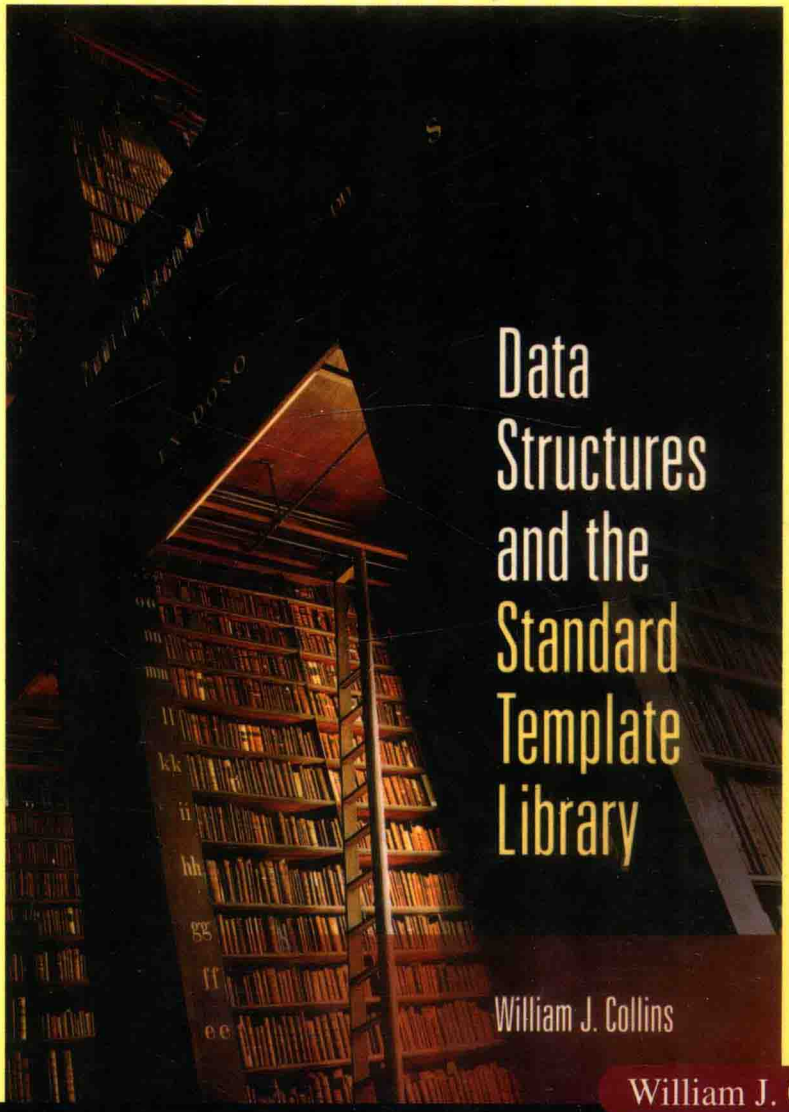
经 典 原 版 书 库

数据结构与STL

（英文版）



William J. Collins 著

机械工业出版社
China Machine Press

Mc Graw Hill Education

经 典 原 版 书 库

数据结构与STL

（英文版）

Data Structures and the Standard Template Library

William J. Collins 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2003-1005

图书在版编目（CIP）数据

数据结构与STL（英文版）/柯林斯（Collins, W. J.）著. －北京：机械工业出版社，2003.3

（经典原版书库）

书名原文：Data Structures and the Standard Template Library

ISBN 7-111-11501-5

Ⅰ.数… Ⅱ.柯… Ⅲ.数据结构－英文 Ⅳ.TP311.12

中国版本图书馆CIP数据核字（2003）第001764号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：华 章

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2003年3月第1版第1次印刷

787mm×1092mm 1/16·43.25印张

印数：0 001-3 000册

定价：69.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

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

CONTENTS