PREFACE

This book is intended for a course in data structures and algorithms. The implementation language is C++, and it is assumed that students have taken an introductory course in which that language was used. That course need not have been object-oriented, but it should have covered the fundamental statements and data types, as well as arrays and the basics of file processing.

THE STANDARD TEMPLATE LIBRARY

One of the distinctive features of this text is its reliance on the Standard Template Library—specifically, the implementation of that library provided by the Hewlett-Packard Company. There are several advantages to this approach. First, students will be working with code that has been extensively tested; they need not depend on modules created by the instructor or textbook author. Second, students will have the opportunity to study professionals' code, which is substantially more efficient—and succinct—than what students have seen before. Third, the library is available for later courses in the curriculum, and beyond!

For the most part, the library does not prescribe an implementation of these data structures. This has the advantage that we can initially focus on the services provided to users rather than on implementation details. For the definitions of these classes, we turn to the original implementation by Stepanov and others (see Stepanov and Lee, 1994) at Hewlett-Packard Research Labs. This Hewlett-Packard implementation is the basis for all implementations the author is aware of.

OTHER IMPLEMENTATIONS CONSIDERED

As important as the Hewlett-Packard implementation of the Standard Template Library is, it cannot be the exclusive focus of such a fundamental course in data structures and algorithms. Approaches that differ from those in the Hewlett-Packard implementation also deserve consideration. For example, the implementation of the list class utilizes a doubly linked list with a header node, so there is a separate section on singly linked lists and doubly linked lists with head and tail fields. There is also a discussion of the trade-offs of one design over the other. Also, there is coverage of data structures (such as graphs) and algorithms (such as backtracking) that are not yet included in the Standard Template Library.

This text also satisfies another essential need of a data structures and algorithms course: Students practice developing their own data structures. There are programming projects in which data structures are either created "from the ground up" or extended from examples in the chapters. And there are other projects to develop or extend applications that *use* the Standard Template Library.

STANDARD C++

All the code presented is based on ANSI/ISO Standard C++ and has been tested both on a Windows platform (C++Builder and Visual C++) and on a Unix platform (G++). The Standard Template Library specifications—but no particular implementation—are part of ANSI/ISO C++.

PEDAGOGICAL FEATURES

This text offers several features that may improve the teaching environment for instructors and the learning environment for students. Each chapter starts with a list of objectives and concludes with at least one major programming assignment. Each data structure is carefully described, with a precondition and postcondition for each method. In addition, most of the methods include examples of how to call the method, and the results of that call.

The details, especially of the Hewlett-Packard implementation of the Standard Template Library, are carefully investigated in the text and reinforced in a suite of 29 labs. See the "Organization of the Labs" section of this preface for more information about these labs. Each chapter has a variety of exercises, and the answers to all the exercises are available to the instructor.

SUPPORT MATERIAL

The website for all the support material is

www.mhhe.com/collins

That website has links to the following information for students:

- An overview of the labs and how to access them
- The source code for all projects developed in the text
- Applets for projects that have a strong visual component

Additionally, instructors can obtain the following from the website:

- Instructors' options with regard to the labs
- PowerPoint slides for each chapter (approximately 1500 slides)
- Answers to every chapter exercise, PowerPoint-presentation exercise, and lab experiment

SYNOPSES OF THE CHAPTERS

Chapter 1 presents those features of C++ that serve as the foundation for subsequent chapters. Most of the material reflects an object orientation: classes, inheritance, constructors, destructors, and operator overloading. There are lab experiments to review classes, as well as on inheritance and operator overloading.

Chapter 2 introduces container classes and issues related to the storage of containers. Pointers are needed both for contiguous and linked storage. As an illustration of linked storage, a singly-linked-list class is created. This oversimplified

Linked class provides a backdrop for presenting several key features of the Standard Template Library, such as templates, iterators, and generic algorithms. The associated lab experiments are on pointers, iterators, operator overloading, and generic algorithms.

Chapter 3, an introduction to software engineering, outlines the four stages of the software-development life cycle: analysis, design, implementation, and maintenance. The Unified Modeling Language is introduced as a design tool to depict inheritance, composition, and aggregation. Big-O notation, which pervades subsequent chapters, allows environment-independent estimates of the time requirements for methods. Both run-time validation, with drivers, and timing are discussed, and for each of those topics there is a follow-up lab.

Chapter 4, on recursion, represents a temporary shift in emphasis from data structures to algorithms. Backtracking is introduced, as a general technique for problem solving. And the same BackTrack class is used for searching a maze; placing eight queens on a chessboard, where none is under attack by another queen; and illustrating that a knight can traverse every square in a chessboard without landing on any square more than once. Other applications of recursion, such as for the Towers of Hanoi game and generating permutations, further highlight the elegance of recursion, especially when compared to the corresponding iterative methods. Recursion is also encountered in later chapters, notably in the implementation of Quick Sort and in the definition of binary trees. Moreover, recursion is an indispensable—even if seldom used—tool for every computing professional.

In Chapter 5, we begin our study of the Standard Template Library with the vector and deque classes. A vector is a smart array: automatically resizable, and with methods to handle insertions and deletions at any index. Furthermore, vectors are templated, so the method to insert an **int** item into a vector of **int** items is the same method used to insert a **string** item into a vector of string items. The design starts with the ***method interface***—precondition, postcondition, and method heading—of the most widely used methods in the vector class. There follows an outline of the Hewlett-Packard implementation, and further details are available in a lab. The application of the vector class, high-precision arithmetic, is essential for public-key cryptography. This application is extended in a lab and in a programming project. A deque is similar to a vector, at least from a data structures perspective. But the implementation details are considerably different, and some of these details are investigated in a lab.

Chapter 6 presents the list data structure and class, characterized by linear-time methods for inserting, removing, or retrieving at an arbitrary position. This property makes a compelling case for the use of ***list iterators:*** objects that traverse a list object and have constant-time methods for inserting, removing, or retrieving at the "current" iterator position. The doubly linked, circular implementation is introduced in this chapter, and additional details are covered in a lab. The application is a small line editor, for which list iterators are well suited. This application is extended in a programming project. There is a lab experiment on iterator categories, and another to perform a run-time comparison of vectors, deques, and lists.

The queue and stack classes are the subjects of Chapter 7. Both of these classes are ***container adaptors:*** They adapt the method interfaces of some other class. For both the queue and stack classes, the default "other" class is the deque class. The resulting method definitions for the stack and queue classes are one-liners. The specific application of queues—calculating the average waiting time at a car wash—falls into the general category of *computer simulation.* There are two applications of the stack class: the implementation of recursion, and the conversion from infix to postfix. This latter application is expanded in a lab, and forms the basis for a project on evaluating a condition.

Chapter 8 focuses on binary trees in general, and binary search trees in particular. The essential features of binary trees are presented; these are important for understanding later material on AVL trees, red-black trees, heaps, Huffman trees, and decision trees. The binary-search-tree class is a monochromatic version of the Hewlett-Packard implementation of red-black trees.

In Chapter 9, we look at AVL trees. Rotations are introduced as the mechanism by which rebalancing is accomplished. With the help of Fibonacci trees, we establish that the height of an AVL tree is always logarithmic in the number of items in the tree. The AVLTree class is not part of the Standard Template Library, but includes several important features, such as function objects; there is a follow-up lab on this difficult topic. The entire class is implemented, except for the erase method (Project 9.1). The application of AVL trees is a simple spell-checker.

Red-black trees are investigated in Chapter 10. The algorithms for inserting and deleting in a red-black tree are carefully studied, and there are associated lab experiments. Red-black trees are not in the Standard Template Library, but they are the basis for most implementations of four associative-container classes that *are* in the Standard Template Library: the set, map, multiset, and multimap classes. In a set, each item consists of a key only, and duplicate keys are not allowed. A multiset allows duplicate keys. In a map, each item has a unique key part and also another part. A multimap allows duplicate keys. There is an application to count the frequency of each word in a file, and lab experiments on the four associative-container classes.

Chapter 11 introduces the priority_queue class, which is another container adaptor. The default is the vector class, but behind the scenes there is a heap, allowing insertions in constant average time, and removal of the highest-priority element in logarithmic time, even in the worst case. Implementations that are list-based and set-based are also considered. The application is in the area of data compression, specifically, Huffman encodings: Given a text file, generate a minimal, prefix-free encoding. The project assignment is to convert the encoding back to the original text file. The lab experiment incorporates fairness into a priority queue, so that ties for the highest-priority item are always resolved in favor of the item that was on the priority queue for the longest time.

Sorting is the topic of Chapter 12. Estimates of the minimum lower bounds for comparison-based sorts are developed. Four "fast" sorts are investigated: Tree Sort (for multisets), Heap Sort (for random-access containers), Merge Sort (for lists), and

Quick Sort (for random-access containers). The chapter's lab experiment compares these sorts on randomly generated values. The project assignment is to sort a file of names and social security numbers.

Chapter 13 starts with a review of sequential and binary searching, and then investigates hashing. Currently there are no hash classes supported by either Standard C++ or the Hewlett-Packard implementation of the Standard Template Library. A hash_map class is developed. This class has method interfaces that are identical to those in the map class, except that the average time for inserting, deleting, or searching is constant instead of logarithmic! Applications include the creation and maintenance of a symbol table, and a revision of the spell-checker application from Chapter 9. There is also a comparison of chained hashing and open-address hashing; this comparison is further explored in a programming project. The speed of the hash_map class is the subject of a lab experiment.

The most general data structures—graphs, trees, and networks—are presented in Chapter 14. There are outlines of the essential algorithms: breadth-first iteration, depth-first iteration, connectedness, finding a minimum spanning tree, and finding the shortest path between two vertices. The only class developed is the (directed) network class, with an adjacency-list implementation. Other classes, such as undirected_graph and undirected_network, can be straightforwardly defined as subclasses of the network class. The Traveling Salesperson problem is investigated in a lab, and there is a programming project to complete an adjacency-matrix version of the network class. Another backtracking application is presented, with the same BackTrack class that was introduced in Chapter 4.

With each chapter, there is an associated web page that includes all programs developed in the chapter, and applets, where appropriate, to animate the concepts presented.

APPENDICES

Appendix 1 contains the background that will allow students to comprehend the mathematical aspects of the chapters. Summation notation and the rudimentary properties of logarithms are essential, and the material on mathematical induction will lead to a deeper appreciation of the analysis of binary trees and open-address hashing.

The string class is the subject of Appendix 2. This powerful class is an important part of the Standard Template Library and allows students to avoid the drudgery of character arrays.

Polymorphism, the ability of a pointer to refer to different objects in an object hierarchy, is introduced in Appendix 3. Polymorphism is an essential feature of object-oriented programming, but has been relegated to appendix status because it is not a necessary topic in an introduction to data structures and algorithms.

ORGANIZATION OF THE LABS

There are 29 website labs associated with this text. For both students and instructors, the Uniform Resource Locator (URL) is

www.mhhe.com/collins

The labs do not contain essential material, but provide reinforcement of the text material. For example, after the vector, deque, and list classes have been investigated, there is a lab to perform some timing experiments on those three classes.

The labs are self-contained, so the instructor has considerable flexibility in assigning the labs. They can be assigned as

1. Closed labs
2. Open labs
3. Ungraded homework

In addition to the obvious benefit of promoting active learning, these labs also encourage use of the scientific method. Basically, each lab is set up as an experiment. Students *observe* some phenomenon, such as the organization of the Standard Template Library's list class. They then formulate and submit a *hypothesis*—with their own code—about the phenomenon. After *testing* and, perhaps, revising their hypothesis, they submit the *conclusions* they drew from the experiment.

ACKNOWLEDGMENTS

Chun Wai Liew initiated the study of the Standard Template Library at Lafayette College, and also contributed his general expertise in C++. The following reviewers made many helpful suggestions:

Moe Bidgoli, *Saginaw Valley State University*
Scott Cannon, *Utah State University*
Jiang-Hsing Chu, *Southern Illinois University, Carbondale*
Karen C. Davis, *University of Cincinnati*
Matthew Evett, *Florida Atlantic University*
Eduardo B. Fernandez, *Florida Atlantic State University*
Sheila Foster, *California State University, Long Beach*
Mahmood Haghighi, *Bradley University*
Jack Hodges, *San Francisco State University*
Robert A. Hogue, *Youngstown State University*
Christopher Lacher, *Florida State University*
Gopal Lakhani, *Texas Tech University*
Tracy Bradley Maples, *California State University, Long Beach*

Nancy E. Miller, *Mississippi State University*
G. M. Prabhu, *Iowa State University*
Zhi-Li Zhang, *University of Minnesota*

It was a pleasure to work with the McGraw-Hill team: Emily Lupash, Betsy Jones, Jane Mohr, Lucy Mullins, and Philip Meek.

Several students from Lafayette College made important contributions. Eric Panchenko created all the applets and many of the driver programs. And Eric, along with Yi Sun and Xenia Taoubina, developed the overall format of the labs. Finally, I am indebted to all the students at Lafayette College who participated in the class testing of the book and endured earlier versions of the labs.

Bill Collins