Koffman and Wolfgang introduce data structures in the context of C++ programming. They embed the design and implementation of data structures into the practice of sound software design principles that are introduced early and reinforced by 20 case studies.

Data structures are introduced in the C++ STL format whenever possible. Each new data structure is introduced by describing its interface in the STL. Next, one or two simpler applications are discussed then the data structure is implemented following the interface previously introduced. Finally, additional advanced applications are covered in the case studies, and the cases use the STL.

In the implementation of each data structure, the authors encourage students to perform a thorough analysis of the design approach and expected performance before actually undertaking detailed design and implementation. Students gain an understanding of why different data structures are needed, the applications they are suited for, and the advantages and disadvantages of their possible implementations.

Case studies follow a five-step process (problem specification, analysis, design, implementation, and testing) that has been adapted to object-oriented programming. Students are encouraged to think critically about the five-step process and use it in their problem solutions. Several problems have extensive discussions of testing and include methods that automate the testing process. Some cases are revisited in later chapters and new solutions are provided that use different data structures.

The text assumes a first course in programming and is designed for Data Structures or the second course in programming, especially those courses that include coverage of OO design and algorithms. A C++ primer is provided for students who have taken a course...
in another programming language or for those who need a review in C++. Finally, more advanced coverage of C++ is found in an appendix.

**Course Hierarchy:**

Course is the second course in the CS curriculum

Required of CS majors

Course names include Data Structures and Data Structures & Algorithms

---

**ABOUT THE AUTHOR**

**Elliot Koffman** is currently Professor of Computer and Information Sciences at Temple University. He received his Ph.D. at Case Institute of Technology. He has authored and co-authored numerous textbooks and other publications.

**Paul Wolfgang** is currently Professor of Computer and Information Sciences at Temple University. He received his B.S. in electrical engineering at University of Pennsylvania.

---

**RELATED RESOURCES**

**Student**

View Student Companion Site

**Instructor**

View Instructor Companion Site

Contact your Rep for all inquiries

---

**NEW TO EDITION**

- **Boxed features** (Pitfalls, Design Concepts, Program Style, Syntax Boxes) are called-out in the text, making key “do’s and don’ts” easy to remember

- **Updated problems**: A variety of problem types can be used for self-assessment and homework – self-check questions, review questions, programming exercises, programming projects
FEATURES

- **Data structures follow the C++ Standard Template Library (STL) format.** Students are encouraged to use the STL as a resource for their programming. New data structures are introduced by specifying an abstract data type as an interface that is adapted from the C++ STL. The data structure implementations in the text are simplified versions of the STL implementations.

- **Data structures are taught in the context of software design principles.** Early chapters present software design concepts, efficiency, and testing; the data structures chapters apply these principles to the design and implementation of data structures needed to solve particular problems. Students gain an understanding of why different data structures are needed, the applications they are suited for, and the advantages and disadvantages of their possible implementations. Emphasis on documentation is provided by using the documentation style developed for Java (Javadoc) which is also applicable to C++ classes and functions.

- **Objects covered early.** Object-oriented principles are introduced early and used throughout in the problem-solving process and in the design and use of C++ classes. Judicious use of UML class diagrams show relationships between classes and emphasize the advantages of good class design and software reuse. The text illustrates how to use inheritance to build new classes from earlier ones. See for examples—
  - AVL tree and Red Black tree classes in Chapter 11 are extensions of the binary search tree class which in is an extension of the binary tree class
  - See UML diagrams in Figs. 8.15 (p. 469), 11.8 (p. 628), 11.18 (p. 634), and 11.31 (p. 650)
  - Students requiring review of C++ and object-oriented issues can refer to C++ Primer

- **Case studies reinforce good programming practice.** Case studies follow a five-step process, sometimes iterative, to model the software engineering principles used in industry: problem specification, analysis, design, implementation, and testing. This process encourages students to think before they code and work through design decisions before implementing a solution. Case Studies reinforce the importance of testing through discussion of test cases relevant to the solution. Several case studies illustrate design principles by revisiting earlier problems to show how they may be solved or a solution refined with a different data structure. See examples—
  - Phone directory case which uses an array in Sections 1.6 and 1.7 and is revisited using a vector in Section 4.1 (p. 239) and a map in Section 9.6 (p. 558)
  - A term paper index using a binary search tree is covered in Section 8.4 (p. 479) and using a map in Section 8.2 (p. 527)
• The Huffman tree is implemented as a priority queue in Section 8.6 (p. 498) and a map is used in Section 9.6 (p. 561) to complete the case

• **Emphasis on modern C++ programming style.** The textbook emphasizes a C++ programming style that facilitates object-oriented programming. The text begins with an extensive primer on C++ that would be very useful for someone who has previously studied Java or needs a C++ refresher. Unlike earlier data structures books in C++, this text emphasizes a programming style that supports object-oriented programming, beginning with a discussion of classes in Chapter 1 and class hierarchies in Chapter 3 which is carried throughout the book. Sections of primary interest to Java programmers would be Section P.5 which focuses on objects, pointers, and dynamic allocation, Section P.6 which discusses call-by-reference parameters, Section P.7 which discusses arrays and how to access static and dynamically allocated arrays through pointers, and Section P.9 which discusses input/output streams.

• **Emphasis on iterators:** Iterators introduced in chapters 4

  • The list class and Iterator in Section 4.6 (p. 264)

  • Implementing the Iterator in Section 4.7 (p. 268)

  • Show how to use them to access elements of lists and other STL data structures and also to access array elements.

  • Demonstrate how to use iterator arguments with the standard functions in the algorithm class in Section 4.10 (p. 297)

  • Use iterators in the sorting chapter (Chapter 10), so that the sorting functions in this chapter will be able to sort the data in several STL classes in addition to arrays. For example, the selection sort function in Section 10.2 (p. 575), the merge sort function, Section 10.7 (p. 592).

• **Thorough coverage of testing process.** See examples—

  • Ch 2 introduces program correctness and efficiency by discussing exception and exception handling, testing strategies, and has brief coverage of big-O notation.

  • Testing the ordered list class in Section 4.8 (p. 290)

  • Testing a binary search tree in Sections 8.4 (p. 479 and p. 483)

  • Testing the Huffman Tree in Section 8.6 (p. 503) and Section 9.6 (p. 563)

  • Testing the sorting functions in Section 10.10 (p. 614)
To purchase this product, please visit https://www.wiley.com/en-us/9780471467557