大学计算机教育国外著名教材系列（影印版）

PEARSON Addison Wesley

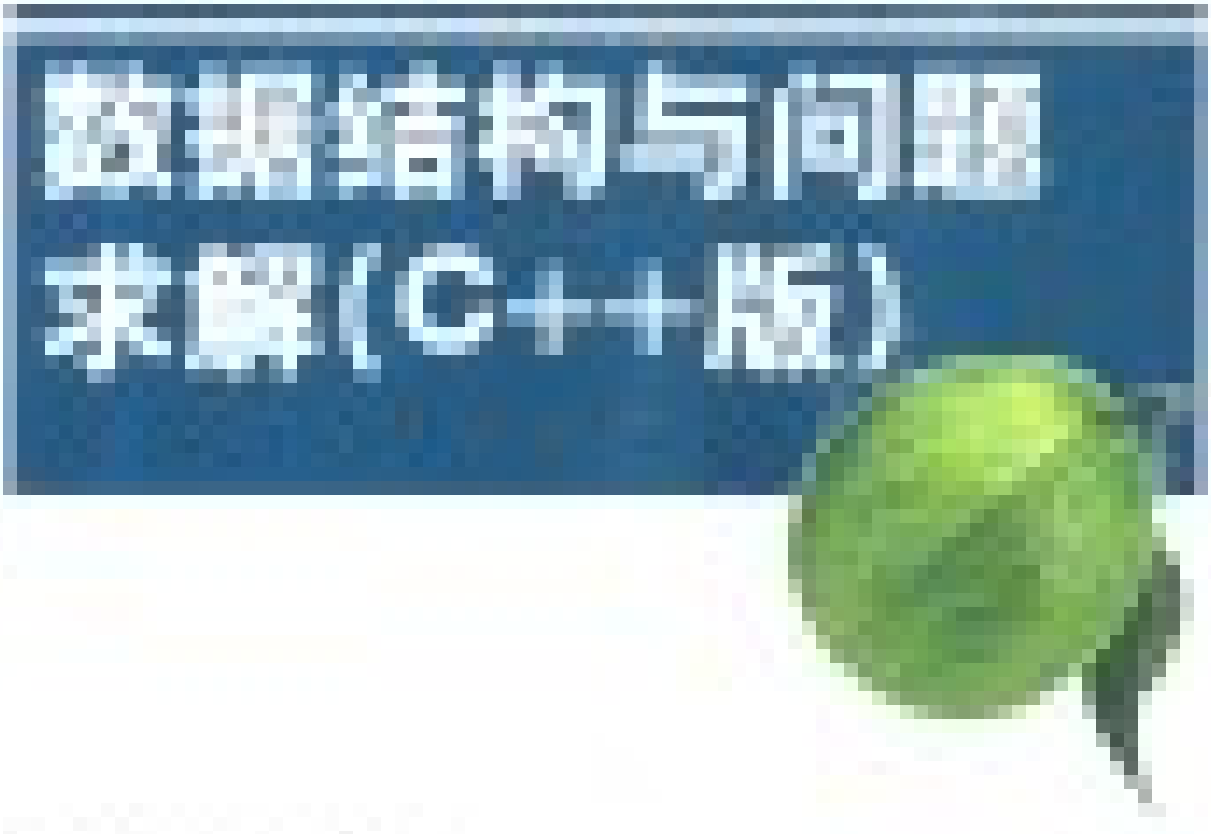
DATA STRUCTURES AND PROBLEM SOLVING USING C++ (2nd Edition)

数据结构与问题求解(C++版)

Mark Allen Weiss 著

清华大学出版社

DATA STRUCTURES AND PROBLEM SOLVING USING C++



大学计算机教育国外著名教材系列(影印版

Data Structures and Problem Solving Using C ++

(2nd Edition)

数据结构与问题求解(C++版)

Mark Allen Weiss 著

清华大学出版社

北京

图书在版编目(CIP)数据

数据结构与问题求解: C++ 版 = Data Structures and Problem Solving Using C++, 2nd Edition/(美)威尔斯(Weiss, M. A.)著. —影印本. —北京:清华大学出版社,2004. 10
(大学计算机教育国外著名教材系列)
ISBN 7-302-09765-8

Ⅰ. 数… Ⅱ. 威… Ⅲ. ①数据结构—英文 ②C++ 语言—程序设计—高等学校—教材—英文
Ⅳ. ①TP311. 12 ②TP312

中国版本图书馆 CIP 数据核字(2004)第 107003 号

出 版 者: 清华大学出版社　　地　　址: 北京清华大学学研大厦
http://www.tup.com.cn　　邮　　编: 100084
社总机: 010-62770175　　客户服务: 010-62776969
责任编辑: 常晓波
印 刷 者: 清华大学印刷厂
装 订 者: 三河市金元装订厂
发 行 者: 新华书店总店北京发行所
开　　本: 185×230　印张: 60.5
版　　次: 2004 年 10 月第 1 版　2004 年 10 月第 1 次印刷
书　　号: ISBN 7-302-09765-8/TP·6742
印　　数: 1~4000
定　　价: 84.00 元

出 版 说 明

进入21世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从1996年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书，受到国内读者的欢迎和支持。跨入21世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高等本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列(影印版)”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好，更适合高校师生的需要。

清华大学出版社

Preface

This book is designed for a two-semester sequence in computer science, beginning with what is typically known as Data Structures (CS-2) and continuing with advanced data structures and algorithm analysis.

The content of the CS-2 course has been evolving for some time. Although there is some general consensus concerning topic coverage, considerable disagreement still exists over the details. One uniformly accepted topic is principles of software development, most notably the concepts of encapsulation and information hiding. Algorithmically, all CS-2 courses tend to include an introduction to running-time analysis, recursion, basic sorting algorithms, and elementary data structures. An advanced course is offered at many universities that covers topics in data structures, algorithms, and running-time analysis at a higher level. The material in this text has been designed for use in both levels of courses, thus eliminating the need to purchase a second textbook.

Although the most passionate debates in CS-2 revolve around the choice of a programming language, other fundamental choices need to be made, including

- whether to introduce object-oriented design or object-based design early,
- the level of mathematical rigor,
- the appropriate balance between the implementation of data structures and their use, and
- programming details related to the language chosen.

My goal in writing this text was to provide a practical introduction to data structures and algorithms from the viewpoint of abstract thinking and problem solving. I tried to cover all of the important details concerning the data structures, their analyses, and their C++ implementations, while staying

away from data structures that are theoretically interesting but not widely used. It is impossible to cover in a single course all the different data structures, including their uses and the analysis, described in this text. So, I designed the textbook to allow instructors flexibility in topic coverage. The instructor will need to decide on an appropriate balance between practice and theory and then choose those topics that best fit the course. As I discuss later in this Preface, I organized the text to minimize dependencies among the various chapters.

A Unique Approach

My basic premise is that software development tools in all languages come with large libraries, and many data structures are part of these libraries. I envision an eventual shift in emphasis of data structures courses from implementation to use. In this book I take a unique approach by separating the data structures into their specification and subsequent implementation and take advantage of an already existing data structures library, the Standard Template Library (STL).

A subset of the STL suitable for most applications is discussed in a single chapter (Chapter 7) in Part II. Part II also covers basic analysis techniques, recursion, and sorting. Part III contains a host of applications that use the STL's data structures. Implementation of the STL is not shown until Part IV, once the data structures have already been used. Because the STL is part of C++ (older compilers can use the textbook's STL code instead—see *Code Availability*, xxix), students can design large projects early on, using existing software components.

Despite the central use of the STL in this text, it is neither a book on the STL nor a primer on implementing the STL specifically; it remains a book that emphasizes data structures and basic problem-solving techniques. Of course, the general techniques used in the design of data structures are applicable to the implementation of the STL, so several chapters in Part IV include STL implementations. However, instructors can choose the simpler implementations in Part IV that do not discuss the STL protocol. Chapter 7, which presents the STL, is essential to understanding the code in Part III. I attempted to use only the basic parts of the STL.

Many instructors will prefer a more traditional approach in which each data structure is defined, implemented, and then used. Because there is no dependency between material in Parts III and IV, a traditional course can easily be taught from this book.

Prerequisites

Students using this book should have knowledge of either an object-oriented or procedural programming language. Knowledge of basic features, including primitive data types, operators, control structures, functions (methods), and input and output (but not necessarily arrays and classes) is assumed.

Students who have taken a first course using C++ or Java may find the first two chapters "light" reading in some places. However, other parts are definitely "heavy" with C++ details that may not have been covered in introductory courses.

Students who have had a first course in another language should begin at Chapter 1 and proceed slowly. They also should consult Appendix A which discusses some language issues that are somewhat C++ specific. If a student would like also to use a C++ reference book, some recommendations are given in Chapter 1, pages 38–39.

Knowledge of discrete math is helpful but is not an absolute prerequisite. Several mathematical proofs are presented, but the more complex proofs are preceded by a brief math review. Chapters 8 and 19–24 require some degree of mathematical sophistication. The instructor may easily elect to skip mathematical aspects of the proofs by presenting only the results. All proofs in the text are clearly marked and are separate from the body of the text.

Summary of Changes in the Second Edition

1. Much of Part I was rewritten. In Chapter 1, primitive arrays are no longer presented (a discussion of them was moved to Appendix D); vectors are used instead, and `push_back` is introduced. Pointers appear later in this edition than in the first edition. In Chapter 2, material was significantly rearranged and simplified. Chapter 3 has additional material on templates. In Chapter 4, the discussion on inheritance was rewritten to simplify the initial presentation. The end of the chapter contains the more esoteric C++ details that are important for advanced uses.
2. An additional chapter on design patterns was added in Part I. Several object-based patterns, including Functor, Wrapper, and Iterator, are described, and patterns that make use of inheritance, including Observer, are discussed.
3. The Data Structures chapter in Part II was rewritten with the STL in mind. Both generic interfaces (as in the first edition) and STL interfaces are illustrated in the revised Chapter 7.

4. The code in Part III is based on the STL. In several places, the code is more object-oriented than before. The Huffman coding example is completely coded.
5. In Part IV, generic data structures were rewritten to be much simpler and cleaner. Additionally, as appropriate, a simplified STL implementation is illustrated at the end of the chapters in Part IV. Implemented components include `vector`, `list`, `stack`, `queue`, `set`, `map`, `priority_queue`, and various function objects and algorithms.

C++

Using C++ presents both advantages and disadvantages. The C++ class allows the separation of interface and implementation, as well as the hiding of internal details of the implementation. It cleanly supports the notion of abstraction. The advantage of C++ is that it is widely used in industry. Students perceive that the material they are learning is practical and will help them find employment, which provides motivation to persevere through the course. One disadvantage of C++ is that it is far from a perfect language pedagogically, especially in a second course, and thus additional care needs to be expended to avoid bad programming practices. A second disadvantage is that C++ is still not a stable language, so the various compilers behave differently.

It might have been preferable to write the book in a language-independent fashion, concentrating only on general principles such as the theory of the data structures and referring to C++ code only in passing, but that is impossible. C++ code is complex, and students will need to see complete examples to understand some of its finer points. As mentioned earlier, a brief review of parts of C++ is provided in Appendix A. Part I of the book describes some of C++'s more advanced features relevant to data structures.

Several parts of the language stand out as requiring special consideration: templates, inheritance, exceptions, namespaces and other recent C++ additions, and the Standard Library. I approached this material in the following manner.

- *Templates:* Templates are used extensively. Some instructors may have reservations with this approach because it complicates the code, but I included them because they are fundamental concepts in any sophisticated C++ program.
- *Inheritance:* I use inheritance relatively sparingly because it adds complications, and data structures are not a strong application area

for it. This edition contains less use of inheritance than in the previous edition. However, there is a chapter on inheritance, and part of the design patterns chapter touches on inheritance-based patterns. For the most part, instructors who are eager to avoid inheritance can do so, and those who want to discuss inheritance will find sufficient material in the text.

- *Exceptions:* Exception semantics have been standardized and exceptions seem to work on many compilers. However, exceptions in C++ involve ugly code, significant complications (e.g., if used in conjunction with templates), and probably require discussing inheritance. So I use them sparingly in this text. A brief discussion of exceptions is provided, and in some places exceptions are thrown in code when warranted. However, I generally do not attempt to catch exceptions in any Part III code (most of the Standard Library does not attempt to throw exceptions).
- *Namespaces:* Namespaces, which are a recent addition to C++, do not work correctly on a large variety of compilers. I do not attempt to use namespaces and I import the entire `std` namespace when necessary (even though not great style, it works on the largest number of compilers). Appendix A discusses the namespace issues.
- *Recent language additions:* The `bool` data type is used throughout. The new `static_cast` operator is used in preference to the old-style cast. Finally, I use `explicit` when appropriate. For the most part, other additions are not used (e.g., I generally avoid using `typename`).
- *Standard Library:* As previously mentioned, the STL is used throughout, and a safe version (that does extra bounds checking) is available online (and implemented in Part IV). We also use the `string` class and the newer `istringstream` class that are part of the standard library.

Text Organization

In this text I introduce C++ and object-oriented programming (particularly abstraction) in Part I. I discuss arrays, pointers and some other C++ topics and then go on to discuss the syntax and use of classes, templates, and inheritance. The material in these chapters was substantially rewritten. New to this edition is an entire chapter on design patterns.

In Part II I discuss Big-Oh and algorithmic paradigms, including recursion and randomization. An entire chapter is devoted to sorting, and a separate chapter contains a description of basic data structures. I use the STL in presenting the interfaces and running times of the data structures. At this

point in the text, the instructor may take several approaches to present the remaining material, including the following two.

1. Discuss the corresponding implementations (either the STL versions or the simpler versions) in Part IV as each data structure is described. The instructor can ask students to extend the classes in various ways, as suggested in the exercises.
2. Show how the STL class is used and cover implementation at a later point in the course. The case studies in Part III can be used to support this approach. As complete implementations are available on every modern C++ compiler (or on the Internet for older compilers), the instructor can use the STL in programming projects. Details on using this approach are given shortly.

Part V describes advanced data structures such as splay trees, pairing heaps, and the disjoint set data structure, which can be covered if time permits or, more likely, in a follow-up course.

Chapter-by-Chapter Text Organization

Part I consists of five chapters that describe some advanced features of C++ used throughout the text. Chapter 1 describes arrays, strings, pointers, references, and structures. Chapter 2 begins the discussion of object-oriented programming by describing the class mechanism in C++. Chapter 3 continues this discussion by examining templates, and Chapter 4 illustrates the use of inheritance. Several components, including strings and vectors, are written in these chapters. Chapter 5 discusses some basic design patterns, focusing mostly on object-based patterns such as function objects, wrappers and adapters, iterators, and pairs. Some of these patterns (most notably the wrapper pattern) are used later in the text.

Part II focuses on the basic algorithms and building blocks. In Chapter 6 a complete discussion of time complexity and Big-Oh notation is provided, and binary search is also discussed and analyzed. Chapter 7 is crucial because it covers the STL and argues intuitively what the running time of the supported operations should be for each data structure. (The implementation of these data structures, in both STL-style and a simplified version, is not provided until Part IV. The STL is available on recent compilers.) Chapter 8 describes recursion by first introducing the notion of proof by induction. It also discusses divide-and-conquer, dynamic programming, and backtracking. A section describes several recursive numerical algorithms that are used to implement the RSA cryptosystem. For many students, the material in the

second half of Chapter 8 is more suitable for a follow-up course. Chapter 9 describes, codes, and analyzes several basic sorting algorithms, including the insertion sort, Shellsort, mergesort, and quicksort, as well as indirect sorting. It also proves the classic lower bound for sorting and discusses the related problems of selection. Finally, Chapter 10 is a short chapter that discusses random numbers, including their generation and use in randomized algorithms.

Part III provides several case studies, and each chapter is organized around a general theme. Chapter 11 illustrates several important techniques by examining games. Chapter 12 discusses the use of stacks in computer languages by examining an algorithm to check for balanced symbols and the classic operator precedence parsing algorithm. Complete implementations with code are provided for both algorithms. Chapter 13 discusses the basic utilities of file compression and cross-reference generation, and provides a complete implementation of both. Chapter 14 broadly examines simulation by looking at one problem that can be viewed as a simulation and then at the more classic event-driven simulation. Finally, Chapter 15 illustrates how data structures are used to implement several shortest path algorithms efficiently for graphs.

Part IV presents the data structure implementations. Implementations that use simple protocols (`insert`, `find`, `remove` variations) are provided. In some cases, STL implementations that tend to use more complicated C++ syntax are presented. Some mathematics is used in this part, especially in Chapters 19–21, and can be skipped at the discretion of the instructor. Chapter 16 provides implementations for both stacks and queues. First these data structures are implemented using an expanding array; then they are implemented using linked lists. The STL versions are discussed at the end of the chapter. General linked lists are described in Chapter 17. Singly linked lists are illustrated with a simple protocol, and the more complex STL version that uses doubly linked lists is provided at the end of the chapter. Chapter 18 describes trees and illustrates the basic traversal schemes. Chapter 19 is a detailed chapter that provides several implementations of binary search trees. Initially, the basic binary search tree is shown, and then a binary search tree that supports order statistics is derived. AVL trees are discussed but not implemented; however, the more practical red–black trees and AA-trees are implemented. Then the STL `set` and `map` are implemented. Finally, the B-tree is examined. Chapter 20 discusses hash tables and implements the quadratic probing scheme, after examination of a simpler alternative. Chapter 21 describes the binary heap and examines heapsort and external sorting. The STL `priority_queue` is implemented in this chapter.

Part Chapter V contains material suitable for use in a more advanced course or for general reference. The algorithms are accessible even at the

first-year level; however, for completeness sophisticated mathematical analyses were included that are almost certainly beyond the reach of a first-year student. Chapter 22 describes the splay tree, which is a binary search tree that seems to perform extremely well in practice and is also competitive with the binary heap in some applications that require priority queues. Chapter 23 describes priority queues that support merging operations and provides an implementation of the pairing heap. Finally, Chapter 24 examines the classic disjoint set data structure.

The appendices contain additional C++ reference material. Appendix A describes tricky C++ issues, including some unusual operators, I/O, and recent language changes. Appendix B lists the operators and their precedence. Appendix C summarizes some C++ libraries. Appendix D describes primitive arrays and strings for those who want details of what is going on under the hood of the `vector` and `string` classes.

Chapter Dependencies

Generally speaking, most chapters are independent of each other. However, the following are some of the notable dependencies.

- *Part I:* The first three chapters should be covered in their entirety first. I recommend a brief discussion of inheritance in Chapter 4. Some instructors will want to cover all of inheritance, but it is possible to get by with just the basics of inheritance and avoid some of the more difficult C++ issues that inheritance involves. Some of the object-based patterns (e.g., wrappers and function objects) in Chapter 5 can be discussed shortly after templates, or later in the course as the need arises. Some of these patterns are used in the chapter on sorting and in Part IV.
- *Chapter 6* (*Algorithm Analysis*): This chapter should be covered prior to Chapters 7 and 9. Recursion (Chapter 8) can be covered prior to this chapter, but the instructor will have to gloss over some details about avoiding inefficient recursion.
- *Chapter 7* (*STL*): This chapter can be covered prior to, or in conjunction with, material in Part III or IV.
- *Chapter 8* (*Recursion*): The material in Sections 8.1–8.3 should be covered prior to discussing recursive sorting algorithms, trees, the tic-tac-toe case study, and shortest-path algorithms. Material such as the RSA cryptosystem, dynamic programming, and backtracking (unless tic-tac-toe is discussed) is otherwise optional.
- *Chapter 9* (*Sorting*): This chapter should follow Chapters 6 and 8. However, it is possible to cover Shellsort without Chapters 6 and 8.

Shellsort is not recursive (hence there is no need for Chapter 8), and a rigorous analysis of its running time is too complex and is not covered in the book (hence there is little need for Chapter 6).

- *Chapters 16 and 17 (Stacks/Queues/Lists)*: These chapters may be covered in either order. However, I prefer to cover Chapter 16 first, because I believe that it presents a simpler example of linked lists.
- *Chapters 18 and 19 (Trees/Search trees)*: These chapters can be covered in either order or simultaneously.

Separate Entities

The other chapters have little or no dependencies:

- *Chapter 10 (Randomization)*: The material on random numbers can be covered at any point as needed.
- *Part III (Case Studies)*: Chapters 11–15 can be covered in conjunction with, or after, the STL (in Chapter 7), and in roughly any order. There are a few references to earlier chapters. These include Section 11.2 (tic-tac-toe), which references a discussion in Section 8.7, and Section 13.2 (cross-reference generation), which references similar lexical analysis code in Section 12.1 (balanced symbol checking).
- *Chapters 20 and 21 (Hashing/Priority Queues)*: These chapters can be covered at any point.
- *Part V (Advanced Data Structures)*: The material in Chapters 22–24 is self-contained and is typically covered in a follow-up course.

Mathematics

I have attempted to provide mathematical rigor for use in CS-2 courses that emphasize theory and for follow-up courses that require more analysis. However, this material stands out from the main text in the form of separate theorems and, in some cases, separate sections (or subsections). Thus it can be skipped by instructors in courses that deemphasize theory.

In all cases, the proof of a theorem is not necessary to the understanding of the theorem's meaning. This is another illustration of the separation of an interface (the theorem statement) from its implementation (the proof). Some inherently mathematical material, such as Section 8.4 (*Numerical Applications of Recursion*), can be skipped without affecting comprehension of the rest of the chapter.

Course Organization

A crucial issue in teaching the course is deciding how the materials in Parts II–IV are to be used. The material in Part I should be covered in depth, and the student should write one or two programs that illustrate the design, implementation, and testing of classes and generic classes—and perhaps object-oriented design, using inheritance. Chapter 6 discusses Big-Oh notation. An exercise in which the student writes a short program and compares the running time with an analysis can be given to test comprehension.

In the separation approach, the key concept of Chapter 7 is that different data structures support different access schemes with different efficiency. Any case study (except the tic-tac-toe example that uses recursion) can be used to illustrate the applications of the data structures. In this way, the student can see the data structure and how it is used but not how it is efficiently implemented. This is truly a separation. Viewing things this way will greatly enhance the ability of students to think abstractly. Students can also provide simple implementations of some of the STL components (some suggestions are given in the exercises in Chapter 7) and see the difference between efficient data structure implementations in the existing STL and inefficient data structure implementations that they will write. Students can also be asked to extend the case study, but, again, they are not required to know any of the details of the data structures.

Efficient implementation of the data structures can be discussed afterward, and recursion can be introduced whenever the instructor feels it is appropriate, provided it is prior to binary search trees. The details of sorting can be discussed at any time after recursion. At this point, the course can continue by using the same case studies and experimenting with modifications to the implementations of the data structures. For instance, the student can experiment with various forms of balanced binary search trees.

Instructors who opt for a more traditional approach can simply discuss a case study in Part III after discussing a data structure implementation in Part IV. Again, the book's chapters are designed to be as independent of each other as possible.

Exercises

Exercises come in various flavors; I have provided four varieties. The basic *In Short* exercise asks a simple question or requires hand-drawn simulations of an algorithm described in the text. The *In Theory* section asks questions that either require mathematical analysis or asks for theoretically interesting solutions to problems. The *In Practice* section contains simple programming questions, including questions about syntax or particularly tricky lines of code. Finally, the *Programming Projects* section contains ideas for extended assignments.

Pedagogical Features

- Margin notes are used to highlight important topics.
- The *Objects of the Game* section lists important terms along with definitions and page references.

- The *Common Errors* section at the end of each chapter provides a list of commonly made errors.
- References for further reading are provided at the end of most chapters.

Code Availability

The code in the text is fully functional and has been tested on numerous platforms. It is available from my home page `http://www.fiu.edu/~weiss`. Be sure to browse the `README` file for information on compiler dependencies and bug fixes. The *On the Internet* section at the end of each chapter lists the filenames for the chapter's code.

Instructor's Resource Guide

An *Instructor's Guide* that illustrates several approaches to the material is available. It includes samples of test questions, assignments, and syllabi. Answers to select exercises are also provided. Instructors should contact their Addison Wesley Longman local sales representative for information on its availability or send an e-mail message to `aw.cse@awl.com`. This guide is not available for sale and is available to instructors only.

Acknowledgments

Many, many people have helped me in the preparation of this book. Many have already been acknowledged in the first edition and the related title, *Data Structures and Problem Solving Using Java*. Others, too numerous to list, have sent e-mail messages and pointed out errors or inconsistencies in explanations that I have tried to fix in this version.

For this book, I would like to thank all of the folks at Addison Wesley Longman: my editor, Susan Hartman, and associate editor, Katherine Harutunian, helped me make some difficult decisions regarding the organization of the C++ material and were very helpful in bringing this book to fruition. My copyeditor, Jerrold Moore, and proofreaders, suggested numerous rewrites that improved the text. Diana Coe did a lovely cover design. As always, Michael Hirsch has done a superb marketing job. I would especially

like to thank Pat Mahtani, my production editor, and Lynn Steines at Shepherd, Inc. for their outstanding efforts coordinating the entire project.

I also thank the reviewers, who provided valuable comments, many of which have been incorporated into the text:

Zhengxin Chen, University of Nebraska at Omaha
Arlan DeKock, University of Missouri–Rolla
Andrew Duchowski, Clemson University
Seth Copen Goldstein, Carnegie Mellon University
G. E. Hedrick, Oklahoma State University
Murali Medidi, Northern Arizona University
Chris Nevison, Colgate University
Gurpur Prabhu, Iowa State University
Donna Reese, Mississippi State University
Gurdip Singh, Kansas State University
Michael Stinson, Central Michigan University
Paul Wolfgang, Temple University

Some of the material in this text is adapted from my textbook *Efficient C Programming: A Practical Approach* (Prentice-Hall, 1995) and is used with permission of the publisher. I have included end-of-chapter references where appropriate.

My World Wide Web page, `http://www.cs.fiu.edu/~weiss`, will contain updated source code, an errata list, and a link for receiving bug reports.

M. A. W.
Miami, Florida
September, 1999