

Data Structures
and Algorithms

7231
B422

9960175

John Beidler

DATA STRUCTURES AND ALGORITHMS

An Object-Oriented Approach
Using Ada 95



With 29 Illustrations



E9960175



Springer

John Beidler
Department of Computing Sciences
University of Scranton
Scranton, PA 18510-4664
USA

Series Editors

David Gries
Fred B. Schneider
Department of Computer Science
Cornell University
Upson Hall
Ithaca, NY 14853-7501
USA

Library of Congress Cataloging-in-Publication Data

Beidler, John, 1941-

Data structures and algorithms : an object-oriented approach using

Ada 95 / John Beidler.

p. cm. — (Undergraduate texts in computer science)

Includes bibliographical references and index.

ISBN 0-387-94834-1 (hardcover : alk. paper)

1. Object-oriented programming (Computer science) 2. Ada
(Computer program language) 3. Data structures (Computer science)
4. Computer algorithms. I. Title. II. Series.

QA76.64.B43 1996

005.13'3—dc20

96-23982

Printed on acid-free paper.

© 1997 Springer-Verlag New York, Inc.

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

Production managed by Bill Imbornoni; manufacturing supervised by Joe Quatela.
Camera-ready copy prepared using the author's WordPerfect files.
Printed and bound by R.R. Donnelley and Sons, Harrisonburg, VA.
Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-94834-1 Springer-Verlag New York Berlin Heidelberg SPIN 10523571

UNDERGRADUATE TEXTS IN COMPUTER SCIENCE

Editors
David Gries
Fred B. Schneider

Springer

New York
Berlin
Heidelberg
Barcelona
Budapest
Hong Kong
London
Milan
Paris
Santa Clara
Singapore
Tokyo

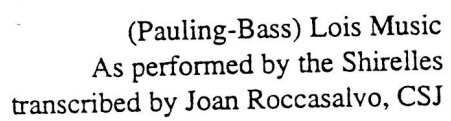
UNDERGRADUATE TEXTS IN COMPUTER SCIENCE

Beidler, Data Structures and Algorithms

Kozen, Automata and Computability

Merritt and Stix, Migrating from Pascal to C++

Zeigler, Objects and Systems



(Pauling-Bass) Lois Music

As performed by the Shirelles

transcribed by Joan Roccasalvo, CSJ

Preface

Picture this: sitting in a cottage by a peat fire in a small farm village just outside of Limerick City in Ireland, with a pint of Guinness, a copy of Booch's software components in Ada book, and a laptop computer. That picture describes how I spent part of my sabbatical during the 1989–1990 academic year, after spending the summer of 1989 working with an Ada programming group at the Naval Surface Warfare Center in Dahlgren, Virginia. This was preceded by many years of looking at ways of improving the Data Structures and Algorithms course as more and more material filtered out of that course and into the CS 2 course. That year in Limerick provided the opportunity to think through the variety of issues that led to this book.

The Data Structures and Algorithms course is commonly referred to as the ACM CS 7 course. During the past decade, a large amount of material has moved from the CS 7 course to the CS 2 course. I viewed this relationship between the CS 2 and CS 7 courses as an opportunity to enhance the CS 2 course and modernize the CS 7 course. Two elements that played key roles in this process are software reuse and object-oriented programming.

My experiences at Dahlgren, during the summer of 1989, convinced me of Ada's value as an educational tool. I found Ada's features to be great tools for enhancing and presenting software development concepts. Of particular importance, I found Ada's encapsulation features and the ability to present specifications without even a hint of implementational details an extremely important software abstraction feature.

While I was on my sabbatical, my colleagues back at the University of Scranton made the decision to select Ada as the core programming language for

the CS 2 and CS 7 courses. We had been using Modula-2 since 1980. Dr. Dennis Martin played a lead role in making the case for the transition to Ada. In fact, the year I was away was a significant year for our department, as we moved to new quarters with new laboratories and a fully networked environment. I'd like to recognize the key role Rich Plishka played in putting it all together.

Between 1990 and 1992 we received two Software Engineering and Ada grants for the development of support resources for the CS 2 and CS 7 courses. This provided us with an opportunity to build upon the resources that were constructed during my sabbatical. In 1992 we started refocusing our materials toward the coming transition to Ada 9X, today we know as Ada 95. I must acknowledge the direct and indirect roles Bob McCloskey played in the development of the data structure suites that I use to support both courses.

This transition to an Ada 95-based resource lead, in the fall of 1994—to experimentation with packages that employed type extension in lieu of generic instantiation as the method of interfacing reusable software components to the needs of clients—eventually led to the construction of a second suite of data structure components that has a very distinct object-oriented flavor.

The Course

The course implied by this book requires substantial software support. I spent my 1989–1990 sabbatical constructing an outline for the course and course support materials, including a component suite, based upon the Booch components, intended to meet educational needs, but with industrial-strength features. I wanted these components to provide good object-oriented support and software reuse experience for students in the CS 2 and CS 7 courses.

It is assumed that the reader of this book is familiar with the topics normally covered in a strong CS 2 course. We expect the CS 2 course to be a broad-based introduction to the discipline with a strong emphasis on analysis, design, abstraction, and the basics of formal specifications, and with a software engineering flavor. In the CS 2 course the students gain a fundamental understanding of the essential concepts of the basic logical data structures, like stacks, queues, lists, and trees. This book moves forward from that foundation with an in-depth presentation of representation, encapsulation, and measurement issues.

There are four recurring themes in this book: abstraction, implementation, encapsulation, and measurement. One significant difference you may find between this book and others is that this book addresses a great variety of

encapsulation issues and separates representational issues from encapsulation issues.

There is more than enough material in this book to support a CS 7 course. My preference is to cover the first six chapters in the first half of the semester. My experience has been that the first two chapters must be addressed carefully. Students never seem to fully grasp the implications of static representations, the fundamental reasons behind the need for both private and limited private types, and the subtleties of tagged and controlled types. While covering Chapters 3 through 6, I emphasize encapsulation and the variety of implementation and measurement issues.

During the second half of the semester, I cover about 80 percent of the material in the remaining chapters, leaving the uncovered material as the basis for individual and team projects. For example, in Chapter 7 I may cover the AVL tree restructuring in depth, then give a cursory presentation of B-trees, leaving the construction of B-tree algorithms as an assignment. Another year I will switch and do B-trees in depth and give various AVL-based assignments.

Support

This book describes, and is supported by, two approaches to encapsulation, the traditional encapsulation of reusable Ada resources in generic packages and a polymorphic approach that makes use of Ada 95's object-oriented features. Both suites may be obtained across the World Wide Web from

`http://academic.uofs.edu/faculty/beidler/`

by following the Ada link. The packages may also be obtained through an anonymous ftp from

`ftp.cs.uofs.edu`

in the pub/Ada directory. For those without network access, the data structure component suites may also be obtained by contacting the author at (717) 941-7774.

Gratia Tibi Ago . . .

So many people contributed in so many ways to this manuscript. Of particular note are the many students who suffered through many variations of course notes

in Cmps 240 from 1991 to 1996. They were the biggest contributors. They pushed, and questioned, and made being an educator a real treat.

I'd like to thank my departmental colleagues. I could not forget our departmental support staff of Mary Alice Lynott and Bill Gunshannon. Without Mary Alice's assistance at critical times, and Bill's work in keeping our "ancient" UNIX systems alive, I wonder if this book would ever have been completed. If it was not for our department's unofficial open-door policy, I could not have tried out many ideas on Rich, Bob, Paul, Bi, Dennis, and Dick. I would also like to thank Chip for allowing me to try to improve his racquetball game.

My sabbatical in Ireland played an important role in getting this book started. If Mary Engel, the associate dean of the College of Arts and Sciences, had not met with Barra O'Connadie of the University of Limerick, my sabbatical would not have come to be. The year at Limerick was made both a pleasure and an academic success by such good folks as Wally Ryder, Tony Cahill, Norah Powers, and Mike Coughlin. Nor can I forget the invaluable contribution of Fionbarr McLaughlin and Liam O'Brien in teaching me to play squash.

In the last few years, numerous discussions with three good "Ada" friends helped this project along. They are John McCormack, Mike Feldman, and Nick DeLillo. Nick and I, along with our wives, share a full appreciation of the other CIA, The Culinary Institute of America. Now if only he could learn to make a good cappuccino.

Of course, what makes this venture so worthwhile is the support and love of my family. They may be added to the dedication, if you figured it out, by replacing the word "one" with the word "ones" when you sing the song. Now that this book is completed, you may find it difficult getting in touch with me. I'm busy playing with my grandchildren.

Jack Beidler
Summer 1996

Contents

Preface	vii
1 Preliminaries	1
1.1 Object-Oriented Software Development	3
1.1.1 The Object-Oriented Concept	3
1.1.2 Objects and Their Attributes	4
1.1.3 Operations	6
1.2 Problem Analysis	6
1.3 Solution Design	8
1.4 Design to Implementation	10
1.5 Software Maintenance	11
1.6 Data Structures and Algorithms	12
1.7 Ada 95	14
1.8 Simple Static Types	15
1.8.1 Ordinal Objects	17
1.8.2 Access Types	19
1.8.3 Nonordinal Objects	26
1.9 Structured Data Types	28
1.9.1 Arrays	31
1.9.1.1 Constrained Arrays	31
1.9.1.2 Unconstrained (Generic) Arrays	35
1.9.2 Records	36
1.9.2.1 Simple Record Structure	36
1.9.2.2 Record Discriminants	37
1.9.2.3 Variant Records	39
1.9.2.4 Protected Types	41
1.9.3 Compound Structures	43
1.10 Explorations	45
2 Encapsulation	49
2.1 Concept	49
2.2 Packaging Fundamentals	50
2.2.1 Initialize/Finalize	50

2.2.2 Private Types and Safe Client/Package Interface	50
2.2.2.1 Private Types and Equality	54
2.2.2.2 Issues Surrounding Limited Private Types	55
2.3 Using Generic Packages	58
2.4 Object-Oriented Support	60
2.4.1 Tagged Types	60
2.4.2 Abstract Types	63
2.4.3 Abstract Subprograms	66
2.4.4 'Class	67
2.4.5 Dynamic Dispatch	68
2.4.6 Controlled Types	70
2.5 Using Object-Oriented Packaging	73
2.5.1 Polymorphic Versus Generic Packaging	77
2.6 A Taxonomy of Client/Package Visibility	77
2.6.1 Safe Handling of User Objects	78
2.6.1.1 Private	78
2.6.1.2 Limited Private	79
2.6.1.3 Tagged	80
2.6.1.4 Controlled	81
2.6.2 Safe Encapsulation of Data Structures	82
2.6.2.1 Private	82
2.6.2.2 Limited Private	83
2.6.2.3 Controlled	84
2.6.2.4 Encapsulated	85
2.7 Composability	86
2.8 Child Units	86
2.8.1 Generic Child Units	88
2.9 Explorations	89
3 Stacks	91
3.1 Linear Structures	91
3.2 Elementary Stacks	93
3.2.1 Abstraction	93
3.2.2 Representation	96
3.2.2.1 Static	96
3.2.2.2 Dynamic	98
3.2.3 Encapsulation	101
3.2.3.1 stack_Pt_Pt	101
3.2.3.2 Dynamic Representations	101

3.2.4 Measurement	106
3.3 Advanced Features	106
3.3.1 Abstraction	106
3.3.2 Representation	107
3.3.2.1 Static	107
3.3.2.2 Dynamic	109
3.3.3 Encapsulation	111
3.3.4 Measurement	111
3.4 Iterators	114
3.4.1 Abstraction	114
3.4.2 Representation	116
3.4.2.1 Static	116
3.4.2.2 Dynamic	118
3.4.3 Encapsulation	119
3.4.4 Measurement	119
3.5 Explorations	120
4 Queues	121
4.1 Elementary Queues	121
4.1.1 Abstraction	121
4.1.2 Representation	124
4.1.2.1 Static: First Thoughts	124
4.1.2.2 Static: Wraparound	125
4.1.2.3 Dynamic: Two-Pointer Queues	127
4.1.2.4 Dynamic: One-Pointer Queues	130
4.1.3 Encapsulation	133
4.2 Advanced Features	136
4.2.1 Abstraction	136
4.2.2 Representation	137
4.2.2.1 Static Wraparound	137
4.2.2.2 Dynamic: One-Pointer Queues	137
4.3 Iterators	139
4.3.1 Abstraction	139
4.3.2 Representation	142
4.3.2.1 Static	142
4.3.2.2 Dynamic	143
4.3.3 Measurement	143
4.4 Explorations	144

5 Lists	145
5.1 Recursive Paradigm	146
5.1.1 Abstraction	146
5.1.2 Representation	150
5.1.2.1 Static	150
5.1.2.2 Dynamic	151
5.1.3 Encapsulation	159
5.1.4 Advanced Support	159
5.1.5 Iterators	160
5.1.6 Measurement	163
5.2 Positional Paradigm	164
5.2.1 Abstraction	165
5.2.2 Representation	167
5.3 Explorations	178
6 Trees	181
6.1 Nonlinear Structures	181
6.2 Binary Trees, Positional Paradigm	185
6.2.1 Abstraction	185
6.2.2 Representation	189
6.3 Binary Trees, Recursive Paradigm	196
6.3.1 Abstraction	196
6.3.2 Representation	199
6.4 Tree Traversals	204
6.4.1 Depth-first	204
6.4.2 Breadth-first	207
6.4.3 Other Schema	208
6.5 Complete Trees	209
6.6 <i>N</i> -ary Trees	210
6.7 Measurement	213
6.8 Explorations	215
7 Tree Applications	217
7.1 Tree Restructuring	217
7.1.1 Binary Search Trees Revisited	217
7.1.2 AVL Trees	221
7.1.3 B-trees	227

7.2	Heaps	233
7.2.1	Heapsort	239
7.2.2	Priority Queues	243
7.2.3	Huffman Coding	244
7.3	Dictionary Trees	247
7.4	Explorations	248
8	Graphs	251
8.1	Graphs and Digraphs	251
8.2	Digraph Specifications	254
8.3	Matrix Representation	257
8.4	Table Representation	260
8.5	Dynamic Representation	263
8.6	Applications	265
8.7	Explorations	267
9	Sets	269
9.1	Specifications	269
9.2	Static Representation	272
9.3	Dynamic Representation	277
9.4	Hashing Representation	279
9.5	Hamming Codes	280
9.6	Bags	284
9.7	Explorations	285
10	Strings	287
10.1	Specifications	287
10.1.1	String Constructors	289
10.1.2	String Observers	291
10.1.3	String Exceptions	292
10.2	Static Representations	293
10.2.1	Ada Strings	294
10.2.2	TurboPascal-like Strings	296
10.2.3	The Sentinel Method	300
10.3	Dynamic Representations	301
10.3.1	Strings as Lists	301
10.3.2	Strings as Piecewise Lists	302
10.3.3	Tradeoffs	304

10.4 String Search Algorithms	306
10.4.1 The Obvious Algorithm	306
10.4.2 A Human Variation	308
10.4.3 Knuth-Morris-Pratt Algorithm	309
10.4.4 Boyer-Moore Algorithm	311
10.4.5 Comparisons	313
10.5 Tradeoffs	314
10.6 Explorations	314
11 Sorting	317
11.1 $O(n^2)$ Sorts	318
11.1.1 Selection Sort	318
11.1.2 Insertion (Bubble) Sort	320
11.1.3 Timing Characteristics	322
11.2 Better Sorts	322
11.2.1 Shell Sort	323
11.2.2 Quicksort	325
11.2.3 Timing Characteristics	329
11.3 $O(n \log n)$ Sorts	330
11.3.1 Heapsort	330
11.3.2 Mergesort	330
11.3.3 Polyphase Merge	333
11.4 Explorations	335
12 Search / Update	337
12.1 Sequential Search	338
12.2 Bisection Method Search	339
12.3 Block Sequential Search/Update	340
12.4 Address Calculation Search/Update	344
12.5 Hashing	350
12.6 Evaluation	353
12.7 Explorations	354
References	355
Index	361

1

Preliminaries

This book presents data structure techniques in the context of object-oriented software development with the eventual implementation of algorithms in Ada 95. Object-oriented software development is a contemporary approach to the design of reliable and robust software. The complexity of the implementation of software systems is a combination of the complexity of the representations of information and the complexity of the algorithms that manipulate the representations. **Data structures** is the study of methods of representing objects, the design of algorithms to manipulate the representations, the proper **encapsulation** of objects in a reusable form, and the **evaluation** of the cost of the implementation, including the measurement of the complexity of the time and space requirements.

Programming languages play an important role in representing the solutions to problems in an efficient, reliable, and maintainable manner. Many modern programming languages support the layered representation of information and algorithms, frequently referred to as abstraction. **Abstraction** is the separation, or layering, of software to distinguish between **what** a data structure represents, or **what** an algorithm accomplishes, from the implementational details of **how** things are actually carried out. Abstraction is very important because frequently there are several competing methods for representing a structure. Usually, there are tradeoffs among competing representations and their algorithms. These