
Modern Compiler Implementation in Java Second Edition

This textbook describes all phases of a compiler: lexical analysis, parsing, abstract syntax, semantic actions, intermediate representations, instruction selection via tree matching, dataflow analysis, graph-coloring register allocation, and runtime systems. It includes good coverage of current techniques in code generation and register allocation, as well as the compilation of functional and object-oriented languages, which is missing from most books. The most accepted and successful techniques are described concisely, rather than as an exhaustive catalog of every possible variant. Detailed descriptions of the interfaces between modules of a compiler are illustrated with actual Java classes.

The first part of the book, Fundamentals of Compilation, is suitable for a one-semester first course in compiler design. The second part, Advanced Topics, which includes the compilation of object-oriented and functional languages, garbage collection, loop optimization, SSA form, instruction scheduling, and optimization for cache-memory hierarchies, can be used for a second-semester or graduate course.

This new edition has been rewritten extensively to include more discussion of Java and object-oriented programming concepts, such as visitor patterns. A unique feature is the newly redesigned compiler project in Java for a subset of Java itself. The project includes both front-end and back-end phases, so that students can build a complete working compiler in one semester.

Andrew W. Appel is Professor of Computer Science at Princeton University. He has done research and published papers on compilers, functional programming languages, runtime systems and garbage collection, type systems, and computer security; he is also author of the book *Compiling with Continuations*. He is a designer and founder of the Standard ML of New Jersey project. In 1998, Appel was elected a Fellow of the Association for Computing Machinery for “significant research contributions in the area of programming languages and compilers” and for his work as editor-in-chief (1993–97) of the *ACM Transactions on Programming Languages and Systems*, the leading journal in the field of compilers and programming languages.

Jens Palsberg is Associate Professor of Computer Science at Purdue University. His research interests are programming languages, compilers, software engineering, and information security. He has authored more than 50 technical papers in these areas and a book with Michael Schwartzbach, *Object-oriented Type Systems*. In 1998, he received the National Science Foundation Faculty Early Career Development Award, and in 1999, the Purdue University Faculty Scholar award.

Modern Compiler Implementation in Java

Second Edition

ANDREW W. APPEL

Princeton University

with JENS PALSBERG

Purdue University



CAMBRIDGE
UNIVERSITY PRESS

Cambridge University Press
 052182060X - Modern Compiler Implementation in Java, Second Edition
 Andrew W. Appel
 Frontmatter
[More information](#)

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
 The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
 The Edinburgh Building, Cambridge CB2 2RU, UK
 40 West 20th Street, New York, NY 10011-4211, USA
 477 Williamstown Road, Port Melbourne, VIC 3207, Australia
 Ruiz de Alarcón 13, 28014 Madrid, Spain
 Dock House, The Waterfront, Cape Town 8001, South Africa
<http://www.cambridge.org>

© Cambridge University Press 2002

This book is in copyright. Subject to statutory exception
 and to the provisions of relevant collective licensing agreements,
 no reproduction of any part may take place without
 the written permission of Cambridge University Press.

First edition published 1998
 Second edition published 2002

Printed in the United States of America

Typefaces Times, Courier, and Optima *System* L^AT_EX [AU]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication data

Appel, Andrew W., 1960–
 Modern compiler implementation in Java / Andrew W. Appel with Jens Palsberg. —
 [2nd ed.]
 p. cm.
 Includes bibliographical references and index.
 ISBN 0-521-82060-X
 1. Java (Computer program language) 2. Compilers (Computer programs) I. Palsberg,
 Jens. II. Title.
 QA76.73.J38 A65 2002
 005.4'53—dc21

2002073453

ISBN 0 521 58274 1 Modern Compiler Implementation in ML (first edition, hardback)
 ISBN 0 521 82060 X Modern Compiler Implementation in Java (hardback)

Contents

Preface ix

Part I Fundamentals of Compilation

1 Introduction	3
1.1 Modules and interfaces	4
1.2 Tools and software	5
1.3 Data structures for tree languages	7
2 Lexical Analysis	16
2.1 Lexical tokens	17
2.2 Regular expressions	18
2.3 Finite automata	21
2.4 Nondeterministic finite automata	24
2.5 Lexical-analyzer generators	30
3 Parsing	38
3.1 Context-free grammars	40
3.2 Predictive parsing	45
3.3 LR parsing	55
3.4 Using parser generators	68
3.5 Error recovery	76
4 Abstract Syntax	86
4.1 Semantic actions	86
4.2 Abstract parse trees	89
4.3 Visitors	93
5 Semantic Analysis	103
5.1 Symbol tables	103

 CONTENTS

5.2	Type-checking MiniJava	111
6	Activation Records	116
6.1	Stack frames	118
6.2	Frames in the MiniJava compiler	126
7	Translation to Intermediate Code	136
7.1	Intermediate representation trees	137
7.2	Translation into trees	140
7.3	Declarations	155
8	Basic Blocks and Traces	162
8.1	Canonical trees	163
8.2	Taming conditional branches	169
9	Instruction Selection	176
9.1	Algorithms for instruction selection	179
9.2	CISC machines	187
9.3	Instruction selection for the MiniJava compiler	190
10	Liveness Analysis	203
10.1	Solution of dataflow equations	205
10.2	Liveness in the MiniJava compiler	214
11	Register Allocation	219
11.1	Coloring by simplification	220
11.2	Coalescing	223
11.3	Precolored nodes	227
11.4	Graph-coloring implementation	232
11.5	Register allocation for trees	241
12	Putting It All Together	249
 Part II Advanced Topics		
13	Garbage Collection	257
13.1	Mark-and-sweep collection	257
13.2	Reference counts	262
13.3	Copying collection	264

CONTENTS

13.4	Generational collection	269
13.5	Incremental collection	272
13.6	Baker's algorithm	274
13.7	Interface to the compiler	275
14	Object-Oriented Languages	283
14.1	Class extension	283
14.2	Single inheritance of data fields	284
14.3	Multiple inheritance	286
14.4	Testing class membership	289
14.5	Private fields and methods	292
14.6	Classless languages	293
14.7	Optimizing object-oriented programs	293
15	Functional Programming Languages	298
15.1	A simple functional language	299
15.2	Closures	301
15.3	Immutable variables	302
15.4	Inline expansion	308
15.5	Closure conversion	316
15.6	Efficient tail recursion	319
15.7	Lazy evaluation	321
16	Polymorphic Types	335
16.1	Parametric polymorphism	336
16.2	Polymorphic type-checking	339
16.3	Translation of polymorphic programs	344
16.4	Resolution of static overloading	347
17	Dataflow Analysis	350
17.1	Intermediate representation for flow analysis	351
17.2	Various dataflow analyses	354
17.3	Transformations using dataflow analysis	359
17.4	Speeding up dataflow analysis	360
17.5	Alias analysis	369
18	Loop Optimizations	376
18.1	Dominators	379
18.2	Loop-invariant computations	384

 CONTENTS

18.3	Induction variables	385
18.4	Array-bounds checks	391
18.5	Loop unrolling	395
19	Static Single-Assignment Form	399
19.1	Converting to SSA form	402
19.2	Efficient computation of the dominator tree	410
19.3	Optimization algorithms using SSA	417
19.4	Arrays, pointers, and memory	423
19.5	The control-dependence graph	425
19.6	Converting back from SSA form	428
19.7	A functional intermediate form	430
20	Pipelining and Scheduling	440
20.1	Loop scheduling without resource bounds	444
20.2	Resource-bounded loop pipelining	448
20.3	Branch prediction	456
21	The Memory Hierarchy	464
21.1	Cache organization	465
21.2	Cache-block alignment	468
21.3	Prefetching	470
21.4	Loop interchange	476
21.5	Blocking	477
21.6	Garbage collection and the memory hierarchy	480
	Appendix: MiniJava Language Reference Manual	484
A.1	Lexical Issues	484
A.2	Grammar	484
A.3	Sample Program	486
	<i>Bibliography</i>	487
	<i>Index</i>	495

Preface

This book is intended as a textbook for a one- or two-semester course in compilers. Students will see the theory behind different components of a compiler, the programming techniques used to put the theory into practice, and the interfaces used to modularize the compiler. To make the interfaces and programming examples clear and concrete, we have written them in Java. Another edition of this book is available that uses the ML language.

Implementation project. The “student project compiler” that we have outlined is reasonably simple, but is organized to demonstrate some important techniques that are now in common use: abstract syntax trees to avoid tangling syntax and semantics, separation of instruction selection from register allocation, copy propagation to give flexibility to earlier phases of the compiler, and containment of target-machine dependencies. Unlike many “student compilers” found in other textbooks, this one has a simple but sophisticated back end, allowing good register allocation to be done after instruction selection.

This second edition of the book has a redesigned project compiler: It uses a subset of Java, called MiniJava, as the source language for the compiler project, it explains the use of the parser generators JavaCC and SableCC, and it promotes programming with the Visitor pattern. Students using this edition can implement a compiler for a language they’re familiar with, using standard tools, in a more object-oriented style.

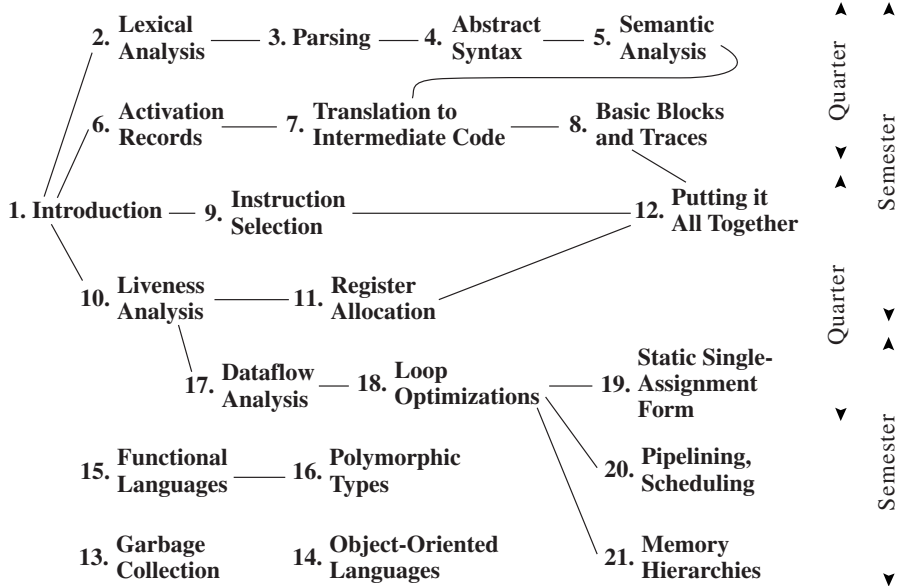
Each chapter in Part I has a programming exercise corresponding to one module of a compiler. Software useful for the exercises can be found at <http://uk.cambridge.org/resources/052182060X> (outside North America); <http://us.cambridge.org/titles/052182060X.html> (within North America).

Exercises. Each chapter has pencil-and-paper exercises; those marked with a star are more challenging, two-star problems are difficult but solvable, and

 PREFACE

the occasional three-star exercises are not known to have a solution.

Course sequence. The figure shows how the chapters depend on each other.



- A one-semester course could cover all of Part I (Chapters 1–12), with students implementing the project compiler (perhaps working in groups); in addition, lectures could cover selected topics from Part II.
- An advanced or graduate course could cover Part II, as well as additional topics from the current literature. Many of the Part II chapters can stand independently from Part I, so that an advanced course could be taught to students who have used a different book for their first course.
- In a two-quarter sequence, the first quarter could cover Chapters 1–8, and the second quarter could cover Chapters 9–12 and some chapters from Part II.

Acknowledgments. Many people have provided constructive criticism or helped us in other ways on this book. Vidyut Samanta helped tremendously with both the text and the software for the new edition of the book. We would also like to thank Leonor Abraido-Fandino, Scott Ananian, Nils Andersen, Stephen Bailey, Joao Cangussu, Maia Ginsburg, Max Hailperin, David Hanson, Jeffrey Hsu, David MacQueen, Torben Mogensen, Doug Morgan, Robert Netzer, Elma Lee Noah, Mikael Petterson, Benjamin Pierce, Todd Proebsting, Anne Rogers, Barbara Ryder, Amr Sabry, Mooly Sagiv, Zhong Shao, Mary Lou Soffa, Andrew Tolmach, Kwangkeun Yi, and Kenneth Zadeck.