Programming Language Fundamentals by Example

© 2007 by Taylor & Francis Group, LLC

Other Auerbach Publications in Software Development, Software Engineering, and Project Management

Accelerating Process Improvement Using Agile Techniques
Deb Jacobs
0-8493-3796-8

Antipatterns: Identification, Refactoring, and Management
Phillip A. Laplante and Colin J. Neill
0-8493-2994-9

Business Process Management Systems
James F. Chang
0-8493-2310-X

The Complete Project Management Office Handbook
Gerard M. Hill
0-8493-2173-5

Defining and Deploying Software Processes
F. Alan Goodman
0-8493-9845-2

Embedded Linux System Design and Development
P. Raghavan, Amol Lad, and Sriram Neelakandan
0-8493-4058-6

Global Software Development Handbook
Raghvinder Sangwan, Matthew Bass, Neel Mullick, Daniel J. Paulish, and Juergen Kazmeier
0-8493-9384-1

Implementing the IT Balanced Scorecard
Jessica Keyes
0-8493-2621-4

The Insider's Guide to Outsourcing Risks and Rewards
Johann Rost
0-8493-7017-5

Interpreting the CMMI®: A Process Improvement Approach
Margaret Kulpa and Kent Johnson
0-8493-1654-5

Modeling Software with Finite State Machines
Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, and Peter Wolstenholme
0-8493-8086-3

Optimizing Human Capital with a Strategic Project Office
J. Kent Crawford and Jeannette Cabanis-Brewin
0-8493-5410-2

A Practical Guide to Information Systems Strategic Planning, Second Edition
Anita Cassidy
0-8493-5073-5

Process-Based Software Project Management
F. Alan Goodman
0-8493-7304-2

Project Management Maturity Model, Second Edition
J. Kent Crawford
0-8493-7945-8

Real Process Improvement Using the CMMI®
Michael West
0-8493-2109-3

Reducing Risk with Software Process Improvement
Louis Poulin
0-8493-3828-X

The ROI from Software Quality
Khaled El Emam
0-8493-3298-2

Software Engineering Quality Practices
Ronald Kirk Kandt
0-8493-4633-9

Software Sizing, Estimation, and Risk Management
Daniel D. Galorath and Michael W. Evans
0-8493-3593-0

Software Specification and Design: An Engineering Approach
John C. Munson
0-8493-1992-7

Software Testing and Continuous Quality Improvement, Second Edition
William E. Lewis
0-8493-2524-2

Strategic Software Engineering: An Interdisciplinary Approach
Fadi P. Deek, James A.M. McHugh, and Osama M. Eljabiri
0-8493-3939-1

Successful Packaged Software Implementation
Christine B. Tayntor
0-8493-3410-1

UML for Developing Knowledge Management Systems
Anthony J. Rhem
0-8493-2723-7

AUERBACH PUBLICATIONS
www.auerbach-publications.com
To Order Call: 1-800-272-7737 • Fax: 1-800-374-3401
E-mail: orders@crcpress.com

© 2007 by Taylor & Francis Group, LLC

Programming Language Fundamentals by Example

D. E. Stevenson

Auerbach Publications
Taylor & Francis Group
Boca Raton New York

Auerbach Publications is an imprint of the
Taylor & Francis Group, an informa business

© 2007 by Taylor & Francis Group, LLC

Auerbach Publications
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2007 by Taylor & Francis Group, LLC
Auerbach is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-10: 0-8493-7016-7 (Hardcover)
International Standard Book Number-13: 978-0-8493-7016-8 (Hardcover)

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

No part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (http://www.copyright.com/) or contact the Copyright Clearance Center, Inc. (CCC) 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Stevenson, D. E.
Programming language fundamentals by example / D.E. Stevenson.
p. cm.
Includes bibliographical references and index.
ISBN 0-8493-7016-7 (alk. paper)
1. Computer programming--Textbooks. 2. Programming languages (Electronic computers)--Textbooks. I. Title.

QA76.6.S746 2006
005.1--dc22 2006048426

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the Auerbach Web site at
http://www.auerbach-publications.com

CONTENTS

List of Figures . . . xi
The Author . . . xiii
Preface . . . xv

1 A Word about Using This Text . . . 1
1.1 Expectations for the Student and Instructor . . . 1
1.2 Opening Comments . . . 2
1.3 Possible Semester Course . . . 4
1.3.1 General Planning . . . 5
1.3.1.1 Learning Principle 1: Prior Knowledge . . . 5
1.3.1.2 Learning Principle 2: Foundational Knowledge . . . 7
1.3.1.3 Learning Principle 3: Metacognition . . . 8
1.4 Detailed Semester Plan . . . 8
1.4.1 Milestone Maps . . . 8

2 Introduction to the Problem . . . 11
2.1 Memorandum from the President . . . 11
2.2 Background . . . 12
2.2.1 Cambridge Polish . . . 12
2.2.2 A Note on Notation . . . 13
2.2.3 A Note on Names . . . 14
2.3 *SOL* . . . 14
2.4 Primitive Data and Primitive Operations . . . 15
2.4.1 Constants . . . 15
2.4.2 Variables . . . 15
2.4.2.1 Arrays . . . 16
2.4.2.2 Assignment . . . 16
2.5 Primitive Data Operations . . . 16
2.5.1 Stack Control Operations . . . 16
2.5.1.1 Stack Position Variable . . . 17
2.5.1.2 Stack Operations . . . 17
2.6 Control Structures . . . 17
2.6.1 `If`-Selection . . . 17
2.6.2 `Select`-Selection . . . 17

© 2007 by Taylor & Francis Group, LLC

2.6.2.1 While 18
2.6.2.2 Compound Statement: Begin-End 18
2.7 Expressions 18
2.8 Input File Structures and File Scope 19
2.9 Derived Types 20
2.10 Scope and Persistence 21
2.10.1 Scope Definition 21
2.10.2 Scoping within Functions and Compound Statements 21
2.11 Type System 22
2.12 Primitive Functions and Arguments 22
2.13 Built-In Values, Functions, and Pseudo-Functions 23
2.13.1 Reads, Writes, and File Operations 23
2.14 Type Conversions 26

PART I: MILESTONES

3 General Information 31
3.1 General Information on Milestones 31
3.1.1 Contract Grading 31
3.2 Milestone Report 32
3.2.1 Design Information 32
3.2.2 Software Project Management 33
3.2.3 Metacognitive Reflection 33
3.3 General Resources 33
3.4 Makefile Prototype 33
Milestone I Learning a New Language, Gforth 37
Milestone II Lexical Structure and Scanner 47
Milestone III Parsing 59
Milestone IV Type Checking 73
Milestone V Elementary Compiling: Constants Only 83
Milestone VI Scoping and Local Variables 89
Milestone VII User-Defined Functions 95
Milestone VIII User-Defined Complex Data 101
Milestone IX Classes and Objects 109

PART II: GENERAL INFORMATION

4 Project 115
4.1 Class 115
4.1.1 Week 1 115
4.1.2 Week 2 116

© 2007 by Taylor & Francis Group, LLC

4.2 Mind Maps 116
4.2.1 The Exercise 116
4.2.2 Why It Works 117

5 Introduction to Linguistics 119
5.1 Focus Problem 119
5.1.1 Technical Vocabulary 119
5.1.2 Deliverable 120
5.2 Philosophy of Linguistics 121
5.3 Meaning: Language, Mind, and World 122
5.4 What Is Syntax? 123
5.5 Grammar 128
5.6 Semantics 131
5.7 Focused SQRRR 132
5.7.1 Survey and Question 132
5.7.2 Read 134
5.7.3 Recite 134
5.7.4 Review 134

6 Linguistics for Programming Languages 135
6.1 Metalanguage versus Object Language 135
6.2 Surface Structure and Deep Structure 136
6.3 Structural Induction 137
6.4 Interpretation Semantics 138
6.5 What about Pragmatics? 140

7 What Is a General Structure for Compiling? 141
7.1 A General Framework 141
7.2 Types 142
7.3 Recap 143
7.4 Defining a Language 143
7.4.1 Algorithms and Data Representation 144
7.4.2 Evolution 144
7.4.3 Implications for the Project 145

8 A Unified Approach to Design 147
8.1 Introduction to Design 147
8.2 Two Elements 148
8.2.1 Linguistic Issues 148
8.2.2 Schemata 149
8.3 Using These Ideas in Design 150
8.3.1 Understanding the Semantics of Grammars 151
8.3.1.1 Reading Terminals 152
8.3.1.2 Dealing with Nonterminals 152

© 2007 by Taylor & Francis Group, LLC

8.3.2 Designing a Postfix Printer Program 153
8.3.3 Calculator 155
8.4 Design Recap 156

9 Technical Foundations in Programming Languages 159
9.1 Introduction 159
9.2 Binding Times: When Do We Know? 160
9.3 Scope 160
9.4 The λ-Calculus Model 162
9.4.1 Syntax 162
9.4.1.1 Variables 163
9.4.1.2 Introducing Variable Names 163
9.4.2 Semantics 163
9.4.3 Substitution and Rewriting 164
9.4.4 Recursion and the Unfolding Model 164
9.4.5 Arguments, Prologues, and Epilogues 165
9.4.5.1 Argument Passing 166
9.5 Implementation Issues 167

10 Storage Management Issues 169
10.1 Computer Memory 169
10.2 Determining the Size Required 170
10.2.1 Compiler-Defined Primitive Data 171
10.2.2 Space for Structures 171
10.2.3 Arrays 171
10.2.3.1 One-Dimensional Array Addressing Polynomials 173
10.2.3.2 One-Dimensional Arrays with Nonstandard Origins 173
10.2.4 Multidimensional Arrays 174
10.3 Runtime Storage Management 174
10.3.1 Scoping as a Visibility Function 175
10.3.2 Stack Frame 175

11 Personal Software Design Process Portfolio 177
11.1 Time Management for Computer Science Students 178
11.1.1 There Are Not Twenty-Four Hours in a Day 178
11.1.2 A Three-Credit Course Is Nine Hours 178
11.1.3 The Key Is Finding Out How You Spend Your Time 179
11.1.4 Keeping Track of Programming Time 179

© 2007 by Taylor & Francis Group, LLC

11.2 Discovering How Long It Takes to Write a Program 180
11.2.1 Problem Decomposition 181
11.2.2 Work Breakdown Structure 181
11.3 Earned Value Computations 182
11.3.1 Baseline and Schedule Variance Computations 182
11.3.2 Rebudgeting 183
11.3.3 Process 183
11.3.4 Templates 185
11.4 Summary Information for PDSPP0 185
11.5 Requirements Template (RWT) 186
11.6 Time Recording Log (TRT) 187
11.7 Schedule Planning Template (SPT) 187
11.8 Defect Log Template (DLT) 187

12 How Did We Get Here? Where Do We Go from Here? 189
12.1 Looking Behind 189
12.1.1 The Difference Engine 190
12.1.2 The Golden Age of Mathematics 190
12.1.3 Arab Mathematics 190
12.1.4 Hindu Number Systems and Algebra 191
12.1.5 Greek Geometrical Algebra 192
12.1.6 Egyptian Algebra 192
12.1.7 Babylonian Algebra 192
12.1.8 What We Can Learn 193
12.2 The Role of the λ-Calculus 193
12.2.1 The Influence of Logic 195
12.2.2 Computability 195
12.3 Moving Forward from 1954 196
12.4 From 1954 until 1965 197
12.4.1 Imperative Languages from 1954 to 1965 197
12.4.2 Lisp and the Functional Paradigm 198
12.4.3 SNOBOL and Pattern Matching 198
12.4.4 APL and Matrices 199
12.5 Programming Languages from 1965 to 1980 199
12.5.1 Structured Programming 199
12.5.2 The Advent of Logical Programming 199
12.5.3 ML and Types 200
12.5.4 Our Old Friend: Forth 200
12.6 The 1980s and 1990s: The Object-Oriented Paradigm 200
12.7 The Parallel Processing Revolution 200
12.7.1 Architecture Nomenclature 201
12.7.2 Languages for Non-SISD Architectures 201
12.8 Critique of the Present 203

© 2007 by Taylor & Francis Group, LLC

Appendix A 205

References 215

© 2007 by Taylor & Francis Group, LLC

LIST OF FIGURES

1.1 Sample Organization for Introduction 5
1.2 Sample Milestone Schedule 6
1.3 Bloom's Taxonomy of Educational Objectives 7
1.4 Relationship of the Milestones 9
1.5 Control and Data Flow 9
2.1 Tree for `[+ [* 2 3 ] 4 ]` 13
2.2 Drawing of the Tree `[let b [+ [* 2 3] 4]]` 14
2.3 Type Sublanguage 20
2.4 Class Syntax 20
2.5 Booleans 22
2.6 Integers 22
2.7 Floats 23
2.8 Strings 24
2.9 Files and Special Constants 25
2.10 Input Grammar, Part 1 25
2.11 Input Grammar Part 2 26
2.12 *SOL* Input Grammar 26
3.1 Grading Example. Passing Grades Are Determined by Milestone 32
3.2 Makefile Prototype 34
I.1 Milestone I Elements 40
I.2 Case 1 Problems 43
I.3 Sets of Expressions 44
II.1 Beginning Work Breakdown Structure (WBS) Based on Requirements 48
II.2 Finite State Machine, Graphic Presentation 51
II.3 C Program for FSA Graph 52
II.4 Finite State Machine, Graphic Presentation 54
II.5 Simple Database Configuration 56
III.1 Regular Productions for Recognizing Names 63
III.2 Productions for Arithmetic Expressions without Variables 64
III.3 Parsing $1 + 2 * 3$ by Simple Rules of Rewriting 66
III.4 Nondeterministic Parse of $1 + 2 * 3$ 66
III.5 Production System for Arithmetic Expressions without Variables 66

© 2007 by Taylor & Francis Group, LLC

III.6 Derivation Using Production System 69
IV.1 Design Matrix for Addition 80
V.1 Booleans 84
V.2 Integers 85
V.3 Floats 85
V.4 Strings 86
V.5 Files and Special Constants 86
VI.1 Analysis of Scope-Storage Interactions 91
VI.2 Illustration of Scoping Issues 92
VIII.1 Output of Program Processing a Five Element Integer Array 103
VIII.2 Output of `struct` Text Program 103
VIII.3 Class Syntax 104
5.1 Linguistics Word Association Map 120
5.2 Mind Map of the Dictionary Case 126
5.3 Syntax Tree for "My Brother Is a Lawyer" 130
5.4 Concept Map for Syntax and Grammars 131
5.5 Usage of Words in "My Brother Is a Lawyer" 132
5.6 Syntax Tree and Diagram of Sentence 133
7.1 Phases in Compiling 142
7.2 Classes of Statements in Assembler Language 145
7.3 General Classes of Statements in Higher-Level Languages 146
7.4 Design Table for Language Features 146
8.1 The Old Faithful Grammar 151
8.2 Unrolled Old Faithful Grammar 151
8.3 A Program Schema for Programs 153
8.4 Schema for `progF` 153
8.5 `Match` Schema 154
8.6 Program to Print *E* Recognized Terms 154
8.7 Complete Design for Print Program 155
8.8 Complete Design for Print Program (continued) 156
8.9 Complete Design for Gforth Program 156
8.10 Complete Design for Gforth Program (continued) 157
10.1 Scheme of von Neumann Architecture 170
10.2 Test Program in C for `sizeof` 172
10.3 Simple C Program Illustrating Visibility 175
10.4 Illustration of Visibility 176
11.1 Notations for Documenting TUL Form 179
11.2 Work Breakdown Structure Headings 180
11.3 Planned Workload 182
11.4 Earned Value Computation 183
12.1 Architecture Classification by Instructions and Data 202

© 2007 by Taylor & Francis Group, LLC

THE AUTHOR

Professor D.E. (Steve) Stevenson is an associate professor of computer science at Clemson University, where he is also the director of Institute for Modeling and Simulation Applications. He received a B.A. in mathematics in 1965 from Eastern Michigan University, an M.S. in computer science from Rutgers University in 1975, and a Ph.D. in mathematical sciences from Clemson in 1983.

He was an infantry officer for four plus years from 1965–1969. He spent one year in Korea and one year in Viet Nam, where he was an advisor to the 3/3 Battalion, 1st ARVN Division. After leaving the military, he was employed at Bell Telephone Laboratories from 1969–1980. While at Bell Labs, he worked on system support, IMS development (loaned to IBM), and as an internal consultant on performance and database design. Dr. Stevenson worked with early versions of C and Unix. He also worked in many interdisciplinary activities involving modeling and simulation; such activities are now known as computational science and engineering.

Computational science led Dr. Stevenson to leave Bell Labs in 1980 to pursue a Ph.D. degree in mathematical sciences at Clemson, specializing in numerical analysis. Since arriving at Clemson, he has worked to develop computational science concepts. Along with Drs. Robert M. Panoff, Holly P. Hirst, and D.D. Warner, he founded the Shodor Education Foundation, Inc. Shodor is recognized as a premier computational science education materials developer and is known for its education of college and university faculty in authentic modeling and simulation activities for science education. The Shodor activities led Dr. Stevenson to investigate new pedagogies; hence, this text is the culmination of some 40 years of experience in interdisciplinary educational and professional activities.

This work is driven by two sayings, the first attributed to Confucius:

"I hear, and I forget.
I see, and I remember.
I do, and I understand."

The second saying comes from Plato: "The unexamined life is not worth living."

© 2007 by Taylor & Francis Group, LLC

PREFACE

This book is the outgrowth of approximately ten years of experimentation in education. It began when I became dissatisfied with the performance of undergraduate students in computer science studies. It seemed to me that the students were not involved in the process and that their professional skills were not improving. From that point onward, I have found myself far more involved in questions of psychology and educational pedagogy than the material itself.

Although the technical aspects of programming languages have advanced in the past ten years, students still start from the same base of knowledge. I found that students had little understanding of natural language grammars and even less understanding of semantics and pragmatics. Such a lack of understanding meant that the students only had a superficial understanding of language in the broader sense of the word. This lack of understanding of language means that the students have little idea of what language does and therefore how to write a program that transforms language. Therefore, this project starts with an introduction to linguistics.

Before retreating to the hallowed halls of academia, I had the good fortune to work at the "old" Bell Telephone Laboratories. It was my further good fortune to work in a group that had a working relationship with the Unix development folks in Murray Hill, people like Ken Thompson and Dennis Ritchie. This association colored—as it has most of a generation of computer scientists—my perception of language and systems. I have tried to give this text a flavor of the type of decisions that are faced by language designers by answering questions about language internals with "What does C do?" Although I use a broad spectrum of languages in my work, including Fortran 95, Prolog, and OCaml, our students have a very limited experience base. Asking them to consider the details of C gives them practice in using their own powers of deduction.

The second break at Bell Labs was an assignment to IBM to participate in IMS development and a subsequent assignment at Bell Labs in a "consultant" organization. The upshot of this experience is the emphasis in this text on project management. Judging by the numbers of computer science undergraduates versus graduate students, it is clear that most undergraduates must find work in industry. In industry, the rules of the game are

© 2007 by Taylor & Francis Group, LLC

much different than those in academia. Anecdotal evidence with students shows that they have very poor time management skills and even fewer design skills. Although most students have at least one software engineering course, it is rare to find software engineers interested in design. Design is learned by designing.

This is a long way back to the opening idea: students were not engaged and I needed to find a new way of teaching to get the real-world flavor. That new approach is really quite old: problem-based learning and case-based methods. These are two of a group of methods known as inquiry-based or active learning methods. Although I hope this text is a shining example of how to use these ideas in computer science education, I hasten to point out that there is much to be learned about computer science education pedagogy. I urge you to begin by reading *How People Learn* (Bransford, Brown, and Cocking 2000). It is more than worth every second to read it.

AVAILABILITY OF COURSE MATERIALS

1. **Gforth Installation.** This text uses Gforth as the target of the project. Actually, any Forth will do but Gforth is widely available—for free—and is under maintenance. If you are not familiar with the GNU project, it is recommended that you visit their Web site at `www.gnu.org`. Instructions on downloading Gforth are found on the Web site.
2. **Forth Documentation.** There are two major documents supporting Gforth: the 1993 American National Standards Institute (ANSI) Standard and the *Gforth Reference Manual*. Both documents are available at `www.cs.clemson.edu/steve/CRCBook`.
3. **PSDP Forms.** The Microsoft® Excel spreadsheets used for the project management portion of the project are available at `www.cs.clemson.edu/steve/CRCBook`.

© 2007 by Taylor & Francis Group, LLC