

国外著名高等院校
信息科学与技术优秀教材



编译原理 技术与工具

(第二版)

Compilers

Principles, Techniques, & Tools
Second Edition

〔美〕 Alfred V. Aho Monica S. Lam Ravi Sethi Jeffrey D. Ullman 著



(英文版)



人民邮电出版社
POSTS & TELECOM PRESS

国外著名高等院校信息科学与技术优秀教材

编译原理 技术与工具

(第二版) (英文版)

Compilers: Principles, Techniques, & Tools
Second Edition

Alfred V. Aho

Monica S. Lam

[美]

Ravi S. Sedgwick

Jeffrey D. Ullman

江苏工业学院图书馆
藏书章

图书在版编目 (CIP) 数据

编译原理、技术与工具: 第 2 版: 英文 / (美) 阿霍 (Aho, A.V.) 等著. —北京: 人民邮电出版社, 2008.2

国外著名高等院校信息科学与技术优秀教材
ISBN 978-7-115-17265-5

I. 编… II. 阿… III. 编译程序—程序设计—高等学校—教材—英文 IV. TP314

中国版本图书馆 CIP 数据核字 (2007) 第 185367 号

版 权 声 明

Original edition, entitled COMPILERS: PRINCIPLES, TECHNIQUES, AND TOOLS, 2E, 9780321486813 by AHO, ALFRED V.; LAM, MONICA S.; SETHI, RAVI; ULLMAN, JEFFREY D., published by Pearson Education, Inc, publishing as Addison-Wesley, Copyright © 2007 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD., and POSTS & TELECOMMUNICATIONS PRESS Copyright © 2008.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in People's Republic of China excluding Hong Kong, Macau and Taiwan.

仅限于中华人民共和国境内 (不包括中国香港、澳门特别行政区和中国台湾地区) 销售。

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签。无标签者不得销售。

国外著名高等院校信息科学与技术优秀教材 编译原理 技术与工具 (第二版) (英文版)

◆ 著 [美] Alfred V. Aho Monica S. Lam
Ravi Sethi Jeffrey D. Ullman

责任编辑 李 际

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号

邮编 100061 电子函件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

北京铭成印刷有限公司印刷

新华书店总店北京发行所经销

◆ 开本: 700×1000 1/16

印张: 64.5

字数: 1 240 千字

2008 年 2 月第 1 版

印数: 1—5 000 册

2008 年 2 月北京第 1 次印刷

著作权合同登记号 图字: 01-2007-5621 号

ISBN 978-7-115-17265-5/TP

定价: 79.00 元

读者服务热线: (010) 67132705 印装质量热线: (010) 67129223

反盗版热线: (010) 67171154

内容提要

作为编译器设计的教程,本书重点主要放在解决设计语言翻译器过程中普遍需要面对的一些问题上,而并不考虑源语言或者目标机器。本书共 12 章。第一章是一些关于学习动机的资料,同时也给出了一些关于计算机体系结构和程序设计语言原理的背景知识。第二章开发了一个缩微的编译器,并介绍了很多重要的概念,这些概念将在后面的各个章节中深入介绍。这个编译器本身在附录中给出。第三章讨论了词法分析、正则表达式、有穷状态自动机和词法分析器的生成工具,这些内容是各种正文处理的基础。第四章讨论了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)。第五章介绍了语法制导定义和语法制导翻译的基本思想。第六章介绍了如何使用第五章中的理论为一个典型的程序设计语言生成中间代码。第七章讨论了运行时刻环境,主要是运行时刻栈的管理和垃圾收集机制。第八章介绍了关于目标代码生成的内容,主要讨论了基本块的构造,从表达式和基本块生成代码的方法,以及寄存器分配技术。第九章介绍了代码优化技术,包括流图、数据流分析框架以及求解这些框架的迭代算法。第十章讨论了指令级优化。该章的重点是从小段指令代码中抽取并行性,并在那些可以同时做多件事情的单处理器上调度这些指令。第十一章讲的是大规模并行的检测和利用。这章的重点是数值计算代码。这些代码具有对多维数组进行遍历的紧致循环。第十二章介绍的是关于过程间分析技术的内容,讨论了指针分析、别名和数据流分析。这些分析中都考虑了到达代码中某个给定点时的过程调用序列。

本书可作为高校计算机专业本科和研究生编译原理的教科书,也可供从事计算机软件开发的人员参考。

前言

从本书的 1986 版出版到现在,编译器设计的世界已经发生了很大的改变。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。可能最有意思的事情是,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,这些技术被有些工具用于寻找软件中的错误,以及最重要的是,寻找现有代码中的安全漏洞。并且,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,本书的先前版本所体现的我们的价值观一直没有改变。我们认识到只有很少的读者将会去构建甚至维护一个主程序语言设计的编译器。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。因此我们的重点是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

要学完本书的全部或大部分内容至少需要两个季度甚至两个学期。通常会在在一门本科课程中讲授本书的前半部分内容;而本书的后半部分——强调代码优化——会在研究生层面或小范围的另一门课程中讲授。下面是各个章节的概要介绍。

第一章是一些关于学习动机的资料,同时也给出了一些关于计算机体系结构和程序设计语言原理的背景知识。

第二章开发了一个缩微的编译器,并介绍了很多重要的概念。这些概念将在后面的各个章节中深入介绍。这个编译器本身在附录中给出。

第三章讨论了词法分析、正则表达式、有穷状态自动机和词法分析器的生成工具。这些内容是各种正文处理的基础。

第四章讨论了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)。

第五章介绍了语法制导定义和语法制导翻译的基本思想。

第六章使用第五章中的理论,显示了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。

第七章讨论了运行时刻环境,主要是运行时刻栈的管理和垃圾收集机制。

第八章介绍关于目标代码生成的内容。它讨论了基本块的构造,从表达式和基本块生成代码的方法,以及寄存器分配技术。

第九章介绍了代码优化技术,包括流程图、数据流分析框架以及求解这些框架的迭代算法。

第十章讨论了指令级优化。本章的重点是从小段指令代码中抽取并行性，并在那些可以同时做多件事情的单处理器上调度这些指令。

第十一章讲的是大规模并行的检测和利用。这里的重点是数值计算代码。这些代码具有对多维数组进行遍历的紧致循环。

第十二章介绍关于过程间分析技术的内容。它讨论了指针分析、别名和数据流分析。这些分析中都考虑了到达代码中某个给定点时的过程调用序列。

一些使用本书内容的课程已经在哥伦比亚大学、哈佛大学、斯坦福讲授。在哥伦比亚，一门关于程序设计语言和翻译器的课程使用了本书前八章的内容。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，他们还使用了第二章和第五章中讨论的语法制导翻译技术。后续的研究生课程的重点是本书第九章到第十二章的内容，着重强调针对当代计算机的代码生成和优化。这些计算机包括网络处理器和多处理器体系结构。

在斯坦福，一门一个季度的入门课程大致涵盖了第一章到第八章的内容，同时还包括了第九章中全局代码优化的介绍。第二门编译器课程包括了第九章到第十二章，另外还包括了第七章中有关垃圾收集的深入内容。学生使用一个本校开发的、基于 Java 的系统 Joeq 来实现数据流分析算法。

课程的预备知识

读者应该拥有一些“计算机科学的综合知识”，至少学过两门关于编程的课程，以及关于数据结构和离散数学的课程。有关多个不同程序设计语言的知识对本课程的学习会有所帮助。

练习

本书包含了广泛的练习，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。最难的练习有两个感叹号。

万维网上的支持

本书的主页是：

dragonbook.stanford.edu

你将在这里找到我们所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。我们也计划贴出由一些重要编译器的作者写的关于这些编译器的描述。

致谢

本书英文版封面由 Strange Tonic Productions 的 S.D.Ullman 设计。

Jon Bentley 针对这本书的早期草稿中的多章内容向我们提出了广泛深入的评审意见。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、Vibbav Garg、Kim Hazelwood、Gaurav Kc、Wei Li、Mike Smith、Art Staminess、Krysta Syore、Olivier Tardieu 和 Jia Zeng。我们衷心感谢这些人的帮助。当然,书中的残留错误是因为我们自身的原因。

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 年的时间里传授给她的编译器相关知识: 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

Table of Contents

1	Introduction	1
1.1	Language Processors	1
1.1.1	Exercises for Section 1.1	3
1.2	The Structure of a Compiler	4
1.2.1	Lexical Analysis	5
1.2.2	Syntax Analysis	8
1.2.3	Semantic Analysis	8
1.2.4	Intermediate Code Generation	9
1.2.5	Code Optimization	10
1.2.6	Code Generation	10
1.2.7	Symbol-Table Management	11
1.2.8	The Grouping of Phases into Passes	11
1.2.9	Compiler-Construction Tools	12
1.3	The Evolution of Programming Languages	12
1.3.1	The Move to Higher-level Languages	13
1.3.2	Impacts on Compilers	14
1.3.3	Exercises for Section 1.3	14
1.4	The Science of Building a Compiler	15
1.4.1	Modeling in Compiler Design and Implementation	15
1.4.2	The Science of Code Optimization	15
1.5	Applications of Compiler Technology	17
1.5.1	Implementation of High-Level Programming Languages	17
1.5.2	Optimizations for Computer Architectures	19
1.5.3	Design of New Computer Architectures	21
1.5.4	Program Translations	22
1.5.5	Software Productivity Tools	23
1.6	Programming Language Basics	25
1.6.1	The Static/Dynamic Distinction	25
1.6.2	Environments and States	26
1.6.3	Static Scope and Block Structure	28
1.6.4	Explicit Access Control	31
1.6.5	Dynamic Scope	31
1.6.6	Parameter Passing Mechanisms	33

1.6.7	Aliasing	35
1.6.8	Exercises for Section 1.6	35
1.7	Summary of Chapter 1	36
1.8	References for Chapter 1	38
2	A Simple Syntax-Directed Translator	39
2.1	Introduction	40
2.2	Syntax Definition	42
2.2.1	Definition of Grammars	42
2.2.2	Derivations	44
2.2.3	Parse Trees	45
2.2.4	Ambiguity	47
2.2.5	Associativity of Operators	48
2.2.6	Precedence of Operators	48
2.2.7	Exercises for Section 2.2	51
2.3	Syntax-Directed Translation	52
2.3.1	Postfix Notation	53
2.3.2	Synthesized Attributes	54
2.3.3	Simple Syntax-Directed Definitions	56
2.3.4	Tree Traversals	56
2.3.5	Translation Schemes	57
2.3.6	Exercises for Section 2.3	60
2.4	Parsing	60
2.4.1	Top-Down Parsing	61
2.4.2	Predictive Parsing	64
2.4.3	When to Use ϵ -Productions	65
2.4.4	Designing a Predictive Parser	66
2.4.5	Left Recursion	67
2.4.6	Exercises for Section 2.4	68
2.5	A Translator for Simple Expressions	68
2.5.1	Abstract and Concrete Syntax	69
2.5.2	Adapting the Translation Scheme	70
2.5.3	Procedures for the Nonterminals	72
2.5.4	Simplifying the Translator	73
2.5.5	The Complete Program	74
2.6	Lexical Analysis	76
2.6.1	Removal of White Space and Comments	77
2.6.2	Reading Ahead	78
2.6.3	Constants	78
2.6.4	Recognizing Keywords and Identifiers	79
2.6.5	A Lexical Analyzer	81
2.6.6	Exercises for Section 2.6	84
2.7	Symbol Tables	85
2.7.1	Symbol Table Per Scope	86
2.7.2	The Use of Symbol Tables	89

2.8	Intermediate Code Generation	91
2.8.1	Two Kinds of Intermediate Representations	91
2.8.2	Construction of Syntax Trees	92
2.8.3	Static Checking	97
2.8.4	Three-Address Code	99
2.8.5	Exercises for Section 2.8	105
2.9	Summary of Chapter 2	105
3	Lexical Analysis	109
3.1	The Role of the Lexical Analyzer	109
3.1.1	Lexical Analysis Versus Parsing	110
3.1.2	Tokens, Patterns, and Lexemes	111
3.1.3	Attributes for Tokens	112
3.1.4	Lexical Errors	113
3.1.5	Exercises for Section 3.1	114
3.2	Input Buffering	115
3.2.1	Buffer Pairs	115
3.2.2	Sentinels	116
3.3	Specification of Tokens	116
3.3.1	Strings and Languages	117
3.3.2	Operations on Languages	119
3.3.3	Regular Expressions	120
3.3.4	Regular Definitions	123
3.3.5	Extensions of Regular Expressions	124
3.3.6	Exercises for Section 3.3	125
3.4	Recognition of Tokens	128
3.4.1	Transition Diagrams	130
3.4.2	Recognition of Reserved Words and Identifiers	132
3.4.3	Completion of the Running Example	133
3.4.4	Architecture of a Transition-Diagram-Based Lexical Analyzer	134
3.4.5	Exercises for Section 3.4	136
3.5	The Lexical-Analyzer Generator Lex	140
3.5.1	Use of Lex	140
3.5.2	Structure of Lex Programs	141
3.5.3	Conflict Resolution in Lex	144
3.5.4	The Lookahead Operator	144
3.5.5	Exercises for Section 3.5	146
3.6	Finite Automata	147
3.6.1	Nondeterministic Finite Automata	147
3.6.2	Transition Tables	148
3.6.3	Acceptance of Input Strings by Automata	149
3.6.4	Deterministic Finite Automata	149
3.6.5	Exercises for Section 3.6	151
3.7	From Regular Expressions to Automata	152

3.7.1	Conversion of an NFA to a DFA	152
3.7.2	Simulation of an NFA	156
3.7.3	Efficiency of NFA Simulation	157
3.7.4	Construction of an NFA from a Regular Expression	159
3.7.5	Efficiency of String-Processing Algorithms	163
3.7.6	Exercises for Section 3.7	166
3.8	Design of a Lexical-Analyzer Generator	166
3.8.1	The Structure of the Generated Analyzer	167
3.8.2	Pattern Matching Based on NFA's	168
3.8.3	DFA's for Lexical Analyzers	170
3.8.4	Implementing the Lookahead Operator	171
3.8.5	Exercises for Section 3.8	172
3.9	Optimization of DFA-Based Pattern Matchers	173
3.9.1	Important States of an NFA	173
3.9.2	Functions Computed From the Syntax Tree	175
3.9.3	Computing <i>nullable</i> , <i>firstpos</i> , and <i>lastpos</i>	176
3.9.4	Computing <i>followpos</i>	177
3.9.5	Converting a Regular Expression Directly to a DFA	179
3.9.6	Minimizing the Number of States of a DFA	180
3.9.7	State Minimization in Lexical Analyzers	184
3.9.8	Trading Time for Space in DFA Simulation	185
3.9.9	Exercises for Section 3.9	186
3.10	Summary of Chapter 3	187
3.11	References for Chapter 3	189
4	Syntax Analysis	191
4.1	Introduction	192
4.1.1	The Role of the Parser	192
4.1.2	Representative Grammars	193
4.1.3	Syntax Error Handling	194
4.1.4	Error-Recovery Strategies	195
4.2	Context-Free Grammars	197
4.2.1	The Formal Definition of a Context-Free Grammar	197
4.2.2	Notational Conventions	198
4.2.3	Derivations	199
4.2.4	Parse Trees and Derivations	201
4.2.5	Ambiguity	203
4.2.6	Verifying the Language Generated by a Grammar	204
4.2.7	Context-Free Grammars Versus Regular Expressions	205
4.2.8	Exercises for Section 4.2	206
4.3	Writing a Grammar	209
4.3.1	Lexical Versus Syntactic Analysis	209
4.3.2	Eliminating Ambiguity	210
4.3.3	Elimination of Left Recursion	212
4.3.4	Left Factoring	214

4.3.5	Non-Context-Free Language Constructs	215
4.3.6	Exercises for Section 4.3	216
4.4	Top-Down Parsing	217
4.4.1	Recursive-Descent Parsing	219
4.4.2	FIRST and FOLLOW	220
4.4.3	LL(1) Grammars	222
4.4.4	Nonrecursive Predictive Parsing	226
4.4.5	Error Recovery in Predictive Parsing	228
4.4.6	Exercises for Section 4.4	231
4.5	Bottom-Up Parsing	233
4.5.1	Reductions	234
4.5.2	Handle Pruning	235
4.5.3	Shift-Reduce Parsing	236
4.5.4	Conflicts During Shift-Reduce Parsing	238
4.5.5	Exercises for Section 4.5	240
4.6	Introduction to LR Parsing: Simple LR	241
4.6.1	Why LR Parsers?	241
4.6.2	Items and the LR(0) Automaton	242
4.6.3	The LR-Parsing Algorithm	248
4.6.4	Constructing SLR-Parsing Tables	252
4.6.5	Viable Prefixes	256
4.6.6	Exercises for Section 4.6	257
4.7	More Powerful LR Parsers	259
4.7.1	Canonical LR(1) Items	260
4.7.2	Constructing LR(1) Sets of Items	261
4.7.3	Canonical LR(1) Parsing Tables	265
4.7.4	Constructing LALR Parsing Tables	266
4.7.5	Efficient Construction of LALR Parsing Tables	270
4.7.6	Compaction of LR Parsing Tables	275
4.7.7	Exercises for Section 4.7	277
4.8	Using Ambiguous Grammars	278
4.8.1	Precedence and Associativity to Resolve Conflicts	279
4.8.2	The “Dangling-Else” Ambiguity	281
4.8.3	Error Recovery in LR Parsing	283
4.8.4	Exercises for Section 4.8	285
4.9	Parser Generators	287
4.9.1	The Parser Generator Yacc	287
4.9.2	Using Yacc with Ambiguous Grammars	291
4.9.3	Creating Yacc Lexical Analyzers with Lex	294
4.9.4	Error Recovery in Yacc	295
4.9.5	Exercises for Section 4.9	297
4.10	Summary of Chapter 4	297
4.11	References for Chapter 4	300

5	Syntax-Directed Translation	303
5.1	Syntax-Directed Definitions	304
5.1.1	Inherited and Synthesized Attributes	304
5.1.2	Evaluating an SDD at the Nodes of a Parse Tree	306
5.1.3	Exercises for Section 5.1	309
5.2	Evaluation Orders for SDD's	310
5.2.1	Dependency Graphs	310
5.2.2	Ordering the Evaluation of Attributes	312
5.2.3	S-Attributed Definitions	312
5.2.4	L-Attributed Definitions	313
5.2.5	Semantic Rules with Controlled Side Effects	314
5.2.6	Exercises for Section 5.2	317
5.3	Applications of Syntax-Directed Translation	318
5.3.1	Construction of Syntax Trees	318
5.3.2	The Structure of a Type	321
5.3.3	Exercises for Section 5.3	323
5.4	Syntax-Directed Translation Schemes	324
5.4.1	Postfix Translation Schemes	324
5.4.2	Parser-Stack Implementation of Postfix SDT's	325
5.4.3	SDT's With Actions Inside Productions	327
5.4.4	Eliminating Left Recursion From SDT's	328
5.4.5	SDT's for L-Attributed Definitions	331
5.4.6	Exercises for Section 5.4	336
5.5	Implementing L-Attributed SDD's	337
5.5.1	Translation During Recursive-Descent Parsing	338
5.5.2	On-The-Fly Code Generation	340
5.5.3	L-Attributed SDD's and LL Parsing	343
5.5.4	Bottom-Up Parsing of L-Attributed SDD's	348
5.5.5	Exercises for Section 5.5	352
5.6	Summary of Chapter 5	353
5.7	References for Chapter 5	354
6	Intermediate-Code Generation	357
6.1	Variants of Syntax Trees	358
6.1.1	Directed Acyclic Graphs for Expressions	359
6.1.2	The Value-Number Method for Constructing DAG's	360
6.1.3	Exercises for Section 6.1	362
6.2	Three-Address Code	363
6.2.1	Addresses and Instructions	364
6.2.2	Quadruples	366
6.2.3	Triples	367
6.2.4	Static Single-Assignment Form	369
6.2.5	Exercises for Section 6.2	370
6.3	Types and Declarations	370
6.3.1	Type Expressions	371

6.3.2	Type Equivalence	372
6.3.3	Declarations	373
6.3.4	Storage Layout for Local Names	373
6.3.5	Sequences of Declarations	376
6.3.6	Fields in Records and Classes	376
6.3.7	Exercises for Section 6.3	378
6.4	Translation of Expressions	378
6.4.1	Operations Within Expressions	378
6.4.2	Incremental Translation	380
6.4.3	Addressing Array Elements	381
6.4.4	Translation of Array References	383
6.4.5	Exercises for Section 6.4	384
6.5	Type Checking	386
6.5.1	Rules for Type Checking	387
6.5.2	Type Conversions	388
6.5.3	Overloading of Functions and Operators	390
6.5.4	Type Inference and Polymorphic Functions	391
6.5.5	An Algorithm for Unification	395
6.5.6	Exercises for Section 6.5	398
6.6	Control Flow	399
6.6.1	Boolean Expressions	399
6.6.2	Short-Circuit Code	400
6.6.3	Flow-of-Control Statements	401
6.6.4	Control-Flow Translation of Boolean Expressions	403
6.6.5	Avoiding Redundant Gotos	405
6.6.6	Boolean Values and Jumping Code	408
6.6.7	Exercises for Section 6.6	408
6.7	Backpatching	410
6.7.1	One-Pass Code Generation Using Backpatching	410
6.7.2	Backpatching for Boolean Expressions	411
6.7.3	Flow-of-Control Statements	413
6.7.4	Break-, Continue-, and Goto-Statements	416
6.7.5	Exercises for Section 6.7	417
6.8	Switch-Statements	418
6.8.1	Translation of Switch-Statements	419
6.8.2	Syntax-Directed Translation of Switch-Statements	420
6.8.3	Exercises for Section 6.8	421
6.9	Intermediate Code for Procedures	422
6.10	Summary of Chapter 6	424
6.11	References for Chapter 6	425

7	Run-Time Environments	427
7.1	Storage Organization	427
7.1.1	Static Versus Dynamic Storage Allocation	429
7.2	Stack Allocation of Space	430
7.2.1	Activation Trees	430
7.2.2	Activation Records	433
7.2.3	Calling Sequences	436
7.2.4	Variable-Length Data on the Stack	438
7.2.5	Exercises for Section 7.2	440
7.3	Access to Nonlocal Data on the Stack	441
7.3.1	Data Access Without Nested Procedures	442
7.3.2	Issues With Nested Procedures	442
7.3.3	A Language With Nested Procedure Declarations	443
7.3.4	Nesting Depth	443
7.3.5	Access Links	445
7.3.6	Manipulating Access Links	447
7.3.7	Access Links for Procedure Parameters	448
7.3.8	Displays	449
7.3.9	Exercises for Section 7.3	451
7.4	Heap Management	452
7.4.1	The Memory Manager	453
7.4.2	The Memory Hierarchy of a Computer	454
7.4.3	Locality in Programs	455
7.4.4	Reducing Fragmentation	457
7.4.5	Manual Deallocation Requests	460
7.4.6	Exercises for Section 7.4	463
7.5	Introduction to Garbage Collection	463
7.5.1	Design Goals for Garbage Collectors	464
7.5.2	Reachability	466
7.5.3	Reference Counting Garbage Collectors	468
7.5.4	Exercises for Section 7.5	470
7.6	Introduction to Trace-Based Collection	470
7.6.1	A Basic Mark-and-Sweep Collector	471
7.6.2	Basic Abstraction	473
7.6.3	Optimizing Mark-and-Sweep	475
7.6.4	Mark-and-Compact Garbage Collectors	476
7.6.5	Copying collectors	478
7.6.6	Comparing Costs	482
7.6.7	Exercises for Section 7.6	482
7.7	Short-Pause Garbage Collection	483
7.7.1	Incremental Garbage Collection	483
7.7.2	Incremental Reachability Analysis	485
7.7.3	Partial-Collection Basics	487
7.7.4	Generational Garbage Collection	488
7.7.5	The Train Algorithm	490

7.7.6	Exercises for Section 7.7	493
7.8	Advanced Topics in Garbage Collection	494
7.8.1	Parallel and Concurrent Garbage Collection	495
7.8.2	Partial Object Relocation	497
7.8.3	Conservative Collection for Unsafe Languages	498
7.8.4	Weak References	498
7.8.5	Exercises for Section 7.8	499
7.9	Summary of Chapter 7	500
7.10	References for Chapter 7	502
8	Code Generation	505
8.1	Issues in the Design of a Code Generator	506
8.1.1	Input to the Code Generator	507
8.1.2	The Target Program	507
8.1.3	Instruction Selection	508
8.1.4	Register Allocation	510
8.1.5	Evaluation Order	511
8.2	The Target Language	512
8.2.1	A Simple Target Machine Model	512
8.2.2	Program and Instruction Costs	515
8.2.3	Exercises for Section 8.2	516
8.3	Addresses in the Target Code	518
8.3.1	Static Allocation	518
8.3.2	Stack Allocation	520
8.3.3	Run-Time Addresses for Names	522
8.3.4	Exercises for Section 8.3	524
8.4	Basic Blocks and Flow Graphs	525
8.4.1	Basic Blocks	526
8.4.2	Next-Use Information	528
8.4.3	Flow Graphs	529
8.4.4	Representation of Flow Graphs	530
8.4.5	Loops	531
8.4.6	Exercises for Section 8.4	531
8.5	Optimization of Basic Blocks	533
8.5.1	The DAG Representation of Basic Blocks	533
8.5.2	Finding Local Common Subexpressions	534
8.5.3	Dead Code Elimination	535
8.5.4	The Use of Algebraic Identities	536
8.5.5	Representation of Array References	537
8.5.6	Pointer Assignments and Procedure Calls	539
8.5.7	Reassembling Basic Blocks From DAG's	539
8.5.8	Exercises for Section 8.5	541
8.6	A Simple Code Generator	542
8.6.1	Register and Address Descriptors	543
8.6.2	The Code-Generation Algorithm	544

8.6.3	Design of the Function <i>getReg</i>	547
8.6.4	Exercises for Section 8.6	548
8.7	Peephole Optimization	549
8.7.1	Eliminating Redundant Loads and Stores	550
8.7.2	Eliminating Unreachable Code	550
8.7.3	Flow-of-Control Optimizations	551
8.7.4	Algebraic Simplification and Reduction in Strength	552
8.7.5	Use of Machine Idioms	552
8.7.6	Exercises for Section 8.7	553
8.8	Register Allocation and Assignment	553
8.8.1	Global Register Allocation	553
8.8.2	Usage Counts	554
8.8.3	Register Assignment for Outer Loops	556
8.8.4	Register Allocation by Graph Coloring	556
8.8.5	Exercises for Section 8.8	557
8.9	Instruction Selection by Tree Rewriting	558
8.9.1	Tree-Translation Schemes	558
8.9.2	Code Generation by Tiling an Input Tree	560
8.9.3	Pattern Matching by Parsing	563
8.9.4	Routines for Semantic Checking	565
8.9.5	General Tree Matching	565
8.9.6	Exercises for Section 8.9	567
8.10	Optimal Code Generation for Expressions	567
8.10.1	Ershov Numbers	567
8.10.2	Generating Code From Labeled Expression Trees	568
8.10.3	Evaluating Expressions with an Insufficient Supply of Reg- isters	570
8.10.4	Exercises for Section 8.10	572
8.11	Dynamic Programming Code-Generation	573
8.11.1	Contiguous Evaluation	574
8.11.2	The Dynamic Programming Algorithm	575
8.11.3	Exercises for Section 8.11	577
8.12	Summary of Chapter 8	578
8.13	References for Chapter 8	579
9	Machine-Independent Optimizations	583
9.1	The Principal Sources of Optimization	584
9.1.1	Causes of Redundancy	584
9.1.2	A Running Example: Quicksort	585
9.1.3	Semantics-Preserving Transformations	586
9.1.4	Global Common Subexpressions	588
9.1.5	Copy Propagation	590
9.1.6	Dead-Code Elimination	591
9.1.7	Code Motion	592
9.1.8	Induction Variables and Reduction in Strength	592