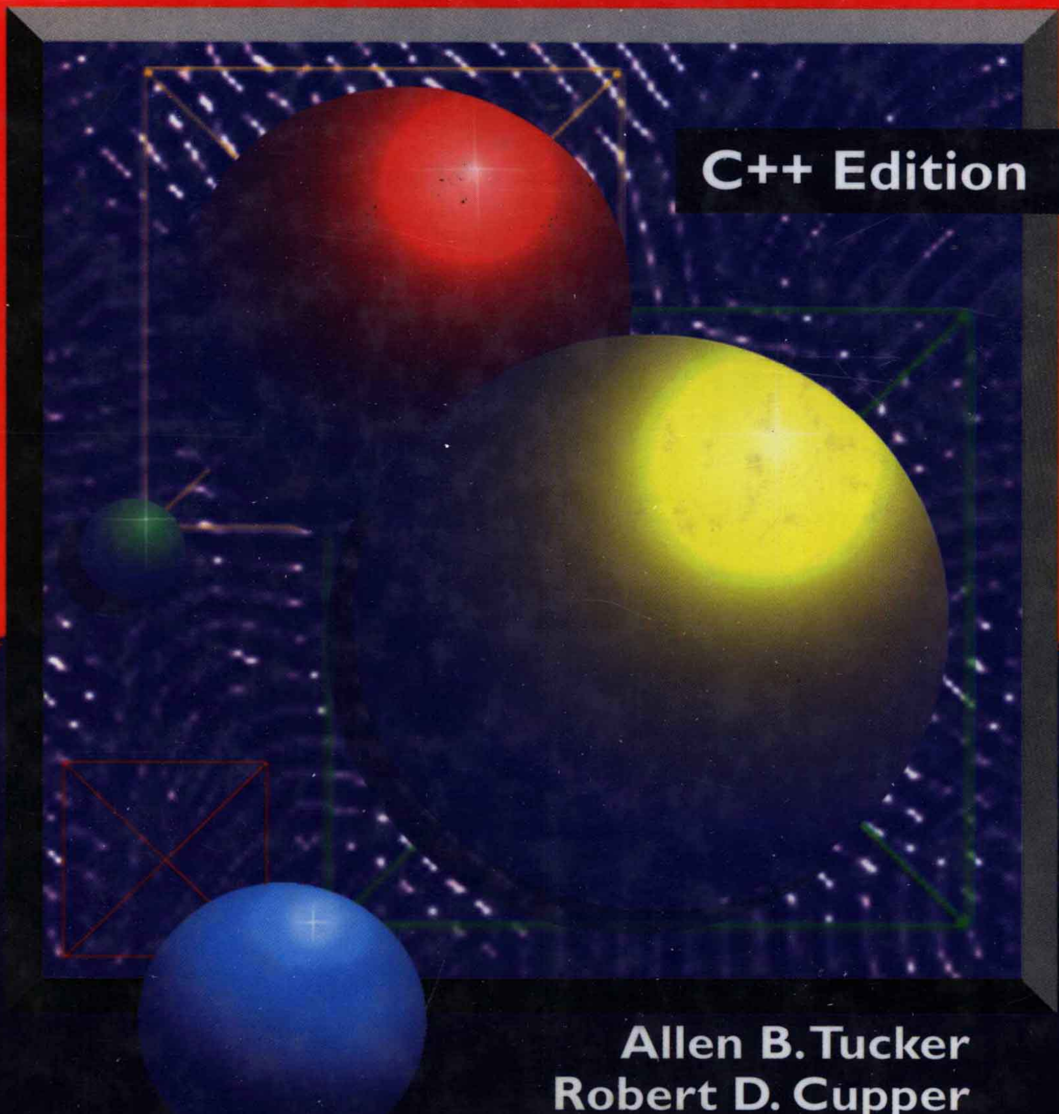
FUNDAMENTALS OF COMPUTING II

Abstraction, Data Structures, and Large Software Systems



C++ Edition

Allen B. Tucker
Robert D. Cupper
W. James Bradley
Richard G. Epstein
Charles F. Kelemen

C++ EDITION

FUNDAMENTALS OF COMPUTING II

Abstraction, Data Structures, and Large Software Systems

Allen B. Tucker
Bowdoin College

Robert D. Cupper
Allegheny College

W. James Bradley
Calvin College

Richard G. Epstein
West Chester University

Charles F. Kelemen
Swarthmore College

McGRAW-HILL, INC.

New York St. Louis San Francisco Auckland Bogotá Caracas
Lisbon London Madrid Mexico City Milan Montreal New Delhi
San Juan Singapore Sydney Tokyo Toronto

FUNDAMENTALS OF COMPUTING II
Abstraction, Data Structures, and Large Software Systems, C++ Edition

This book is printed on recycled, acid-free paper containing 10% postconsumer waste.

2 3 4 5 6 7 8 9 0 FGR FGR 9 0 9 8 7 6 5

ISBN 0-07-065502-2

The editor was Eric M. Munson;
the production supervisor was Paula Flores.
Quebecor Printing/Fairfield was printer and binder.

Library of Congress Catalog Card Number: 94-79884

ABOUT THE AUTHORS

Allen B. Tucker is Professor of Computer Science at Bowdoin College; he has held similar positions at Colgate and Georgetown Universities. He earned a BA in mathematics from Wesleyan University and an MS and PhD in computer science from Northwestern University. Professor Tucker is the author or coauthor of several books and articles in the areas of programming languages, natural language processing, and computer science education. He recently served on the ACM Task Force on the Core of Computing and as cochair of the ACM/IEEE–CS Joint Curriculum Task Force that developed the report *Computing Curricula 1991*. He is a member of ACM, IEEE–CS, CPSR, and the Liberal Arts Computer Science Consortium (LACS).

Robert D. Cupper is Professor and Chair of the Department of Computer Science at Allegheny College. He received a BS from Juniata College and a PhD from the University of Pittsburgh. At Allegheny, Professor Cupper developed one of the first computer science major programs for a liberal arts college, a program that helped motivate the design of the liberal arts model curriculum. He has been an active member of ACM for several years, having served as chair of the Student Chapters Committee and as secretary-treasurer of the Special Interest Group on Computer Science Education (SIGCSE). Professor Cupper has written and spoken on the economics of computing, curriculum development, and accreditation. He is a member of ACM and a cofounder of LACS.

W. James Bradley is Professor of Mathematics and Computer Science at Calvin College. He graduated from MIT with a major in mathematics and completed a PhD in mathematics from the University of Rochester. Professor Bradley also earned an MS in computer science from the Rochester Institute of Technology. He has authored papers in game theory and computer science curriculum, as well as an introductory text in discrete mathematics. His current scholarly interests are in formal methods in decision making, database systems, ethical and social issues in computing, and computer science education. Professor Bradley is a member of MAA, ACM, CPSR, and LACS.

Richard G. Epstein is Professor of Computer Science at West Chester University. He earned a BA in physics at George Washington University and a PhD in computer science at Temple University. He has research interests in the areas of programming languages, object-oriented databases, and curriculum design. He recently served on the ACM/IEEE–CS Joint Curriculum Task Force that developed the report *Computing Curricula 1991*. Professor Epstein is a member of ACM and IEEE–CS.

Charles F. Kelemen is Professor of Computer Science and Mathematics and Director of the Computer Science Program at Swarthmore College. He earned a BA from Valparaiso University and a PhD in Mathematics from the Pennsylvania State University. He has held faculty positions at Ithaca College and LeMoyne College and was Visiting Associate Professor of Computer Science at Cornell. He has published research and educational articles in both mathematics and computer science. His research interests are algorithms and the theory of computation. He holds the Certificate in Computer Programming (Systems Programming) from the Institute for Certification of Computer Professionals. He is a member of ACM, IEEE–CS, CPSR, MAA, and LACS.

Dedicated

Allen B. Tucker: To my wife Meg

Robert D. Cupper: To my wife, Sandy

W. James Bradley: To my wife, Hope

Richard G. Epstein: To my father, David

Charles F. Kelemen: To my wife Sylvia

PREFACE

The discipline of computer science and engineering, or computing, is an extraordinary one. More than any other field of study or professional engagement, the process of solving computational problems and designing computational devices continues to evolve with relentless speed. And so must the curriculum that prepares students to confront the challenges of this unusual discipline.

This text, together with its accompanying laboratory manual and software, is designed for the second course in computing (CS2), which has traditionally confined itself to the study of data structures. Here, we broaden the study of data structures by adding the relevant theory (computational complexity and correctness), modernizing the software design process (using C++ and object-oriented methods), and focusing on the functional elements of operating systems as a compelling application for the study of data structures. The social issue of software reliability is also examined to complete this text. Thus, students who use this text and its laboratory materials will come away not only with an appreciation for the fundamental data structures of computer science but also some strong ideas about their underlying theory, their applications, and the impact of their use on the software systems that they effectively serve.

Overview of This Text

This text has ten chapters and is organized for use in a one-semester course with an accompanying laboratory component. The accompanying *Laboratory Manual* and software are designed to complement the text, so that they should be used in tandem with it. The coordination of laboratory exercises will enrich the textual material that is covered in the lectures.

Chapter 0 reviews important fundamental concepts of programming from the first course. The accompanying Chapter 0 in the laboratory manual contains a C++ tutorial and elementary exercises which can be used by students who are familiar with another language (e.g., Pascal) to gain equivalent familiarity with the elements of C++ that they would encounter in a first course. Students who have already studied C++ in their first course may skip this chapter altogether. That choice will give students more time to work with other aspects of the course.

Chapter 1 provides a detailed introduction to object-oriented software design. It illustrates the basic ideas of class and method by solving a simple programming problem using the object-oriented paradigm. The object-oriented design features of C++ that appear in this chapter are introduced in the accompanying Chapter 1 of the *Laboratory Manual.*

Chapter 2 continues this train of discussion, introducing the use of dynamic objects, virtual methods, inheritance, polymorphism, and a basic class called `Element` that will be used throughout the remainder of the text. Chapter 2 in the *Laboratory Manual* introduces these features as they appear in the C++ language. Because Chapter 2 is a rather long chapter, instructors are encouraged to be selective in its coverage. Sections that are more or less optional are flagged in the table of contents with an asterisk (*). (In fact, optional sections in all chapters are flagged in this way, so that students can remain focussed on the more central issues in the course.)

Chapter 3 focuses on the problem of measuring and classifying the efficiency of algorithms by introducing the notion of computational complexity. It treats the correctness and complexity of various classical sorting and searching algorithms. We recommend that, among these algorithms, students cover at least one $O(n \log n)$ sorting algorithm alongside a conventional $O(n^2)$ algorithm in order to gain an appreciation of the importance of efficiency when making choices among alternatives in algorithm design. Chapter 3 is independent from Chapter 2. It can be covered immediately after Chapter 1 or topics from Chapter 3 can be intermixed with topics from Chapter 2.

Chapters 4 and 5 develop six fundamental data structures in computer science—stacks, queues, lists, binary trees, trees, and graphs—and their applications. Each of these is characterized as a class, and its fundamental operators are identified as the methods of that class. In Chapter 4, applications of stacks are illustrated through the classical problem of evaluating Polish expressions, while applications of queues are explored through the simulation of a waiting line in a bank. There, too, the notion of pseudorandom number generation, so fundamental in the design of computer simulations, is introduced and discussed.

Binary trees, general trees, and graphs are introduced in Chapter 5. The `BinaryTree` class is considered as a restriction of the general `Tree` class, thus giving an important application of the idea of inheritance. The discussion of trees ends with an overview of compilers and the use of the tree class in the design of an expression parser.

Chapter 6 concentrates on the implementation issues that surround these six classes, including tradeoffs in time and space (computational complexity again) between linked and array representations. Chapter 6 in the *Laboratory Manual* contains additional tutorial material on the use of pointers to build linked structures that implement the various classes discussed in Chapters 4 and 5. This organization allows instructors to treat the six classes developed in Chapters 4 and 5 as abstract data types, putting off all implementation details until Chapter 6. Alternatively, it is possible to introduce a class from Chapter 4 or 5 and immediately consider its implementation details from Chapter 6 before going on to another abstract data type.

Chapter 7 draws together the study of object-oriented design and data structures into a single important computer science application—the design of an operating system. Operating systems is a fundamental subject area of computer science, but it is not normally studied until much later in a more traditional curriculum. However, this chapter combines an overview of operating systems with a detailed discussion of key operating system components, providing students with a capstone object-oriented design experience. The tree structure of a UNIX or PC/DOS directory system can be modeled and explored easily using the `Tree` class. The scheduling of memory and processes is also studied in detail. Students can exercise these operating system applications by running the simulations provided in the software and completing the work in Chapter 7 of the *Laboratory Manual.*

Chapter 8 broadens the study of software design by providing a complete overview of the process of software engineering. It contrasts the principles of object-oriented design, used in this text, with the traditional process of function-oriented design. It also introduces software management, testing methods, and the user interface. Thus, it provides a valuable prelude to an intermediate or advanced software engineering course that may appear later in the curriculum.

Chapter 9 discusses three different social dimensions of software design—the dynamics of software teams, the idea of software as property, and the reliability of large software systems. Students have an opportunity here to grapple with important issues that uniquely confront computer scientists and engineers. For instance, how is a large software project organized and managed? What are the risks and liabilities when a complex software system fails? Who owns a software product, and what are the rights and responsibilities of such ownership?

Unless students have had a thorough introduction to C++ in advance, this text provides more material than can be covered in a single semester. Below, we outline different alternative "routes" through this text and lab manual, depending on different student backgrounds and course goals.

Student Audience, Goals, and Alternative Course Organizations

This text assumes that students have already had a first course in computer science, using either Volume I in this series or another introductory text. In either case, we expect that students have had a one–semester introduction to programming and problem solving, in either C++ or another contemporary language (Pascal or Scheme, for instance). We also expect that students will have taken a college level course in discrete mathematics or calculus (or both), either in advance or in parallel with this course. The mathematical discussions are interwoven into this text so that students can see the interplay of mathematics with the study of complexity, correctness, data structures, simulation, and operating systems.

Depending on whether students have had a first course in C++ or a first course in another language, we recommend that one of two alternative "routes" through this text be followed in a one–semester (14–week) course. Route 1 is designed for students who have already had an introduction to C++ programming, while Route 2 is design for students who have not used C++ but have had an introduction to programming in another language.

Text/Lab Chapter	**Topics**	**Route 1 Weeks**	**Route 2 Weeks**
0	C++ tutorial; programs, functions, input/output, arithmetic, specifications (pre– and postconditions)	1–2	
1	Software design; classes, objects, and methods	3–4	1–2
2	Generics, inheritance, polymorphism; class libraries and software reuse	5–6	3–4
3	Complexity, search, and sort; empirical evaluation of sorting	7–8	5–6
4	Stack and queue classes and their methods; the list class and its basic methods; random number generation, simulation, Polish expressions	9	7
5	Trees, binary trees, and graphs; properties and applications	10	8–9
6	Implementation of data structures; linked vs array strategies, complexity issues; C++ pointers and dynamic storage management	11–12	10–11
7	Operating systems and software design; process management, queueing, tree structured directories	13	12–13
8	Overview of software engineering	14	
9	Social issues; software reliability		14

As indicated, Route 1 requires two weeks at the beginning of the semester to bring students up to speed with C++ (Chapter 0 in the *Laboratory Manual* should be particularly helpful in this regard). This time therefore compromises both the breadth and the depth with which data structures and their applications can be covered later in the semester. A more ideal schedule is reflected in

Route 2, which assumes that students are familiar with the rudiments of C++ and can move directly into the object-oriented design aspects of the language. Route 2 provides the luxury of two full weeks' study of trees and graphs and two full weeks' study of operating system applications of data structures.

Other routes through this text are certainly feasible, depending on the instructor's preferences and the course's goals. For instance, our rather brief suggested treatment of stacks and queues (1 week) realistically allows only stacks or queues to be studied in reasonable depth; some may prefer to allocate two weeks for these topics. Courses that wish to emphasize techniques for analysis of algorithms and verification could do all of Chapter 3 immediately after Chapter 1. Different analysis techniques are illustrated in the analyses of the various algorithms presented in Chapter 3. After Chapter 3, the unstarred sections of Chapter 2 could be presented followed by all the material in Chapters 4, 5, and 6. Courses that wish to emphasize object-oriented programming should do all of Chapter 2 and could skip Chapter 3 or just cover a favorite sort. However, to incorporate a reasonable level of breadth into this course, we recommend that at least two weeks be spent working with the material inChapters 7–9 of the text.

Coordination of Laboratory Work

Whether Route 1 or Route 2 is taken through the text, the laboratory material should be coordinated with the text on a week-by-week basis. Each chapter in the *Laboratory Manual* contains detailed descriptions (including program listings in the chapter appendices) of the C++ classes and programs discussed in the corresponding chapter of text; students should frequently reference these details as they read each chapter in the text.

The next page shows a typical laboratory schedule that can be followed for either Route 1 or Route 2. Each lab listed on the right can be done in a week's time, except for the team projects which may require more time.

The laboratory exercises are accompanied by a complete set of software—programs, classes, and data files—to facilitate student laboratory work. This software is on a diskette distributed with the *Instructors Manual*. It may also be obtained directly by sending e–mail to `allen@polar.bowdoin.edu`.

The Breadth–First Approach: The Fundamentals of Computing Series

Readers may know that the course for which this text has been developed is the second in a collection of courses proposed in *Computing Curricula 1991* [2] and labeled as the "breadth-first" curriculum. The general goal of these courses is to provide a broad view of the wide range of subjects in the discipline of computing, an integration of theory with the practice of computing, and a rigorously defined laboratory component. We hope to achieve a curriculum that has much the same goals and style as a two- or three-semester introduction to another science, such as chemistry or biology.

Text/Lab Chapter	Topic	Lab assignment(s)
0	C++ tutorial	Congressional PAC money The gradebook problem
1	Classes and objects	Using the `WeatherObs` class Implementing a class
2	Inheritance and polymorphism	`Element` classes Dynamic objects
3	Searching and sorting	Serial vs binary search Team project—empirical evaluation of sorting algorithms
4	Stacks, queues, and lists	Development of a linked queue application Comparison of random number generators
5	Trees and binary trees	Team project—linked binary tree implementation Binary search trees
6	Implementation issues	Comparison of linked and array implementations of lists
7	Operating systems	Team project—operating system simulation or job scheduler
8	Software engineering	Short (3–5 page) paper on software teams
9	Software reliability	Short (3–5 page) paper on software reliability

This text is therefore the second in a series of texts that are being developed to support the breadth-first approach for the first four courses in the introductory curriculum. At this writing, the first text in this series is also available (in both Pascal and C++ editions) and a Pascal edition of this text is also available. The third and fourth texts are planned for development over the next two or three years. The titles of these texts, which are collectively called the *Fundamentals of Computing Series*, are as follows:

Volume I: Logic, Problem Solving, Programs, and Computers
Volume II: Abstraction, Data Structures, and Large Software Systems

Volume III: Levels of Architecture, Languages, and Applications
Volume IV: Algorithms, Concurrency, and the Limits of Computation

The prerequisite structure assumed here is similar to that which is followed by these courses' counterparts in a traditional curriculum. That is, the course using *Volume I* is a prerequisite for all others, and the course using *Volume II* is a prerequisite for the course using *Volume IV.*

Any of these texts can be used interchangeably with any alternative text for any of the first four courses in the curriculum. For instance, this text can be used in the second course and some alternative for *Volume III* can be used in a more traditionally oriented computer organization course, or vice versa. We have already given some advice on how this text can be used by students whose first course used Pascal or whose first course used a different approach than the breadth-first approach in *Volume I.* In short, the *Fundamentals of Computing Series* is a "loosely coupled" collection of teaching materials designed to cover one or more of the first four courses in the computer science curriculum, and in a wide range of institutional settings.

Acknowledgments

This work results from the toil, suggestions, and support of many people—too numerous to mention individually. Since this text represents a fundamentally new approach to teaching a second course in computer science, we cannot overstate the importance of the feedback we have received from our students and colleagues who have worked through the class testing with us.

In particular, we acknowledge the work of the following reviewers for their contributions to this development and revision process: Art Farley (University of Oregon), Ralph Morelli (Trinity College), Patricia Pineo (Allegheny College), William Punch (Michigan State University), Stephen E. Reichenbach (University of Nebraska), and Antonio Siochi (Christopher Newport University). They have provided immeasurable help in what has been a significant task, and we thank them sincerely.

Allen B. Tucker, Robert D. Cupper, W. James Bradley,
Richard G. Epstein, Charles F. Kelemen

References

[1] P. Denning, D. Comer, D. Gries, M. Mulder, A. Tucker, A. Turner, and P. Young, "Computing as a Discipline," *Report of the ACM Task Force on the Core of Computer Science*, ACM, New York, 1988. Reprinted in *Communications of the ACM* (January 1989) and *Computer* (March 1989).

[2] A. Tucker (ed), B. Barnes, R. Aiken, K. Barker, K. Bruce, J. Cain, S. Conry, G. Engel, R. Epstein, D. Lidtke, M. Mulder, J. Rogers, E. Spaf-

ford, and A. Turner, *Computing Curricula 1991*, ACM/IEEE–CS Joint Curriculum Task Force, ACM and IEEE–CS Press, New York, 1991.

[3] A. Tucker and D. Garnick, "A Breadth-First Introductory Curriculum in Computing," *Computer Science Education 3* (1991), 271-295.

[4] A. Tucker, A. Bernat, J. Bradley, R. Cupper, and G. Scragg, *Fundamentals of Computing I: Logic, Problem Solving, Programs, and Computers*; Pascal and C++ editions, McGraw-Hill (1994 and 1995).

CONTENTS