

Introduction to Compilers and Language Design

Second Edition

**Prof. Douglas Thain
University of Notre Dame**

Introduction to Compilers and Language Design

Copyright © 2023 Douglas Thain.

Paperback ISBN: 979-8-655-18026-0

Second edition.

Anyone is free to download and print the PDF edition of this book for personal use. Commercial distribution, printing, or reproduction without the author's consent is expressly prohibited. All other rights are reserved.

You can find the latest version of the PDF edition, and purchase inexpensive hardcover copies at <http://compilerbook.org>

Revision Date: August 24, 2023

For Lisa, William, Zachary, Emily, and Alia.

Contributions

I am grateful to the following people for their contributions to this book:

Andrew Litteken drafted the chapter on ARM assembly; Kevin Latimer drew the RegEx to NFA and the LR example figures; Benjamin Gunning fixed an error in LL(1) parse table construction; Tim Shaffer completed the detailed LR(1) example.

And the following people corrected typos:

Sakib Haque (27), John Westhoff (26), Emily Strout (26), Gonzalo Martinez (25), Daniel Kerrigan (24), Brian DuSell (23), Livia Johan (22), Ryan Mackey (20), TJ Dasso (18), Nedim Mininovic (15), Noah Yoshida (14), Joseph Kimlinger (12), Nolan McShea (11), Jongsuh Lee (11), Kyle Weingartner (10), Anthony Schmidt (10) Andrew Litteken (9), Thomas Cane (9), Samuel Battalio (9), Stéphane Massou (8), Luis Prieb (7), William Diederich (7), Jonathan Xu (6), Gavin Inglis (6), Kathleen Capella (6), Edward Atkinson (6), Kristin Friday (5), Tanner Juedeman (5), Thanh Son Phung (4), John Johnson (4), Luke Siela (4), Francis Schickel (4), Eamon Marmion (3), Molly Zachlin (3), David Chiang (3), Jacob Mazur (3), Spencer King (2), Yaoxian Qu (2), Maria Aranguren (2), Patrick Lacher (2), Connor Higgins (2), Tango Gu (2), Andrew Syrmakesis (2), Horst von Brand (2), John Fox (2), Jamie Zhang (2), John Sullivan (2), Benjamin Gunning (1), Charles Osborne (1), William Theisen (1), Jessica Cioffi (1), Ben Tovar (1), Ryan Michalec (1), Patrick Flynn (1), Clint Jeffery (1), Ralph Siemsen (1), John Quinn (1), Paul Brunts (1), Luke Wurl (1), Bruce Mardle (1), Dane Williams (1), Thomas Fisher (1), Alan Johnson (1), Jacob Harris (1), Jeff Clinton (1), Reto Habluetzel (1), Chris Fietkiewicz (1), Miguel Pach (1),

Please send any comments or corrections via email to Prof. Douglas Thain (dthain@nd.edu).

Contents

1	Introduction	1
1.1	What is a compiler?	1
1.2	Why should you study compilers?	2
1.3	What's the best way to learn about compilers?	2
1.4	What language should I use?	2
1.5	How is this book different from others?	3
1.6	What other books should I read?	4
2	A Quick Tour	5
2.1	The Compiler Toolchain	5
2.2	Stages Within a Compiler	6
2.3	Example Compilation	7
2.4	Exercises	10
3	Scanning	11
3.1	Kinds of Tokens	11
3.2	A Hand-Made Scanner	12
3.3	Regular Expressions	13
3.4	Finite Automata	15
3.4.1	Deterministic Finite Automata	16
3.4.2	Nondeterministic Finite Automata	17
3.5	Conversion Algorithms	19
3.5.1	Converting REs to NFAs	19
3.5.2	Converting NFAs to DFAs	22
3.5.3	Minimizing DFAs	24
3.6	Limits of Finite Automata	26
3.7	Using a Scanner Generator	27
3.8	Practical Considerations	28
3.9	Exercises	31
3.10	Further Reading	33
4	Parsing	35
4.1	Overview	35
4.2	Context Free Grammars	36

4.2.1	Deriving Sentences	37
4.2.2	Ambiguous Grammars	38
4.3	LL Grammars	40
4.3.1	Eliminating Left Recursion	41
4.3.2	Eliminating Common Left Prefixes	42
4.3.3	First and Follow Sets	43
4.3.4	Recursive Descent Parsing	45
4.3.5	Table Driven Parsing	47
4.4	LR Grammars	49
4.4.1	Shift-Reduce Parsing	50
4.4.2	The LR(0) Automaton	51
4.4.3	SLR Parsing	55
4.4.4	LR(1) Parsing	59
4.4.5	LALR Parsing	62
4.5	Grammar Classes Revisited	62
4.6	The Chomsky Hierarchy	63
4.7	Exercises	65
4.8	Further Reading	67
5	Parsing in Practice	69
5.1	The Bison Parser Generator	70
5.2	Expression Validator	73
5.3	Expression Interpreter	74
5.4	Expression Trees	75
5.5	Exercises	81
5.6	Further Reading	83
6	The Abstract Syntax Tree	85
6.1	Overview	85
6.2	Declarations	86
6.3	Statements	88
6.4	Expressions	90
6.5	Types	92
6.6	Putting it All Together	95
6.7	Building the AST	96
6.8	Exercises	98
7	Semantic Analysis	99
7.1	Overview of Type Systems	100
7.2	Designing a Type System	103
7.3	The B-Minor Type System	106
7.4	The Symbol Table	107
7.5	Name Resolution	111
7.6	Implementing Type Checking	113
7.7	Error Messages	117

7.8	Exercises	118
7.9	Further Reading	118
8	Intermediate Representations	119
8.1	Introduction	119
8.2	Abstract Syntax Tree	119
8.3	Directed Acyclic Graph	120
8.4	Control Flow Graph	125
8.5	Static Single Assignment Form	127
8.6	Linear IR	128
8.7	Stack Machine IR	129
8.8	Examples	130
8.8.1	GIMPLE - GNU Simple Representation	130
8.8.2	LLVM - Low-Level Virtual Machine	131
8.8.3	JVM - Java Virtual Machine	132
8.9	Exercises	133
8.10	Further Reading	134
9	Memory Organization	135
9.1	Introduction	135
9.2	Logical Segmentation	135
9.3	Heap Management	138
9.4	Stack Management	140
9.4.1	Stack Calling Convention	141
9.4.2	Register Calling Convention	142
9.5	Locating Data	143
9.6	Program Loading	146
9.7	Further Reading	148
10	Assembly Language	149
10.1	Introduction	149
10.2	Open Source Assembler Tools	150
10.3	X86 Assembly Language	152
10.3.1	Registers and Data Types	152
10.3.2	Addressing Modes	154
10.3.3	Basic Arithmetic	156
10.3.4	Comparisons and Jumps	158
10.3.5	The Stack	159
10.3.6	Calling a Function	160
10.3.7	Defining a Leaf Function	162
10.3.8	Defining a Complex Function	163
10.4	ARM Assembly	167
10.4.1	Registers and Data Types	167
10.4.2	Addressing Modes	168
10.4.3	Basic Arithmetic	170

10.4.4	Comparisons and Branches	171
10.4.5	The Stack	173
10.4.6	Calling a Function	174
10.4.7	Defining a Leaf Function	175
10.4.8	Defining a Complex Function	176
10.4.9	64-bit Differences	179
10.5	Further Reading	180
11	Code Generation	181
11.1	Introduction	181
11.2	Supporting Functions	181
11.3	Generating Expressions	183
11.4	Generating Statements	188
11.5	Conditional Expressions	192
11.6	Generating Declarations	193
11.7	Exercises	194
12	Optimization	195
12.1	Overview	195
12.2	Optimization in Perspective	196
12.3	High-Level Optimizations	197
12.3.1	Constant Folding	197
12.3.2	Strength Reduction	199
12.3.3	Loop Unrolling	199
12.3.4	Code Hoisting	200
12.3.5	Function Inlining	201
12.3.6	Dead Code Detection and Elimination	202
12.4	Low-Level Optimizations	204
12.4.1	Peephole Optimizations	204
12.4.2	Instruction Selection	204
12.5	Register Allocation	207
12.5.1	Safety of Register Allocation	208
12.5.2	Priority of Register Allocation	208
12.5.3	Conflicts Between Variables	209
12.5.4	Global Register Allocation	210
12.6	Optimization Pitfalls	211
12.7	Optimization Interactions	212
12.8	Exercises	214
12.9	Further Reading	215
A	Sample Course Project	217
A.1	Scanner Assignment	217
A.2	Parser Assignment	217
A.3	Pretty-Printer Assignment	218
A.4	Typechecker Assignment	218

A.5	Optional: Intermediate Representation	218
A.6	Code Generator Assignment	218
A.7	Optional: Extend the Language	219
B	The B-Minor Language	221
B.1	Overview	221
B.2	Tokens	222
B.3	Types	222
B.4	Expressions	223
B.5	Declarations and Statements	224
B.6	Functions	224
B.7	Optional Elements	225
C	Coding Conventions	227
	Index	229

List of Figures

2.1	A Typical Compiler Toolchain	5
2.2	The Stages of a Unix Compiler	6
2.3	Example AST	9
2.4	Example Intermediate Representation	9
2.5	Example Assembly Code	10
3.1	A Simple Hand Made Scanner	12
3.2	Relationship Between REs, NFAs, and DFAs	19
3.3	Subset Construction Algorithm	22
3.4	Converting an NFA to a DFA via Subset Construction	23
3.5	Hopcroft's DFA Minimization Algorithm	24
3.6	Structure of a Flex File	27
3.7	Example Flex Specification	29
3.8	Example Main Program	29
3.9	Example Token Enumeration	30
3.10	Build Procedure for a Flex Program	30
4.1	Two Derivations of the Same Sentence	38
4.2	A Recursive-Descent Parser	46
4.3	LR(0) Automaton for Grammar G_{10}	53
4.4	SLR Parse Table for Grammar G_{10}	56
4.5	Part of LR(0) Automaton for Grammar G_{11}	58
4.6	LR(1) Automaton for Grammar G_{10}	61
4.7	The Chomsky Hierarchy	64
5.1	Bison Specification for Expression Validator	71
5.2	Main Program for Expression Validator	72
5.3	Build Procedure for Bison and Flex Together	72
5.4	Bison Specification for an Interpreter	75
5.5	AST for Expression Interpreter	76
5.6	Building an AST for the Expression Grammar	78
5.7	Evaluating Expressions	80
5.8	Printing Expressions	81
7.1	The Symbol Structure	107

7.2	A Nested Symbol Table	109
7.3	Symbol Table API	110
7.4	Name Resolution for Declarations	112
7.5	Name Resolution for Expressions	112
8.1	Sample DAG Data Structure Definition	120
8.2	Example of Constant Folding	125
8.3	Example Control Flow Graph	126
9.1	Flat Memory Model	135
9.2	Logical Segments	136
9.3	Multiprogrammed Memory Layout	137
10.1	X86 Register Structure	153
10.2	X86 Register Structure	154
10.3	Summary of System V ABI Calling Convention	160
10.4	System V ABI Register Assignments	162
10.5	Example X86-64 Stack Layout	164
10.6	Complete X86 Example	166
10.7	ARM Addressing Modes	169
10.8	ARM Branch Instructions	172
10.9	Summary of ARM Calling Convention	174
10.10	ARM Register Assignments	175
10.11	Example ARM Stack Frame	177
10.12	Complete ARM Example	178
11.1	Code Generation Functions	182
11.2	Example of Generating X86 Code from a DAG	184
11.3	Expression Generation Skeleton	186
11.4	Generating Code for a Function Call	187
11.5	Statement Generator Skeleton	188
12.1	Timing a Fast Operation	197
12.2	Constant Folding Pseudo-Code	198
12.3	Example X86 Instruction Templates	206
12.4	Example of Tree Rewriting	207
12.5	Live Ranges and Register Conflict Graph	210
12.6	Example of Global Register Allocation	211