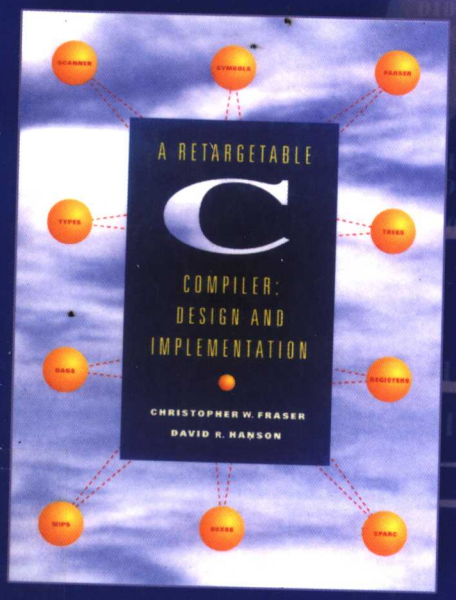
国外计算机科学教材系列

可变目标C编译器

——设计与实现

A Retargetable C Compiler

Design and Implementation



[美] Christopher W. Fraser
David R. Hanson 著

王 挺 黄 春 等译

PEARSON Addison Wesley

PHEI

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

国外计算机科学教材系列

可变目标C编译器
——设计与实现

A Retargetable C Compiler
Design and Implementation

[美] Christopher W. Fraser
David R. Hanson 著

王 挺 黄 春 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书系统地介绍了可变目标ANSI C编译器lcc的设计方法和实现技术。lcc是一个实用的编译器，能够为不同的目标机器（如MIPS R3000、SPARC、Intel 386及其后续产品）生成代码。本书结合lcc的具体实现，详细讲述了存储管理、符号表、词法分析、语法分析、中间代码生成、优化、目标代码产生等编译程序的各个部分。全书共分19章，在各章之后均附有练习。

与其他介绍编译技术的教材相比，本书特色鲜明，实用性强，适合作为高等院校计算机专业的编译原理课程的教材或参考书，对从事编译相关工作的技术人员也有很好的参考价值。

版权贸易合同登记号　图字：01-2002-4055

图书在版编目（CIP）数据

可变目标C编译器——设计与实现/（美）弗雷泽（Fraser, C. W.）著；王挺等译.
北京：电子工业出版社，2005.1
（国外计算机科学教材系列）
书名原文：A Retargetable C Compiler: Design and Implementation
ISBN 7-5053-9922-5

Ⅰ.可…　Ⅱ.①弗…②王…　Ⅲ.编译码器－C语言－程序设计－教材　Ⅳ.①TN762②TP312

中国版本图书馆CIP数据核字（2004）第123528号

责任编辑：杜闽燕
印　　刷：北京兴华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 1092　1/16　　印张：27.5　字数：776千字
印　　次：2005年1月第1次印刷
定　　价：43.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

出 版 说 明

21世纪初的5至10年是我国国民经济和社会发展的重要时期，也是信息产业快速发展的关键时期。在我国加入WTO后的今天，培养一支适应国际化竞争的一流IT人才队伍是我国高等教育的重要任务之一。信息科学和技术方面人才的优劣与多寡，是我国面对国际竞争时成败的关键因素。

当前，正值我国高等教育特别是信息科学领域的教育调整、变革的重大时期，为使我国教育体制与国际化接轨，有条件的高等院校正在为某些信息学科和技术课程使用国外优秀教材和优秀原版教材，以使我国在计算机教学上尽快赶上国际先进水平。

电子工业出版社秉承多年来引进国外优秀图书的经验，翻译出版了“国外计算机科学教材系列”丛书，这套教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。这些教材涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。同时，我们也适当引进了一些优秀英文原版教材，本着翻译版本和英文原版并重的原则，对重点图书既提供英文原版又提供相应的翻译版本。

在图书选题上，我们大都选择国外著名出版公司出版的高校教材，如Pearson Education培生教育出版集团、麦格劳－希尔教育出版集团、麻省理工学院出版社、剑桥大学出版社等。撰写教材的许多作者都是蜚声世界的教授、学者，如道格拉斯·科默（Douglas E. Comer）、威廉·斯托林斯（William Stallings）、哈维·戴特尔（Harvey M. Deitel）、尤利斯·布莱克（Uyless Black）等。

为确保教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本系列教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师、博士，也有积累了几十年教学经验的老教授和博士生导师。

在该系列教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括对所选教材进行全面论证；选择编辑时力求达到专业对口；对排版、印制质量进行严格把关。对于英文教材中出现的错误，我们通过与作者联络和网上下载勘误表等方式，逐一进行了修订。

此外，我们还将与国外著名出版公司合作，提供一些教材的教学支持资料，希望能为授课老师提供帮助。今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

电子工业出版社

教材出版委员会

译 者 序

编译器构造原理和技术可以说是计算机科学中理论与实践相结合的最好典范。到目前为止，大多数教材都是介绍编译器构造原理的，很少有详细介绍实用编译器构造的专门书籍。在编译原理课程的教学过程中，如何设计和组织实验一直是一个难题。这主要是因为，任何实用的编译器往往都是庞大的程序，而小的实验编译器又难以反映编译程序构造的许多重要技术。本书可以说是弥补了传统编译教材在实践方面的缺陷，如果希望向学生展示实用编译器是如何设计的，本书应该是最佳选择。

lcc编译器是一个具有产品级质量的用于研究的C编译器，在UNIX界广为流行。本书深入到lcc编译器的内部，在代码级对该系统的设计和实现进行了详细的介绍。全书共分19章，详细讲述了存储管理、符号表、词法分析、语法分析、中间代码生成、优化、目标代码产生等编译程序的各个部分。本书特别针对3种目标机器（MIPS、SPARC和Intel 386）介绍了代码生成器的设计。通过学习上述内容，读者可以深入了解产品级编译程序设计中的许多关键技术，对于如何设计和实现一个实用的编译器有具体、真切的认识，这是其他教材无法达到的效果。

本书的两位作者都具有深厚的教学和研究背景。Christopher W. Fraser从1975年就开始研究编译技术，尤其对于从紧缩规范自动产生代码生成器这一技术有深入的研究，在该领域发表了多篇论文。他提出了可变目标的窥孔优化方法，该方法被广为流行的C编译器——GCC所采纳。从1977年到1986年，Fraser在亚利桑那大学从事计算机科学（包括编译技术）的教学工作。1986年以后，他在AT&T贝尔实验室主持计算技术的研究工作。David R. Hanson是普林斯顿大学计算机科学教授，具有20多年的研究程序语言的经验，主持了与贝尔实验室的合作研究，是lcc的开发者之一。

本书是作者的教学、科研和开发思想以及经验的总结，读者可以从字里行间体会到两位作者在编译器的研究和设计方面的造诣。

国防科学技术大学计算机学院从事编译原理课程教学和科研工作的几位教师共同完成了本书的翻译工作。全书由王挺、黄春、刘金红、张晓燕和陈耀东负责翻译，由王挺和黄春通篇整理。由于时间和水平有限，翻译错误在所难免，恳请读者指正。

前　言

编译器是程序员使用的关键工具，程序员每天都在使用编译器，并且非常依赖于其正确性和可靠性。编译器必须接受程序语言的所有标准定义，以便源代码可以实现跨平台的可移植性。编译器必须生成高效的目标代码，但更重要的是，编译器必须生成正确的目标代码，只有可靠的编译器才能生成可靠的应用程序。

编译器本身是一个大而复杂的应用程序，值得我们深入分析研究。本书介绍了ANSI C语言编译器lcc的大部分实现，本书对编译器的介绍方式与B. W. Kernighan 和P. J. Plauger合著的“Software Tools”（Addison-Wesley，1976）一书对文本处理（例如文本编辑和宏处理）的介绍类似。研究实用的工具软件，是学习软件设计和实现技术的最好方法。本书在代码级详细介绍了一个实用的编译器，该编译器的完整源代码可在ftp.cs.princeton.edu（128.112.152.13）服务器的pub/lcc目录下，通过匿名ftp服务得到。

lcc不是一个研究系统，而是一个实用的编译器产品。从1988年开始，lcc就用于编译实际程序，现在每天都有数百名C程序员在使用它。由于本书详细分析了lcc编译器的设计与实现，因此用于介绍相关支撑材料的篇幅较少，仅展示了涉及到的理论知识，而更为系统的编译技术的介绍可以参见其他教材。本书有意省略一些涉及琐碎和重复实现的语言特征，而将这部分内容作为练习。

显然，本书将使读者对编译器的构造有更多的了解。然而只有少数程序员需要了解编译器的设计与实现，大多数程序员从事的是应用程序或其他系统程序的开发。但是，基于以下4个原因，大多数C程序员都可以从本书中受益。

首先，一般来说，如果程序员能够理解C编译器的工作原理，通常可以成为较好的程序员，特别是较好的C程序员。编译器设计者必须全面准确地理解C语言的每一个特性，程序员通过学习这些特性的实现，能够更好地掌握语言本身及其在现代计算机上的高效实现。

第二，大多数程序设计教材都是通过一些精简的示例来说明编程技巧的，但大多数程序员都是在从事大型程序的开发，在开发过程中需要不断修改程序，很少有带详细说明的示例可以作为大型程序设计的参考。lcc不是完美的，但是本书详细说明了该程序的优缺点，可以作为大型程序开发的参考。

第三，编译器是计算机科学中理论与实践相结合的最好典范。lcc展示了理论与实践的相互作用及其精美的结果，展示了实践需求牵引理论的发展，这些都可以清楚地从代码中找到。通过一个真实的程序来研究这些相互作用，可以帮助程序员理解何时、何地以及如何运用不同的技术。此外，lcc也阐明了众多的C编程技术。

第四，这本书本身是一个文本程序（literate program），如同D.E. Knuth所著的“TeX: The Program”（Addison-Wesley，1986）一样，本书包括lcc的源代码及说明。为了方便读者理解，本书并未按源程序的顺序对程序代码进行讲解，而是有意进行了调整。

无论是对于在校学生还是专业技术人员，本书都非常适合自学使用。本书为lcc提供了说明完整的源代码，希望进行编译技术实践的，以及在需要使用或实现基于语言的工具和技术的应用领域（如用户接口）中工作的专业人员，将会对本书感兴趣。lcc的相关信息可通过以下地址获得：www.cs.princeton.edu/software/lcc。

本书全面而真实地展示了一个大型软件系统，也可作为软件工程课程的分析实例。

对于编译课程来说，本书弥补了传统编译教材的不足。本书介绍了C编译器的一种实现方法，而传统教材主要介绍编译过程中遇到的各种问题的解决算法，因此传统教材受篇幅限制只能介绍一些实验性的编译器，代码生成也通常面向较高的级别，以避免与具体的机器相关。

因此，许多教师要求学生完成接近实际的编译器项目，使学生获得实践经验。通常，教师必须从头开始编写编译程序，而学生复制其中的大部分，修改后利用其余的部分。然而，由于编译器只是实验性的，文档往往显得不够充分，这种情形使教学双方都不满意。本书通过对一个实际编译器的大部分程序进行文档说明，并提供源代码，为教师提供了一种新的选择。

本书介绍了完整的代码生成器，代码生成面向 MIPS R3000、SPARC 和 Intel 386 及其后续体系结构等不同的平台。本书利用了最新的研究成果，根据目标机器的紧缩规范（compact specification）生成代码生成器。这些方法使得我们能够针对多种机器展示完整的代码生成器，这是其他书籍无法做到的。通过介绍多个代码生成器，既避免了本书依赖于单一的机器，又有助于学生了解如何设计可变目标的软件。

教师布置的作业可以是增加编译器接受的语言特征、优化、改变目标机器等。本书如果与传统教材配合使用，也可以要求学生使用不同的算法代替现有的模块，作为实践作业。如果以实现一个实验编译器作为实践作业，则可能在低级基础结构和重复的语言特征上花费大量的时间。采取上述方法，就能够更接近实际的编译器工程实践。本书的许多练习都涉及编译器工程问题。

除传统的编译目的外，lcc 也有其他用途。例如，它可以用于构建一个C程序浏览器，或者根据声明来生成远程过程调用的桩函数（stub），也能用于语言扩充、新的计算机体系结构和代码生成技术的实验。

本书假设读者熟悉某种计算机的C 和汇编语言，了解编译器的概念，理解编译器的工作原理，同时要求读者的数据结构和算法知识达到一般本科水平，例如，R. Sedgewick 所著的“Algorithms in C”（Addison-Wesley，1990）一书中的内容对于理解 lcc 就足够了。

致谢

感谢众多lcc用户，他们来自于AT&T贝尔实验室、普林斯顿大学和其他地方，他们忍受了许多程序中的错误，并提供了有价值的反馈。感谢Hans Boehm、Mary Fernandez、Michael Golan、Paul Haahr、Brian Kernighan、Doug McIlroy、Rob Pike、Dennis Ritchie 和Ravi Sethi。Ronald Guilmette、David Kristol、David Prosser 和Dennis Ritchie在 ANSI 标准的许多细节及其解释方面提供了非常有价值的信息。David Gay帮助我们改造了PFORT数值计算软件库，以作为lcc代码生成的测试用例，非常有价值。

Jack Davidson、Todd Proebsting、Norman Ramsey、William Waite 和David Wall仔细阅读了我们的代码和文字，大大提高了二者的质量。还要感谢Steve Beck，他安装并改进了书中用到的字体；感谢 Maylee Noah，他使用 Adobe Illustrator 制作完成了本书的图片。

Christopher W. Fraser
David R. Hanson

目　录

第1章 引 论

编译器可以将源代码翻译成目标机器上的汇编代码或目标代码。可变目标编译器能够针对不同的目标机器生成代码。编译器中与机器有关的部分被独立成模块，针对不同的目标机器，这些模块可以方便地进行替换。

本书描述了一个可变目标的ANSI C编译器lcc，重点介绍其实现。大多数编译器教材注重于编译算法，只附带介绍了实验性的编译器。而本书与其他教材不同，描述了一个完整的ANSI C实用编译器，包括针对3种目标机器的代码生成器。本书仅介绍了lcc用到的编译理论。

1.1 文本程序

本书不仅描述了lcc的实现全貌，其本身就是系统的实现。针对文字编程的noweb系统根据同一个文本源程序生成了文本和代码。该源程序中包括了交替出现的说明和带标记的代码片段。代码片段按照有利于描述程序的顺序出现，也就是说，本书说明代码的顺序并不与原来C语言程序中的一样。程序noweave以这种源程序为输入，生成了本书英文版原稿，包括大部分代码和所有文本。程序notangle按照书中要求的顺序抽取了所有的代码。

代码片段包括源代码和对其他代码片段的引用。代码片段的定义是以尖括号括起来的标记开始的，例如下列代码：

```
⟨a fragment label 1⟩≡                                          1
  sum = 0;
  for (i = 0; i < 10; i++) ⟨increment sum 1⟩

⟨increment sum 1⟩≡                                             1
  sum += x[i];
```

这段代码的功能是计算数组x的所有元素之和，其中<*increment* sum 1>显示了代码片段是如何被引用的。多个代码片段可以有相同的名字，程序notangle可以把这些名字相同的定义连接成一段代码。对于这些连续的程序段定义，程序noweave使用+≡而不是≡来表示：

```
⟨a fragment label 1⟩+≡                                         1
  printf("%d\n", sum);
```

程序段定义类似宏定义，程序notangle抽取整个程序的过程是从一个程序段开始的。如果其定义引用了其他程序段，则把引用的程序段扩展进来，如此重复进行。

程序段定义有助于读者通读这些程序。每个程序段的名字的末尾标有一个数字，该数字是这个程序段开始定义的页码。如果没有数字，则表示该程序段未在本书中定义。每个后续的定义都指明了前一个定义，如果有后续定义，还将指明下一个定义。比如14指明当前定义的前一个定义在第14页，31指明下一个定义在第31页上。这种标志实际上把一个程序段的所有定义连成了一个双向链表，向前指针指向前一个定义，向后指针指向下一个定义。链表中第一个定义的向前指针和最后一个定义

的向后指针被省略。这些链表是完全的：如果一个程序段的部分定义在某页中出现，则可以通过这些页码引用找到相关的定义部分。

大多数程序段也指明了该程序段在哪些页中被使用，如上例中<*increment* sum>定义后面的数字1，表明该程序段在第1页使用。下面可以看到，根程序段（定义模块的程序段），以及经常使用的程序段，则没有加这些标志。

程序notangle还实现了C语言的一个扩展。较长的字符串文本可以分成若干行，每行的末尾用下划线结尾。notangle可以剔除后续行的前导空格，把各行连成一个字符串。第90页中error的第一个参数就是这种扩展功能的例子。

1.2 如何使用本书

一般来说，应该从前往后阅读本书，但也有几种可能的变通方法：

- 第5章介绍了编译器前端和后端的接口，这一章的内容尽可能做到独立。
- 第13章~第18章介绍了编译器的后端。只要读者了解了编译器前端和后端的接口，仅须简单了解前面的章节，就可以直接阅读这些内容。事实上，许多读者甚至能够替换编译器前端或后端，而不需要对另一部分内容做更多的了解。
- 第16章~第18章介绍了与3种目标机器MIPS、SPARC、Intel 386及其后续结构相关的模块。这3章相互独立，读者可以阅读其中的任意几章。如果读者阅读了多章的内容，就会发现在内容上有一些重复，但是由于大多数通用的代码已经分解出来并放在第13章~第15章中介绍，少许重复还是可以容忍的。

本书的部分内容采取自底向上的方式描述lcc。例如，管理存储、字符串和符号表等章节介绍了一些最底层或接近最底层的函数，理解这些函数不需要太多的相关知识。

本书的另外一些内容则采取自顶向下的方式进行描述。例如，表达式分析、语句和声明等章节，从最顶层开始介绍，采取自顶向下的方式，先给出一些函数和程序段的使用及其功能简介，然后再介绍这些函数和程序段的细节。

本书还有一些地方采取了自顶向下和自底向上相结合的方式进行介绍。也许采取一致的方式进行介绍会更好，但是通常难以做到这一点。与大多数编译器一样，lcc包括相互之间递归调用的函数。所以，试图完全地先介绍调用程序再介绍被调用程序，或者先介绍被调用程序再介绍调用程序，都是不可能的。

有些程序段在读代码之前解释起来比较容易，而有些则在阅读代码后解释更容易一些。如果理解一个代码段有困难，不妨先看看该代码段前后的文字，可能会事半功倍。

lcc的大多数代码都在教材中出现了，但有些程序段只是加以使用而并未给出定义。在这些程序段中，有些由于篇幅限制被省略了，有些则是因为它们实现的是语言扩展功能、可选的调试帮助或重复的成分。例如，只要了解了处理C语言的for语句的代码，处理do-while语句的代码就不必介绍得太多。惟一全部省略的是解释lcc对C语言的初始化机制的处理，这是因为它既冗长乏味，又无助于其他知识的理解，所以就省略了这部分内容。只使用而未给出定义的程序段很容易识别：这些程序段名字的后面没有页码。

另外我们还省略了断言。lcc包含了几百条断言，大多数是关于参数或数据结构假设的断言。其中一个是assert(0)，它表示程序不应执行到该状态。例如，如果分支语句需要确保测试表达式的所有值都有相应的真正的处理分支，那么，就可以在默认分支中加入assert(0)。

1.3 概述

lcc能够把源程序翻译成汇编程序代码。下面的例子说明了这一转换的各个阶段，介绍了lcc的主要组成和数据结构。每一阶段都把程序变换成另一种表示方式，例如：预处理后的源程序、单词、树、无环路有向图及这些图的序列。例如最开始的源代码是：

```
int round(f) float f; {
        return f + 0.5;  /* truncates */
}
```

round没有函数原型，所以参数f作为double类型的数据传送，round在入口处将其转换成float。然后round将f加上0.5，再把结果截尾转换成整数并返回。

第一个阶段是由C的预处理程序进行宏扩展、引入头文件、选择条件编译代码等工作。虽然起源于UNIX系统，但目前lcc可以在DOS和UNIX系统下运行。与许多UNIX编译器一样，lcc使用独立的预处理器，并且预处理器作为一个独立的进程执行，但这部分内容不在本书范围之内。我们经常使用GNU C编译器的预处理程序。

典型的预处理程序读取上面的源程序并产生如下代码：

```
# 1 "sample.c"
int round(f) float f; {
        return f + 0.5;
}
```

由于本例中没有使用预处理特性，所以预处理程序没有其他效果，只是去除了注解，并增加了一个#命令，把文件名和源代码的行号告诉编译器，以便诊断错误时使用。这段例子代码中的起始行号显然为1，但是，如果源程序中包含多个#include指令，预处理后每个被包括进来的文件都用一对#指令括起来，指令中都包含各自的行号。

预处理程序工作完成后，编译器马上开始工作，首先由词法分析器（lexical analyzer）或扫描器（scanner）将输入的源程序分解成单词（token），见图1.1。图中左边一栏是单词编码（token code），该编码是一个小的整数，右边一栏是附加值，附加值也可以为空。例如，关键字int的附加值是inttype的值，代表整数类型。单个字符构成的单词的编码就是该字符的ASCII码值，EOI表示输入结束。词法分析器输出单词和单词的定义点位置，并处理#指令，编译器的其他部分无须再处理这些指令。lcc的词法分析器将在第6章中介绍。

编译器的下一个阶段是根据C语言的语法规则对单词串进行分析（parse），同时也分析程序的语义（semantic）正确性。例如，检查加法运算中操作数的类型是否合法，以便进行隐式的类型转换。在上面例子的加法运算中，f是float类型，0.5是double类型，这是合法的组合，所得结果为double类型，由于返回类型是int，求出的和被隐式转换成整数。

示例源程序经过分析阶段后，形成两棵抽象语法树（abstract syntax tree），见图1.2。树中的每个节点代表一个基本运算。第一棵树将传入的double类型的参数转换成float类型。节点（INDIR+D）从调用程序中地址为&f的单元（ADDRF+P）取出double类型的值，节点（CVD+F）将该值转换成float值。节点（ASGN+F）把float值存入被调用程序的地址为&f的单元（ADDRF+P）。

第二棵树实现了例子中惟一的一条语句，并返回一个整数（RET+I）。节点（INDIR+F）从被调用程序中地址为&f的单元（ADDRF+P）取出float值，节点（CVF+D）将该值转换成double值，节点（ADD+D）把该值加上double常量0.5（CNST+D），并将结果截尾成整数（CVD+I）。