New Book *Elements of Compiler Design* by

Alexander Meduna

Taylor and Francis Group, New York, 2007
ISBN: 978-1-4200-6323-3
http://www.fit.vutbr.cz/~meduna/books/eocd

Author

Professor Alexander Meduna, PhD

Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, the Czech Republic

E-Mail: meduna@fit.vutbr.cz
Phone: +420 54114-1232
Fax: +420 54114-1270
Website: http://www.fit.vutbr.cz/~meduna

Subject

Approach

- introductory level
- both theoretical and practical treatment

Pedagogical Goals

- understanding compiler design in theory
- learning how to write a compiler in practice

Keywords

- compiler writing
- lexical analysis
- syntax analysis
- syntax-directed translation
- optimization
- code generation
- automata theory
- formal languages

Courses

Primary course

- one-term introductory course in compiler design at an undergraduate level

Secondary course

- automata theory and formal languages

Theory

Theoretical aspects of this book

- formal models underlying compilation phases
- formalization of the concepts, methods, and techniques employed in compilers
- mathematical foundations of compilation
- formal languages, grammars, automata, and transducers

Practice

Practical aspects of this book

- implementation of compilation techniques
- case study that designs a Pascal-like programming language and its compiler
- many examples and programs
- description of software tools, including yacc and lex

Features and Their Benefits 1/2

- **feature:** presents the essentials of compiler writing in an easy-to-follow way
- **benefit:** students grasp compiler construction quickly and clearly

- **feature:** includes intuitive explanations of theoretical concepts, definitions, algorithms, and compilation techniques
- **benefit:** students easily follow the topics under discussion

- **feature:** examines the mathematical foundations of compiler design and related topics, such as formal languages, automata, and transducers
- **benefit:** demonstrates compilation techniques precisely

Features and Their Benefits 2/2

- **feature:** demonstrates how theory and practice work together in a real-world context through the implementation of algorithms, examples, case studies, and software tools, such as lex and yacc
- **benefit:** enhances comprehension

- **feature:** contains the C++ implementation of a real compiler as well as a variety of programs and challenging exercises, many of which are instructively solved
- **benefit:** demonstrates how to write programs to implement the compilation algorithms

- **feature:** accompanying website provides lecture notes, teaching tips, homework assignments, errata, exams, solutions, and implementation of compilers
- **benefit:** enhances comprehension

Brief Contents

- Preface (14 pages)
- Introduction (20 pages)
- Lexical Analysis (54 pages)
- Syntax Analysis (64 pages)
- Deterministic Top-Down Parsing (20 pages)
- Deterministic Bottom-Up Parsing (26 pages)
- Syntax-Directed Translation and Intermediate Code Generation (28 pages)
- Optimization and Target Code Generation (20 pages)
- Conclusion (6 pages)
- Appendix (16 pages)
- Bibliography (22 pages)
- Indices (10 pages)

Contents 1/5

Preface

Introduction

- Mathematical Preliminaries
- Compilation
- Rewriting Systems

Contents 2/5

Lexical Analysis

- Models
- Methods
- Theory

Syntax Analysis

- Models
- Methods
- Theory

Contents 3/5

Deterministic Top-Down Parsing

- Predictive Sets and LL Grammars
- Predictive Parsing

Deterministic Bottom-Up Parsing

- Precedence Parsing
- LR Parsing

Contents 4/5

Syntax-directed Translation and Intermediate Code Generation

- Bottom-Up Syntax-Directed Translation and Intermediate Code Generation
- Top-Down Syntax-Directed Translation
- Semantic Analysis
- Symbol Table
- Software Tools for Syntax-Directed Translation

Contents 5/5

Optimization and Target Code Generation

- Tracking the Use of Variables
- Optimization of Intermediate Code
- Optimization and Generation of Target Code

Conclusion

Appendix: Implementation

Bibliography

Indices

Competition 1/5

Book Aho, A.V., Lam, M. S., Sethi, R., Ullman, J. D.: *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2006 (ISBN 0321486811)

How this book differs

- too complicated for the undergraduate students

Strength

- a revised and updated version of the famous “Dragon Book.”
- covers all the major topics in compiler design in depth
- used as the basis of a graduate class on compilers

Weakness

- written in somewhat dry and encyclopedic way

Competition 2/5

Book Cooper, K. D. *Engineering a Compiler*. Morgan Kaufmann, 2004 (ISBN 155860698X)

How this book differs

- concentrates its attentions only on the back end of a compiler
- cannot be used at an undergraduate level

Strength

- has a nice layout and gives many examples
- all topics are well connected to each other
- helpful for an advanced computer programmer

Weakness

- avoids any mathematical formalism and theoretical concepts
- text is wordy

Competition 3 / 5

Book Bal, H., Grune, D., Jacobs C., and Langendoen, K.: *Modern Compiler Design*. Wiley, 2000 (ISBN 0471976970)

How this book differs

- beyond the level of bachelor students
- necessary to supplement this book, such as Chapter 3 about attribute grammars, with other books on compilers

Strength

- covers a broad range of concepts used in modern compilers
- explains the compilation of object-oriented, functional, logic, parallel, and distributed languages
- describes the implementation of optimization techniques in detail

Weakness

- algorithms are written in a difficult-to-follow pseudo-code
- exercises at the end of each chapter are rather poor

Competition 4/5

Book Parsons, T. W.: *Introduction to Compiler Construction*. Computer Science, 1992 (ISBN 0716782618)

How this book differs

- describes all formal notions in a very informal way
- difficult to understand how these notions are related to the process of compilation

Strength

- provides a throughout introduction to compiler design
- contains all the essential material concerning compilers

Weakness

- presents all concepts in an obscure way
- reader can hardly grasp the principles of compiler writing
- examples are too trivial and somewhat dated
- contains many minor mistakes and misprints

Competition 5/5

Book Fischer, C. and LeBlanc, R.: *Crafting a Compiler with C*. Addison Wesley, 1991 (ISBN 0805321667)

How this book differs

- beyond the level of bachelor students

Strength

- approaches to writing compilers by using C
- includes numerous programs
- covers many advanced topics concerning code generation, optimization, and real-world parsing
- good reference

Weakness

- necessary to supplement this book with books on automata

A Sample: Precedence Parsing 1/10

***Operations* REDUCE *and* SHIFT**

- In a G-based bottom-up parser, where $G = ({}_G\Sigma, {}_GR)$ is a grammar, we use two operations, **REDUCE** and **SHIFT**, which modify the current *pd* top as follows:
  - **REDUCE**$(A \rightarrow x)$ makes a reduction according to $A \rightarrow x \in {}_GR$
  - **SHIFT** pushes *ins* onto *pd* and advances to the next input symbol

Algorithm 5.2 *Operator Precedence Parser*

Input

- a grammar $G = (\Sigma, R)$
- a *G-op-table*
- $ins = w◀$ with $w \in {}_G\Delta^*$

Output

- **ACCEPT** if $w \in L(G)$, and **REJECT** if $w \notin L(G)$

A Sample: Precedence Parsing 2/10

Method

```
begin
  set pd to u;
  repeat
    case G-op-table [pd-top-terminal, $ins_1$] of
      $|$ : SHIFT;
      $\lfloor$ : SHIFT;
      $\rfloor$ : if G contains a rule $A \to x$ with $x =$ G-op-handle then
            REDUCE($A \to x$);
          else REJECT;                {no rule to reduce by}
      ☹ : REJECT;                     {G-op-table-detected error}
      ☺ : ACCEPT;
    end;                              {case}
  until ACCEPT or REJECT;
end.
```

A Sample: Precedence Parsing 3 / 10

Case Study

$$C \rightarrow C \vee C$$
$$C \rightarrow C \wedge C$$
$$C \rightarrow (C)$$
$$C \rightarrow i$$

	∧	∨	*i*	(	)	◀
∧	⌋	⌋	⌊	⌊	⌋	⌋
∨	⌊	⌋	⌊	⌊	⌋	⌋
i	⌋	⌋	☹	☹	⌋	⌋
(	⌊	⌊	⌊	⌊	|	☹
)	⌋	⌋	☹	☹	⌋	⌋
▶	⌊	⌊	⌊	⌊	☹	☺

Operator Precedence Table

A Sample: Precedence Parsing 4/10

Configuration	*Table Entry*	*Parsing Action*
$\underline{\blacktriangleright}\blacklozenge i \wedge (i \vee i)\blacktriangleleft$	$[\blacktriangleright, i] = \lfloor$	**SHIFT**
$\blacktriangleright\ \underline{i}\blacklozenge \wedge (i \vee i)\blacktriangleleft$	$[i, \wedge] = \rfloor$	**REDUCE**$(C \to i)$
$\underline{\blacktriangleright}\ C\blacklozenge \wedge (i \vee i)\blacktriangleleft$	$[\blacktriangleright, \wedge] = \lfloor$	**SHIFT**
$\blacktriangleright\ C\underline{\wedge}\blacklozenge(i \vee i)\blacktriangleleft$	$[\wedge, (] = \lfloor$	**SHIFT**
$\blacktriangleright\ C\wedge\underline{(}\blacklozenge i \vee i)\blacktriangleleft$	$[(, i] = \lfloor$	**SHIFT**
$\blacktriangleright\ C\wedge(\underline{i}\blacklozenge \vee i)\blacktriangleleft$	$[i, \vee] = \rfloor$	**REDUCE**$(C \to i)$
$\blacktriangleright\ C\wedge\underline{(}C\blacklozenge \vee i)\blacktriangleleft$	$[(, \vee] = \lfloor$	**SHIFT**
$\blacktriangleright\ C\wedge(C\underline{\vee}\blacklozenge i)\blacktriangleleft$	$[\vee, i] = \lfloor$	**SHIFT**
$\blacktriangleright\ C\wedge(C\vee \underline{i}\blacklozenge)\blacktriangleleft$	$[i,)] = \rfloor$	**REDUCE**$(C \to i)$
$\blacktriangleright\ C\wedge(C\underline{\vee}\ C\blacklozenge)\blacktriangleleft$	$[\vee,)] = \rfloor$	**REDUCE**$(C \to C \vee C)$
$\blacktriangleright\ C\wedge\underline{(}C\blacklozenge)\blacktriangleleft$	$[(,)] = \mid$	**SHIFT**
$\blacktriangleright\ C\wedge(C\underline{)}\blacklozenge\blacktriangleleft$	$[), \blacktriangleleft] = \rfloor$	**REDUCE**$(C \to (C))$
$\blacktriangleright\ C\underline{\wedge}\ C\blacklozenge\blacktriangleleft$	$[\wedge, \blacktriangleleft] = \rfloor$	**REDUCE**$(C \to C \wedge C)$
$\underline{\blacktriangleright}\ C\blacklozenge\blacktriangleleft$	$[\blacktriangleright, \blacktriangleleft] =$ ☺	**ACCEPT**

Operator Precedence Parsing

A Sample: Precedence Parsing 5/10

Configuration	***Rule***	***Parse Tree***
$\underline{\blacktriangleright}\blacklozenge i \wedge (i \vee i)\blacktriangleleft$		
$\blacktriangleright\ \underline{i}\blacklozenge\wedge (i \vee i)\blacktriangleleft$	$C \rightarrow i$	$\underline{C}\langle i\rangle\wedge(i\vee i)$
$\underline{\blacktriangleright}\ C\blacklozenge\wedge (i \vee i)\blacktriangleleft$		
$\blacktriangleright\ C\underline{\wedge}\blacklozenge(i \vee i)\blacktriangleleft$		
$\blacktriangleright\ C\wedge\underline{(}\blacklozenge i \vee i)\blacktriangleleft$		
$\blacktriangleright\ C\wedge(\underline{i}\blacklozenge\vee i)\blacktriangleleft$	$C \rightarrow i$	$C\langle i\rangle\wedge(\underline{C}\langle i\rangle\vee i)$
$\blacktriangleright\ C\wedge\underline{(}C\blacklozenge\vee i)\blacktriangleleft$		
$\blacktriangleright\ C\wedge(C\underline{\vee}\blacklozenge i)\blacktriangleleft$		
$\blacktriangleright\ C\wedge(C\vee \underline{i}\blacklozenge)\blacktriangleleft$	$C \rightarrow i$	$C\langle i\rangle\wedge(C\langle i\rangle\vee \underline{C}\langle i\rangle)$
$\blacktriangleright\ C\wedge(C\underline{\vee}\ C\blacklozenge)\blacktriangleleft$	$C \rightarrow C\vee C$	$C\langle i\rangle\wedge(\underline{C}\langle C\langle i\rangle\vee C\langle i\rangle\rangle)$
$\blacktriangleright\ C\wedge\underline{(}C\blacklozenge)\blacktriangleleft$		
$\blacktriangleright\ C\wedge(C\underline{)}\blacklozenge\blacktriangleleft$	$C \rightarrow (C)$	$C\langle i\rangle\wedge\ \underline{C}\langle(C\langle C\langle i\rangle\vee C\langle i\rangle\rangle)\rangle$
$\blacktriangleright\ C\underline{\wedge}\ C\blacklozenge\blacktriangleleft$	$C \rightarrow C\wedge C$	$\underline{C}\langle C\langle i\rangle\wedge\ C\langle(C\langle C\langle i\rangle\vee C\langle i\rangle\rangle)\rangle\rangle$
$\underline{\blacktriangleright}\ C\blacklozenge\blacktriangleleft$		

Construction of Parse Tree by Operator-Precedence Parser

A Sample: Precedence Parsing 6 / 10

Construction of an Operator Precedence Table

I. if *a* is an operator that has a higher mathematical precedence than operator *b*, then *a*⌋*b* and *b*⌊*a*

II. if *a* and *b* are left-associative operators of the same precedence, then *a*⌋*b* and *b*⌋*a*
if *a* and *b* are right-associative operators of the same precedence, then *a*⌊*b* and *b*⌊*a*

III. if *a* can legally precede operand *i*, then *a*⌊*i*
if *a* can legally follow *i*, then *i*⌋*a*

IV. if *a* can legally precede (, then *a*⌊(
if *a* can legally follow (, then (⌊*a*
if *a* can legally precede), then a⌋)
if *a* can legally follow), then)⌋*a*

A Sample: Precedence Parsing 7 / 10

	∧	∨	*i*	(	)	◀
∧	⌋	⌋	⌊	⌊	⌋	⌋
∨	⌊	⌋	⌊	⌊	⌋	⌋
i	⌋	⌋	①	②	⌋	⌋
(	⌊	⌊	⌊	⌊	|	③
)	⌋	⌋	④	⑤	⌋	⌋
▶	⌊	⌊	⌊	⌊	⑥	☺

Precedence Table with Error-Recovery Routines

A Sample: Precedence Parsing 8/10

Table-Detected Errors

①	**configuration**:	$pd_1 = i$ and $ins_1 = i$
	diagnostic:	missing operator between two *i*s
	recovery:	change pd_1 to *C*, then push $\wedge$ onto the *pd* top
②	**configuration**:	$pd_1 = i$ and $ins_1 = ($
	diagnostic:	missing operator between *i* and (
	recovery:	change pd_1 to *C*, then push $\wedge$ onto the *pd* top

…

A Sample: Precedence Parsing 9/10

Reduction Errors

❶	**configuration:**	pd_1 = (and ins_1 =)
	diagnostic:	no expression between parentheses
	recovery:	push C onto the pd top
❷	**configuration:**	$pd_1 \in \{\wedge, \vee\}$ and $ins_1 \notin \{i, (\}$
	diagnostic:	missing right operand
	recovery:	push C onto the pd top

…

A Sample: Precedence Parsing 10/10

Configuration	*Table E.*	*Parsing Action*
$\underline{\blacktriangleright}\blacklozenge i(i\vee)\blacktriangleleft$	$[\blacktriangleright, i] = \lfloor$	**SHIFT**
$\blacktriangleright\underline{i}\blacklozenge(i\vee)\blacktriangleleft$	$[i, (] =$ ②	**table-detected error and rec.** ②
$\blacktriangleright C\underline{\wedge}\blacklozenge(i\vee)\blacktriangleleft$	$[\wedge, (] = \lfloor$	**SHIFT**
$\blacktriangleright C\wedge\underline{(}\blacklozenge i\vee)\blacktriangleleft$	$[(, i] = \lfloor$	**SHIFT**
$\blacktriangleright C\wedge(\underline{i}\blacklozenge\vee)\blacktriangleleft$	$[i, \vee] = \rfloor$	**REDUCE**$(C \to i)$
$\blacktriangleright C\wedge\underline{(}C\blacklozenge\vee)\blacktriangleleft$	$[(, \vee] = \lfloor$	**SHIFT**
$\blacktriangleright C\wedge(C\underline{\vee}\blacklozenge)\blacktriangleleft$	$[\vee,)] = \rfloor$	**Reduction error and recovery** ❷
$\blacktriangleright C\wedge(C\underline{\vee}C\blacklozenge)\blacktriangleleft$	$[\vee,)] = \rfloor$	**REDUCE**$(C \to C\vee C)$
$\blacktriangleright C\wedge\underline{(}C\blacklozenge)\blacktriangleleft$	$[(,)] = \mid$	**SHIFT**
$\blacktriangleright C\wedge(C\underline{)}\blacklozenge\blacktriangleleft$	$[), \blacktriangleleft] = \rfloor$	**REDUCE**$(C \to (C))$
$\blacktriangleright C\underline{\wedge}C\blacklozenge\blacktriangleleft$	$[\wedge, \blacktriangleleft] = \rfloor$	**REDUCE**$(C \to C\wedge C)$
$\underline{\blacktriangleright}C\blacklozenge\blacktriangleleft$	$[\blacktriangleright, \blacktriangleleft] =$ ☺	**REJECT** because of errors ② and ❷

Operator Precedence Parsing with Error-Recovery Routines

Bibliographical Notes in Detail

... Aho, A. V. and Ullman, J. D. [1969a], Aho, A. V. and Ullman, J. D. [1969b], Barnett, M. P. and Futrelle, R. P. [1962], Conway, M. E. [1963], de Bakker, J. W. [1969], Evey, J. [1963], Ginsburg, S. and Rice, H. G. [1962], Hartmanis, J., Lewis, P. M., II, and Stearns, R. E. [1965], Irons, E. T [1961], Johnson, W. L., Porter, J. H., Ackley, S. I., and Ross, D. T. [1968], Kasami, T. [1965], Knuth, D. E. [1967a], Knuth, D. E. [1967b], Korenjak, A. J. and Hopcroft. J. E. [1966], Kurki-Suonio, R. [1964], Landin, P. J. [1965], Lewis, P. M., II and Stearns, R. E. [1968], McCarthy, J. [1960], McCarthy, J. and Painter, J. [1967], Naur, P. (ed.) [1960], Oettinger, A. G. [1961], and van Wijngaarden, A. (ed.) [1969]. During the last three decades of the twentieth century, the basic knowledge concerning the construction of compilers was summarized in the books Aho, A. V. and Ullman, J. D. [1972], Aho, A. V. and Ullman, J. D. [1973], Aho, A. V. and Ullman, J. D. [1977], Aho, A. V., Lam, M. S., Sethi, R. and Ullman, J. D. [2007], Alblas, H. [1996], Appel, A. W. [1998], Bergmann, S. [1994], Elder, J. [1994], Fischer, C. N. [1991], Fischer, C. [1999]**...**

Topics Not Covered in This Book 1/4

Lexical Analysis

- acceleration of the scanning process: scanning ahead on the input to recognize and buffer several next lexemes
- buffering these lexemes by using various economically data-organized methods (pairs of cyclic buffers)
- theory of finite automata
- minimization of the number of states in any deterministic finite automata

Syntax Analysis

- time and space complexity of parsing algorithms
- general parsers based upon tables
- Earley Parsing Algorithm

Topics Not Covered in This Book 2/4

Deterministic Top-Down Parsing

- *k*-symbol lookahead
- LL(*k*) parsers based upon LL(*k*) grammars
- automatic top-down parser generator

Deterministic Bottom-Up Parsing

- generalized precedence parser
- varies constructions of the LR tables and the corresponding LR parsers
- canonical LR parsers
- lookahead LR parsers
- the Brute-Force lookahead LR parsers
- shift-reduce and reduce-reduce problems discussed in detail

Topics Not Covered in This Book 3/4

Syntax-Directed Translation and Intermediate Code Generation

- top-down syntax-directed translation discussed in detail
- semantic pushdown
- stack-implemented tree-structured and hash-structured symbol tables
- more software tools, such as SLK and bison

Optimization and Target Code Generation

- time and space complexity
- optimizing compiler
- run-time memory management
- static memory management
- dynamic memory management
- stack storage and heap storage

Topics Not Covered in This Book 4/4

Theory

- deterministic parsers of non-context-free languages
- conditional grammars
- regulated grammars

Design

- compiler design based upon computational cooperation, distribution, concurrence, and parallelism
- functional, logic, and object-oriented languages and their compilers

Discussion and End