Preface

The first edition of this text grew out of the author's experience teaching an introductory data structures course (commonly referred to as CS2) for nearly two decades. It has served as a sequel to the widely used *C++: An Introduction to Computing* by Joel Adams and Larry Nyhoff, which grew out of their many years of teaching a first programming course (CS1) in C++. But computer science curricula change as do teaching pedagogy and methodology. In keeping with these changes, the introductory C++ text underwent revisions and has recently appeared in a third edition.

The content of the second course in computing also has changed, with the broadening of the traditional study of data structures to a study of abstract data types (ADTs) being one of the major trends. Consequently, there is an increased emphasis on ADTs in this new edition and a name change thus seemed appropriate: *ADTs, Data Structures, and Problem Solving with C++*. And as one might expect, there is a corresponding increased emphasis on object-oriented design.

In addition, the author's pedagogy has been honed over many years of successful teaching.[1] Reflecting this, the presentation in this new edition has been improved by reordering some topics, rewriting several sections, and adding new material. Many suggestions also came from those who diligently and thoroughly reviewed the manuscript and its several revisions. Their constructive comments and positive evaluations were very encouraging and much appreciated.

To Instructors

If you used the first edition and liked it, I trust you will like this new edition even more. Scan the overview and list of new features that appear later in this preface to see what some of the improvements are. Those of you who haven't used or who stopped using the first edition and are looking at this edition as one of several candidates for your course will, I hope, give it serious consideration. I have tried to preserve the best features of the first edition and made changes based on feedback from many CS2 teachers and users of the previous edition.

Approach As an illustration of the approach that has worked well in my classes, take a look at Chapter 7 on stacks. Some examples of real-world phenomena that are best modeled by a LIFO structure lead to abstracting from these examples the common features, yielding a stack ADT. But ADTs must be implemented with data structures provided in some language, and so we build a stack class. (Incidentally, while we are doing this in class, my students are working on building a queue class in their lab period.)

[1] *Publisher's Note*: Professor Nyhoff's prowess as a teacher was acknowledged when he received the Presidential Exemplary Teaching Award for the academic year 2002–03 after being recommended by Calvin College colleagues along with current and former students.

Once this new `Stack` type has been created and tested, we use it to solve one or more of the original problems and usually at least one new application. I also believe in starting with a simple implementation—e.g., using a static C-style array—and get a working version. Then, emphasizing the need to preserve the public interface of an ADT, we refine it—e.g., use a dynamic array so the user can specify the stack's capacity; then use a linked list so an a priori capacity specification is not needed; and finally, convert it to a template so the ADT can be used with arbitrary type elements. This spiral/successive-refinement approach demonstrates clearly the "abstract" part of an ADT—that it is independent of the implementation.

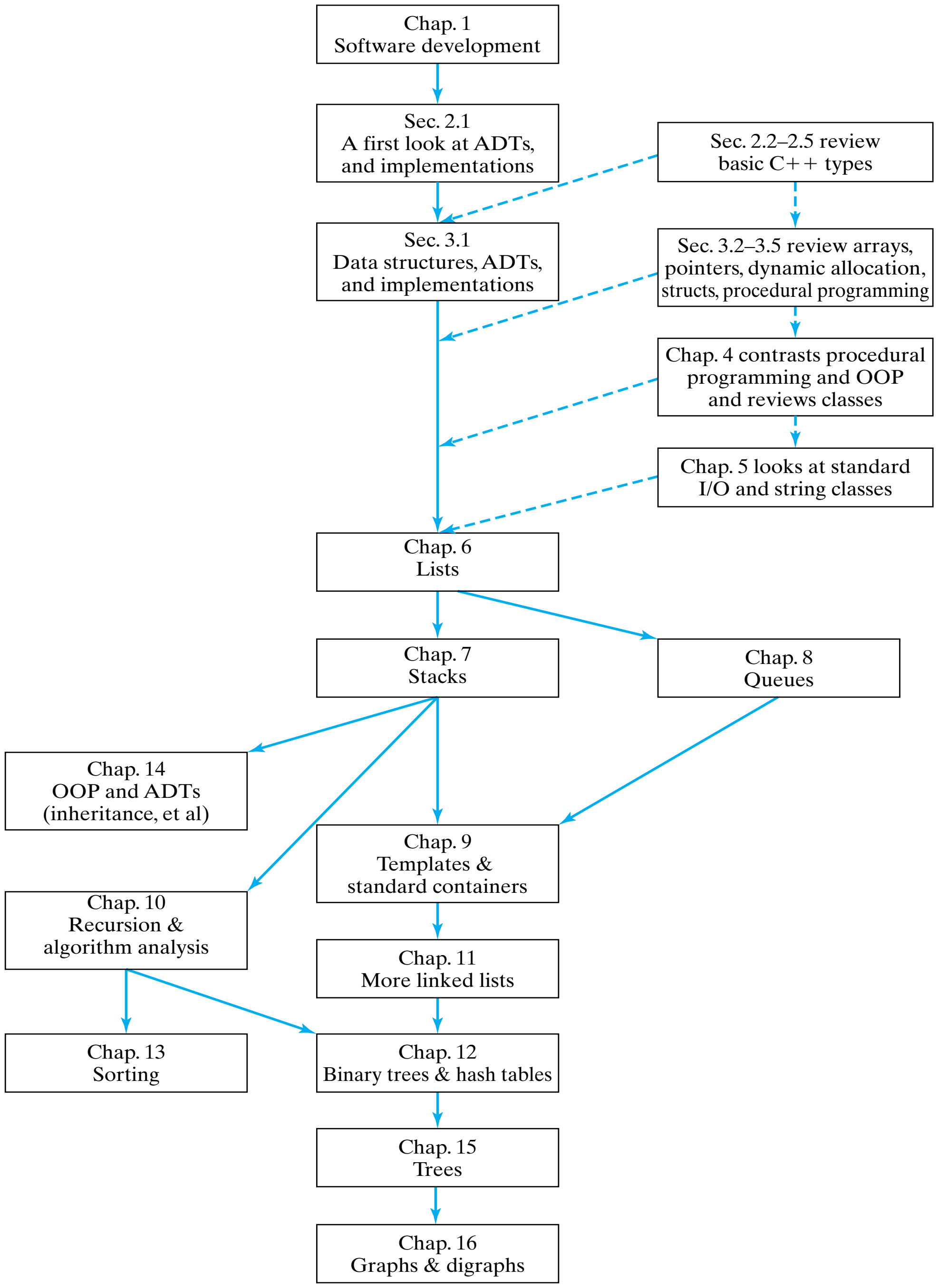
I also cover many of the containers provided in the C++ Standard Template Library (STL), because several of them such as `vector` are very useful and powerful, and it does not make sense to reinvent the wheel by building our own versions. Others, however, such as STL's stacks and queues, are adapters of other containers and waste a lot of the horsepower of these inner containers. For these it makes sense to build our own "lean and mean" implementations, using lower-level data structures such as arrays and linked lists. It also provides practice for students with building customized container types for problems for which none of the standard containers is really suitable.

How to Use this Book There is considerable flexibility in the kind of course that can be taught from this text. In particular, many of the topics can be covered in an order different from that used in the text. The diagram on the next page shows the major dependencies of the various chapters. An arrow running from one box to another indicates a significant dependence of the material in the second box on that in the first; for example, the material in Chapter 9 draws on the material in both Chapters 7 and 8. A dashed arrow indicates that the material in the first box may have been covered in a first course and might be omitted or assigned for review. Boxes not connected are, for the most part, independent of each other (for example, Chapters 7 and 8).

To Students (and Other Users of this Text)

You probably don't read prefaces of most textbooks unless your instructor assigns them and then perhaps only if you will be quizzed on it. For this text, however, you should at least read the section "Overview of the Text," because it is intended to provide an orientation to what the book is about, what its major themes are, and how they fit together. And you should also look through the Table of Contents for this same reason.

The topics covered in this text are typical of those in a course that follows a first course in programming. The aim of these two courses together is to provide you with a solid introduction to computing. You develop the skills to write substantial programs for solving non-trivial problems but are also introduced to important concepts and techniques in computing. These two courses should provide you with a solid base for using the computer as a problem-solving tool in whatever areas of study you pursue. If you do more coursework in computer science, it is important that you work hard at mastering the material of this second course, because the topics covered are fundamental to several upper-level courses. In fact, at many colleges and universities, this course is a prerequisite for intermediate and advanced courses in computer science.


Chap. 1
Software development
Sec. 2.1
A first look at ADTs, and implementations
Sec. 2.2–2.5 review basic C++ types
Sec. 3.1
Data structures, ADTs, and implementations
Sec. 3.2–3.5 review arrays, pointers, dynamic allocation, structs, procedural programming
Chap. 4 contrasts procedural programming and OOP and reviews classes
Chap. 5 looks at standard I/O and string classes
Chap. 6
Lists
Chap. 7
Stacks
Chap. 8
Queues
Chap. 14
OOP and ADTs (inheritance, et al)
Chap. 9
Templates & standard containers
Chap. 10
Recursion & algorithm analysis
Chap. 11
More linked lists
Chap. 13
Sorting
Chap. 12
Binary trees & hash tables
Chap. 15
Trees
Chap. 16
Graphs & digraphs

This text assumes that you have had an introduction to programming, preferably using C++ or Java. Appendix C (*Basic C++*) reviews the basic features of C++ that are typically covered in a first programming course, and Appendix D (*Other C++ Features*) covers some features that are more advanced. Students in my classes have found these two appendixes to be handy references when they need to look up something about the C++ language. If your first course was in Java, you should study Appendix E (*From Java to C++*), which provides a comparison of the main features of the two languages. It has been used successfully in my classes to get students with a background in Java up to speed with the basic features of C++ in the first couple of weeks.

As you read through the text, you should by all means use the Quick Quizzes to check your understanding of some of the main ideas from the reading. The answers to these can be found in Appendix F. These self-test quizzes are usually followed by sets of exercises, some of which your instructor may assign for homework. You are encouraged to try some of these on your own, even if it isn't required, because it will increase your mastery of the material. The same is true of the programming problems at the end of each chapter.

All of the C++ code in the program examples of this text can be downloaded from the author's website for the book: `http://cs.calvin.edu/books/c++/ds`. So if you see a particular function in an example that you can use in a program or class library you are writing, feel free to download and use it—unless your instructor forbids it, of course!

Hopefully, you will enjoy reading and learning from the text. Several hundreds of my students have used earlier versions of this text with very few complaints. But they do enjoy finding errors and informing me about them! I hope that you too will report any that you run across; they are there, in spite of long hours of "debugging" the manuscript before it goes into print. I can't offer you bonus points on your course grade for finding them, but I will recognize your contribution to the improvement of the book by adding your name to the list of other error detectors on the book's website.

Overview of the Text

As the title suggests, there are three main themes in this text:

1. Abstract data types (ADTs)
2. Data structures
3. Problem solving

Abstract data types consist of collections of data elements together with basic operations on the data. Nearly every chapter of this text deals with some aspect of ADTs—defining an ADT such as a list, stack, or queue; studying some application of it; implementing the ADT or studying its implementation in some library; looking at ways to improve the implementation.

Classes play a key role in implementing ADTs because they make it possible to encapsulate the data and the operations so that objects not only store data but also have built-in operations. This is one of the key properties of object-oriented programming and is emphasized from the beginning. Data structures provided in C++ (such as arrays) or that can be built in C++ (e.g., linked lists) play important roles in providing structures to store the data elements of an ADT. These key data structures along with the up-to-date and powerful containers from the Standard Template Library (STL) are studied for this purpose.

The third theme is problem solving. Chapter 1 describes some of the software engineering methodologies used to develop a solution to a problem, and the text emphasizes the use of object-oriented design (OOD) in the design phase. This is a natural continuation of the object-centered design (OCD) approach used in *C++: An Introduction to Computing* and which is similar to that used in many other introductory programming texts. The text has many examples, including several case studies, that show the roles that ADTs play in problem solving.

Implementing the operations of an ADT involves designing algorithms to carry out the operations. This means that the study of algorithms must also play a significant role in a study of ADTs, and this text has many examples of algorithms. These include searching and sorting algorithms along with the powerful algorithms from the Standard Template Library (STL). Analyzing the efficiency of algorithms is also introduced and illustrated, thus providing a first look at important tools used in later courses in computer science.

Algorithms must be implemented in a programming language. Thus this text includes some coverage of C++, especially the more advanced topics not usually covered in a first course and which students need to learn. These include recursion, function and class templates, inheritance, and polymorphism. The C++ features presented conform to the official standard for C++. In addition, some of the C-style topics appropriate in a data structures course are included for several reasons: many students will get jobs as C programmers; many libraries and operating system utilities are written in C or C-style languages; data structures provided in C are usually implemented very efficiently and they are often used to implement some of the more modern standard data types.

Another feature of this text is that it continues the portrayal of the discipline of computer science begun in *C++: An Introduction to Computing* by including examples and exercises to introduce various areas of computer science and thereby provide a foundation for further studies in computer science. The topics include:

- Descriptions of software development methods
- Introduction to data encryption schemes (DES and public key)
- Data compression using Huffman codes
- Doubly-linked lists and large integer arithmetic
- Random number generation and simulation
- Lexical analysis and parsing
- Postfix notation and generation of machine code
- Simple systems concepts such as input/output buffers, parameter-passing mechanisms, address translation, and memory management

New and Improved Features

- Revised first chapter:
    - Introduces other software engineering methods besides the waterfall model.
    - Introduces UML

  - Describes top-down design and object-oriented design in detail
  - Relates some of the “horror stories” of bad software design

- More use of OOD and OOP in examples
- Uniform method of displaying ADT specifications in a UML-style diagram
- Improved pseudocode in algorithms
- Naming conventions that are consistent with common recommendations
- Complete source code for nearly all of the ADTs
- Many improvements to diagrams and several new diagrams
- Expanded and improved discussion of C++'s I/O and string classes (Chapter 5)
- Earlier introduction to pointers and dynamic allocation, including an expanded discussion of the `new` operator (Chapters 2 and 3)
- Earlier presentation of lists (Chapter 6)—before stacks and queues—and revised to include array-based (static and dynamic) list classes, an introduction to linked lists, and more standard symbols in diagrams
- Array-based and linked-list implementations of stacks (Chapter 7)
- Expanded treatment of queues, including array-based and linked-list implementations, and a revised simulation case study (Chapter 8)
- New chapter on searching, including modified and expanded treatment of binary search trees and hash tables (Chapter 12)
- Added discussion of heaps, priority queues, and radix sort (Chapter 13)
- Revised chapter on inheritance (Chapter 14)
- Chapter objectives and end-of-chapter summaries
- Several case studies
- A new appendix *From Java to C++* for those making a transition from Java to C++

Other Key Features

- Self-test Quick Quizzes with answers in back of text
- Large number of written exercises and programming problems
- Chapter notes, programming pointers, and ADT tips at the end of each chapter
- More conformance to the C++ standard than many other texts
- Solid introduction to the C++ Standard Template Library
- A review of C++ in the appendixes for handy reference (Appendices C and D)
- Boxed displays to set off important concepts
- A new design that makes the text more readable and attractive
- Effective use of color to highlight important features and not simply for decoration

- Icons that point out key features, special things to note, and warnings:

Watch out! (potential pitfall)	Note (important feature or concept)
ADT	Algorithm
Designing/building an implementation of an ADT	Using an ADT
Website (more material about a topic)	Chapter notes
Programming pointers	Programming problems
Quick quiz	Exercises

Supplementary Materials

A number of supplementary materials are available for this text:

- An online solutions manual containing solutions for all of the written exercises. Access to these solutions is available to those who adopt this text for use in a course. Solutions to many of the programming problems are also available to them upon request from the author.
- Author website (`http://cs.calvin.edu/books/ds`) and Prentice Hall website (`http://www.prenhall.com/nyhoff`) contain the following:
  - Downloadable source code for text examples
  - Solutions to case studies including source code
  - PowerPoint slides
  - Other supplementary material
- A lab manual with lab exercises and projects (sold separately and also available as a value pack option). It is coordinated with the presentation in the text, reinforcing and expanding what students read there and hear in class.
- Also available are software value pack options that include:
  - Microsoft Visual C++
  - Metrowerks CodeWarrior Learning Edition

Acknowledgments

I express my sincere appreciation to all who helped in any way in the preparation of this text. My gratitude for friendship, perceptive suggestions and directions, and unflagging support and encouragement goes to Alan Apt, a publisher highly respected throughout the publishing and academic communities, and to my editor Toni Holm; their friendship over the past several years has made textbook writing for Prentice Hall an enjoyable experience. I must also thank art director Heather Scott, production editors Chirag Thakkar and Irwin Zucker, and all the others who did such a fantastic job of designing this attractive book and actually getting it into print. Their attention to details has compensated for my lack thereof and their cooperation and kind words were much appreciated. I also appreciate the management of reviews and other details handled by Jake Warde. And I appreciate the many valuable observations and recommendations by the following reviewers of the manuscript; they have strengthened the presentation significantly:

Ping Chen (University of Houston)
Joe Derrick (Radford University)
Eamon Doherty (Farleigh Dickinson University)
James Durbano (University of Delaware)
Eduardo Fernandez (Florida Atlantic University)
Christopher Fox (James Madison University)
Mahmood Haghighi (Bradley University)
Oge Marques (Florida Atlantic)
Mark McCullen (Michigan State University)
William McQuain (Virginia Tech)
Jim Miller (Kansas University)
Jim Richards (Bemidji State University)
Robert Schneider (University of Bridgewater)
Joseph Shinnerl (University of California, Los Angeles)
Michael Stiber (University of Washington)
Al Verbanec (Pennsylvania State University)
John M. Weiss (South Dakota School of Mines and Technology)
Rick Zaccone (Bucknell University)

And, of course, I must once again pay homage to my wife Shar and to our children and grandchildren—Jeff, Dawn, Rebecca, Megan, and Sara; Jim; Greg, Julie, Joshua and Derek; Tom, Joan, Abigail, Micah, and Lucas—for their love and understanding through all the times that their needs and wants were slighted by my busyness. Above all, I give thanks to God for giving me the opportunity, ability, and stamina to prepare this text.

Larry Nyhoff