

第2章 一个简单编译器

介绍一个非常简单的语言Micro，讨论编译Micro的编译器的每个组件。本章包含Micro编译器的部分文本（用Ada语言编写）。而对更全面的Ada子集的语言特性的编译则引出了以下各章所介绍的技术。

第3章 词法分析——理论与实践

介绍构造编译器词法分析组件的基本概念和技术。本章中的讨论既包括开发手写的词法分析器，又包括使用词法分析器生成工具实现表驱动的词法分析器。

第4章 文法和语法分析

介绍形式语言概念和文法的基本知识，包括上下文无关文法、BNF表示法、推导和语法分析树。由于在自顶向下和自底向上的语法分析技术的定义中都用到First和Follow集，本章也对它们进行了定义。本章还包括对语言和文法关系的讨论。

第5章 LL(1)文法及分析器

介绍语法分析的第一种方法——自顶向下的语法分析。讨论递归下降和LL(1)，重点放在后者。语法分析器生成器的使用是本章的重点。

第6章 LR分析

介绍另一种语法分析方法——自底向上的语法分析。介绍LR、SLR和LALR语法分析概念及其与LL技术的比较。语法分析器生成器的使用也是本章的重点。

第7章 语义处理

本章结合自顶向下和自底向上语法分析器来介绍语义处理的基本原理。主题包括：不同编译器组织方式的比较，（在自顶向下的语法分析中）向文法增加动作符号，（在自底向上的语法分析中）为“语义钩子”（semantic hook）重写文法，定义语义记录和使用语义栈，检查语义正确性，产生中间代码。

第8章 符号表

本章强调符号表的使用，它作为一个抽象组件由编译器的其他组件通过精确定义的接口使用。本章介绍其可能的实现，随后讨论用符号表处理嵌套的作用域和其他用来定义可从外围作用域（例如记录和

Ada中的包）访问的名字的语言特性。

第9章 运行时存储组织

介绍运行时存储管理的基本技术，包括静态分配、基于栈的分配和一般动态（堆）分配的讨论。

第10章 处理声明

讨论处理类型、变量和常量声明的基本技术。这些内容的组织基于处理特定语言特性的语义例程。

第11章 处理表达式和数据结构引用

概述处理变量引用和算术、布尔表达式的语义例程。在对变量引用的讨论中，包括针对数组元素和记录域的地址计算方法。在本章及随后两章中，强调检查语义正确性和产生被目标代码生成器使用的中间代码的技术。

第12章 翻译控制结构

本章的重点是针对像**if**语句、**case**语句和各种循环结构等特性的编译技术。强调使用语义栈或语法树简化这些结构的处理。这些结构可以嵌套并扩展到任意规模的程序文本中。学生应通过特定的方法来了解这种通用技术的优点。

第13章 翻译过程和函数

介绍处理子程序声明和调用的技术。由于这个主题的很多复杂性涉及参数，本章提供了很多材料来讲述创建参数描述、检查子程序调用中实际参数的正确性以及不同参数模式所需的代码生成技术。这里讨论运行时活动栈的概念，给出实现它所必需的支持例程。

第14章 属性文法和多遍翻译

多遍翻译由遍历中间代码形式模拟。本章强调信息流的属性模型。

第15章 代码生成和局部代码优化

介绍代码生成器，它作为一个独立组件，将语义例程生成的中间代码翻译为编译器最终的目标代码。介绍像指令选择、寄存器管理和寻址模式的使用等主题。本章还包括对基本块优化的讨论。

第16章 全局优化

本章的焦点是那些通过适度的努力可以生成有用代码改进的实用技术。因此，本章的主要部分包括全局数据流分析、优化子程序调用和优化循环。

第17章 现实世界中的语法分析

本章包括实现一个实际编译器必需的两个主要课题：语法错误处理和表压缩。错误处理部分介绍用于递归下降、LL和LR分析器的错误恢复和错误修复技术。表压缩技术用于LL和LR分析器，以及词法分析器表和任何其他需要用快速访问稀疏表中的元素来高效存储的场合。

致谢

很多人为*Crafting a Compiler*和本书作出贡献。首先，感谢威斯康星大学麦迪逊分校CS 701/2和乔治亚理工学院ICS 4410中的许多学生，他们使用了本书最初版本和最终成为本书部分章节的讲义。此外，还要感谢两所学校里使用我们的讲义授课的许多教师。其中包括Raphael Finkel、Marvin Solomon和K. N. King，他们为我们的讲义贡献了部分材料；还有Nancy Lynch、Martin McKendry、Nancy Griffeth和David Pitts，他们也使用了我们的讲义。乔治亚理工学院的Arnold Robbins提供了正文中出现的所有C语言代码。他还提供了一些章节的习题、封面设计的想法以及许多对我们写作风格很有用的建议。G A

Venkatesh、Will Winsborough和Felix Wu提供了大部分习题的参考答案。Jon Mauney、Gary Sevitsky、Robert Gray和Felix Wu开发了附录中描述的编译器工具。Kathy Schultz出色地完成了手稿的最终订正，Sheryl Pomraning出色地完成了美工工作。

我们感谢C. Wrandel Barth、Jean Gallier、James Harp、Harry Lewis、Eric Roberts和Henry Shapiro，他们对我们最初的提议和样章提供了有价值的反馈。我们非常感激Steve Allan、Henry Bauer、Roger Eggen、Norman Hutchinson、Sathis Menon、Jim Bitner、Charles Shipley、Donald K. Friesen、Donald Cooley、Susan Graham、Steve Zeigler，尤其是Paul Hilfinger和Alan Wendt——他们作为审阅者提供了大量意见，使我们在很长时间内忙于完成本书及其早期版本。

特别感谢Alan Apt,我们杰出的编辑，感谢他的耐心和鼓励。同时感谢Benjamin/Cummings的所有员工，他们如此热情地支持我们的工作；感谢Todd Proebsting和Chris Sabooni，他们细心地检查我们的手稿；感谢Alyssa Weiner以及工作组的其他成员的细致工作；感谢Jane Rundell和Impressions的所有员工。

最后，感谢Miriam Robbins，她在Arnold努力将大量的Ada算法转换为清晰而精确的C算法时表现出了极大的耐心。

目　录

第1章 绪 论

1.1 概述和历史

编译器是现代计算技术的基本组成部分。它们担任*翻译器*（translator）的角色，将面向人的*程序设计语言*（programming language）转换成面向计算机的*机器语言*（machine language）。对于大多数用户来说，编译器可以被看作是一个“黑箱”，执行图1-1所示的转换。

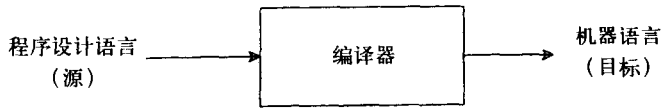


图1-1 用户眼中的编译器

编译器事实上允许所有计算机用户忽略与机器相关的机器语言细节。因此，编译器允许程序和程序设计的专业技术成为*与机器无关的*（machine-independent）。在计算机的数量和种类持续爆炸性增长的时代，这是一项特别有价值的优点。

术语编译器（compiler）是在20世纪50年代早期由Grace Murray Hopper创造。那时翻译被看作是“编译”一系列从程序库中选出的子程序。实际上，编译（正如我们现在所知道的）那时就被称为“自动程序设计”，而对它究竟成功与否却存在着普遍的怀疑。今天，程序设计语言的自动翻译已经是一个既成事实，但程序设计语言翻译器仍被称为编译器。

现代意义上的最早的真正编译器是20世纪50年代晚期的FORTRAN编译器。它们为用户提供了一个面向问题的、与机器密切相关的源语言，并执行一些相当有挑战性的“优化”以产生高效的机器代码。而这种优化对于成功地和当时占统治地位的汇编语言进行竞争来说是必要的。这些FORTRAN系统证明了经过编译的高级（即较少依赖机器的）语言的生命力，并为随后语言和编译器的大量涌现铺平了道路。

第一个FORTRAN编译器花费了18个人年来构造。早期的编译器都是专用结构；组件和技术通常随着编译器的构造而被发明出来。构造编译器是一项复杂和代价高昂的工作。今天，编译技术已被很好地理解，一个简单的编译器可以在几个月中构造出来。尽管如此，构造高效可靠的编译器仍然是一项富有挑战性的任务。我们的方法是掌握编译的基本原理，随后研究高级主题以及许多新近发明的重要的技术革新。

通常认为编译器是将程序设计语言翻译为机器语言指令。然而，编译器技术是广泛适用的并已经用于许多过去未曾预料到的领域。例如，文本格式化语言已经变得日益复杂。格式化程序（formatter），像UNIX®的*nroff* 和*troff*，实际上就是编译器，它们将文本和格式化命令翻译为非常复杂的排版命令。像所有成功的语言一样，*nroff* 和*troff* 已被推广，用来提供新的功能。例如，像*eqn*（用来处理公式）、*tbl*（用来格式化表格）和*pic*（用来绘制图形）这样的预处理器包非常类似于通常用来扩展现有程序设计语言的预处理器（例如，Ratfor FORTRAN预处理器）。程序设计语言和编译器设计的持续挑战之一就是发明允许用简单且自然的方式扩展现有语言的机制。

创建VLSI电路是另一项可以被建模为从高级源语言到低级目标语言的翻译任务。在此情形中，*硅*