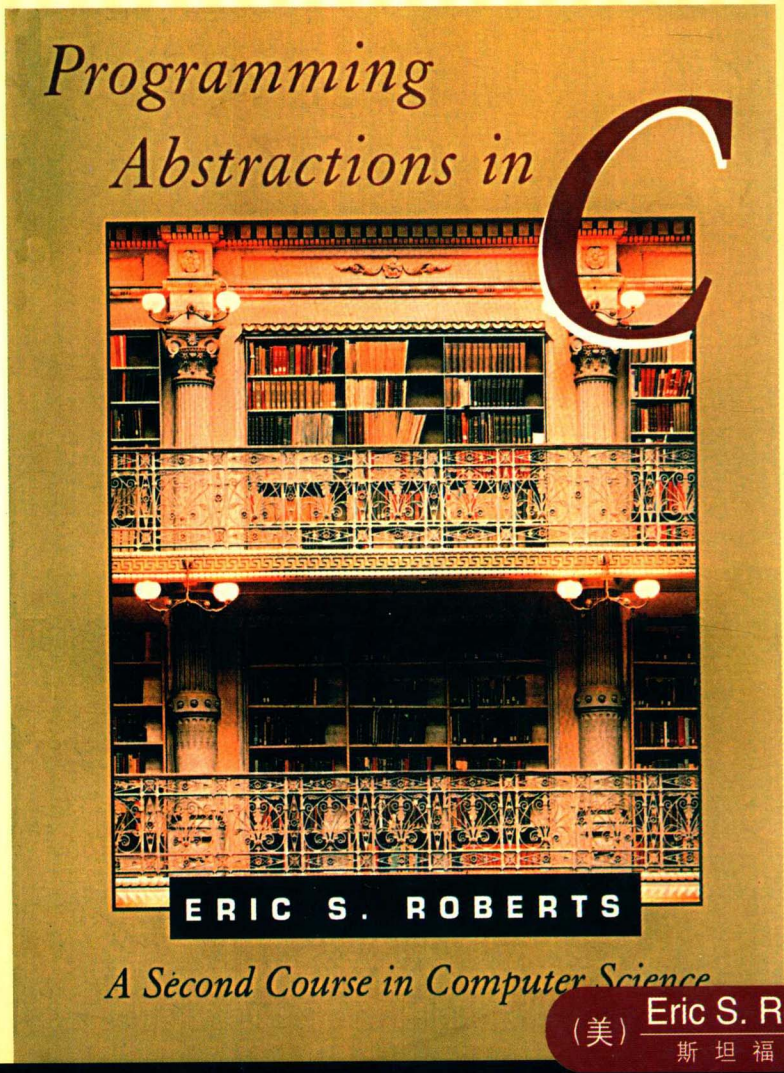
HZ BOOKS

经 典 原 版 书 库

PEARSON Addison Wesley

C程序设计的抽象思维

（英文版）



（美）Eric S. Roberts 著
斯坦福大学

机械工业出版社
China Machine Press

经 典 原 版 书 库

C程序设计的抽象思维

（英文版）

Programming Abstractions in C

A Second Course in Computer Science

（美）Eric S. Roberts 斯坦福大学 著

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2004-0182

图书在版编目（CIP）数据

C程序设计的抽象思维（英文版）/（美）罗伯茨（Roberts, E. S.）著. -北京：机械工业出版社，2004.6

（经典原版书库）

书名原文：Programming Abstractions in C: A Second Course in Computer Science

ISBN 7-111-13788-4

Ⅰ.C… Ⅱ.罗… Ⅲ.C语言-程序设计-英文 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第126644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：迟振春

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2004年6月第1版第1次印刷

787mm × 1092mm 1/16 · 52.75 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

本社购书热线：（010）68326294

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
周立柱	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		

秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

In loving memory of Rae Pober (1900–1997) for all
the joy she brought, not only to my grandmother,
but to everyone whose life she touched.

To the Student

Each year, the world of computing gets more and more exciting. Computing hardware is smaller, faster, and cheaper than ever before. The shelves of your local computer store are lined with all sorts of application programs that would have been unimaginable a decade ago. Technological innovations like the Internet and the World Wide Web are revolutionizing the way people find information, transact business, and communicate with one another. And through it all, the opportunities available to people who understand computing technology seem to grow without bounds.

The study of computer science often works in a similar way. With each new concept you learn, programming becomes increasingly exciting. You can be more creative, solve harder problems, and develop more sophisticated programs and tools. If you are reading this book, you probably have completed an introductory computer science course and understand a little about the sense of empowerment that programming conveys. But you have only scratched the surface.

Most introductory courses focus on the mechanics of programming. You learn about the syntax of a particular language and how to write simple programs in that language. The purpose of this book is to expand your horizons by introducing you to the more intellectually challenging aspects of the programming process. Programming is not about memorizing rules or writing the code for simple processes you already understand. Programming is about solving hard problems. Solving hard problems requires a lot of thought and, in most cases, a great deal of work.

You can, however, simplify the process by taking advantage of the strategies and methodologies presented in this book. Many of the concepts you will learn as you study the different parts of this text—from broad strategies like recursion to specific techniques like hashing—will enable you to solve problems that now seem completely beyond your reach. Learning those concepts will certainly be challenging. It may at times be frustrating. If you rise to the challenge and work past the frustrations, your reward at the end will be a deeper understanding of the power of computing that will create still more opportunities on the path ahead.

I wish you a pleasant journey along that road.

Eric Roberts
Department of Computer Science
Stanford University
June 1997

To the Instructor

This text is intended for use in the second programming course in a typical college or university curriculum. It covers the material in the standard CS2 course, as defined by the ACM *Curriculum '78* report and includes most of the knowledge units in the "Algorithms and Data Structures" subject area of *Computing Curriculum 1991*.

Programming Abstractions in C provides students with solid methodological skills that are consistent with the principles of modern software engineering. It builds on the foundation provided by my 1995 textbook, *The Art and Science of C*, and focuses on abstraction and interface design as central themes. Both texts use a common set of libraries that make coding in C far less complex and consequently much more accessible to the novice. These libraries have proven extremely successful with students, not only at Stanford, but at many other institutions as well. That success, however, depends on having the faculty or staff at each local institution provide libraries for the platforms the students use. The code for the libraries is available by FTP from Addison-Wesley at the following URL:

```
ftp://aw.com/aw.computer.science/Roberts.CS1.C
```

Even though I conceived of the two books as a sequence, you can easily use *Programming Abstractions in C* on its own. Part One of this book includes all the background information students might need from *The Art and Science of C*—certainly enough to understand both the examples and the overall approach taken in the rest of the book. Because the presentation in Part One is fast-paced, students should already be familiar with fundamental programming concepts at the level of an introductory course. They do not, however, need any prior exposure to C, which is covered in the first few chapters. Students who have studied *The Art and Science of C* can simply skip Part One altogether.

With the background provided by Part One, students are ready to move to new material. Part Two focuses on recursion, with an extensive set of examples that spans four chapters. In my experience, the optimal place to introduce recursion is at the beginning of the second programming course, largely for tactical reasons. Many students find recursion a difficult concept—one that requires considerable time to master. If they confront recursion at the beginning of a term, students have more time to come to grips with the concept. The placement of recursion early in this text allows you to include recursion on homework assignments and exams throughout the term. Students who do poorly on recursive problems at midterm time will be alerted to this gap in their understanding early enough to take corrective action.

If you are pressed for time in your treatment of recursion, you can omit Section 6.1 of Part Two without disturbing the flow of the presentation. Although the minimax algorithm may be too complicated for some students, it demonstrates the enormous power of recursion to solve difficult problems with a small amount of

code. Similary, the sections in Chapter 7 that cover the theoretical foundations of big-O notation and mathematical induction are not essential to the rest of the text.

Part Three has a twofold purpose. On the one hand, it introduces the principal nonrecursive types one expects to see in a data structures course, including stacks, queues, and symbol tables. On the other, this part of the text provides students with the tools they need to understand data abstraction in the context of interface-based programming. The concept that unifies the chapters in this part is the *abstract data type* or *ADT,* which is defined by its behavior rather than its representation.

One of the important features of this book is that it uses the incomplete type facility of ANSI C to define ADTs whose internal representation is completely inaccessible to the client. Because this programming style enforces the abstraction barrier, students develop the programming habits they need to write well-structured, modular code. I have also taken the position that the interfaces presented in the text should be useful tools in their own right. In most cases, students will be able to incorporate these interfaces and implementations directly into their own code.

The last chapter in Part Three, Chapter 11, introduces several important concepts, including function pointers, mapping functions, and iterators. Iterators are a relatively new addition to the Stanford course, but an extremely successful one. In our experience, the extra work required to build the iterator abstraction is more than offset by the reduction in complexity of client code.

Parts Three and Four both focus on abstract data types. To a certain extent, the division between these parts is artificial. The difference is that the ADTs in Part Four are implemented recursively while those in Part Three are not. The advantage of this organization is that Part Four plays a unifying role, drawing together the topics of recursion and ADTs in the two preceding parts.

Although the material on expression trees in Chapter 14 may be omitted without losing continuity, I have found it valuable to include this material as early as possible in the curriculum, because doing so reduces the level of mystery surrounding the operation of the C compiler and therefore helps students feel more in control about programming.

Chapter 17 does not really belong to the main body of the text but instead pushes the limits of the material toward the next set of topics that the students are likely to encounter. This final chapter focuses on object-oriented programming, using Java to illustrate the major concepts. Although some institutions have already begun to use Java in the introductory sequence, we believe that it is still makes sense to introduce the procedural approach first and move on to object-oriented programming thereafter for the following reasons:

1. Java environments are changing too rapidly to offer a stable base for teaching.
2. Students need to understand the procedural programming paradigm.
3. If you emphasize data abstraction and interfaces in the introductory courses, students are well prepared for the transition to object-oriented programming.

Our experience at Stanford convinces us that this strategy works remarkably well and allows students to adopt the object-oriented paradigm with relative ease.

Acknowledgments

Writing a textbook is never the work of a single individual. In putting this book together, I have been extremely fortunate to have the help of many talented and dedicated people. I particularly want to thank the following colleagues at Stanford, who have contributed to this project in so many different ways:

- The lecturers who have taught from draft versions of this text over the last few years, including Jerry Cain, Maggie Johnson, Bob Plummer, Mehran Sahami, and Julie Zelenski
- My teaching assistants, Stacey Doerr and Brian O'Connor, who helped refine the assignments for the course, many of which appear as exercises
- The entire team of undergraduate section leaders, who had to explain to students all the concepts that I left out of the earlier drafts
- Steve Freund and the members of the Thetis development team, who have provided a wonderful computing environment for student use
- The staff of the Education Division, most notably Claire Stager and Eddie Wallace, for keeping everything running smoothly
- The participants in my seminar on teaching introductory computer science
- The Stanford students who have taken our courses and showed us the amazing things they can accomplish

I appreciate the contributions provided by Addison-Welsey reviewers, whose comments definitely improved the structure and quality of the book:

- Phillip Barry, *University of Minnesota*
- Martin Cohn, *Brandeis University*
- Dan Ellard, *Harvard University*
- Gopal Gupta, *New Mexico*
- Phillip W. Hutto, *Emory University*
- Randall Pruim, *Boston University*
- Zhong Shao, *Yale University*

I also received extremely useful suggestions from Joe Buhler at Reed College, Pavel Curtis at PlaceWare, Inc., and Jim Mayfield at the University of Maryland, Baltimore County. Much of the work on the book took place while I was on sabbatical at Reed, and I am grateful to Joe and his colleagues in the Mathematics Department for providing such a wonderful place to work.

I want to express my gratitude to my editor, Susan Hartman, and the entire staff at Addison-Wesley—Cynthia Benn, Lynne Doran Cote, Jackie Davies, Julie Dunn, Peter Gordon, Amy Willcutt, Bob Woodbury, and Tom Ziolkowski—for their support on this book as well as its predecessor.

Most of all, I want to thank my partner Lauren Rusk, who has again worked her magic as my developmental editor. Lauren's expertise adds considerable clarity and polish to the text. Without her, nothing would ever come out nearly as well.

Contents

PART ONE
Preliminaries **1**

1 An Overview of ANSI C **3**

1.1 What is C? 4
1.2 The structure of a C program 5
Comments 7, Library inclusions 8, Program-level definitions 8, Function prototypes 9, The main program 9, Function definitions 10
1.3 Variables, values, and types 11
Variables 11, Naming conventions 12, Local and global variables 13, The concept of a data type 13, Integer types 14, Floating-point types 15, Text types 16, Boolean type 18, Simple input and output 18
1.4 Expressions 20
Precedence and associativity 21, Mixing types in an expression 22, Integer division and the remainder operator 23, Type casts 24, The assignment operator 24, Increment and decrement operators 26, Boolean operators 28
1.5 Statements 30
Simple statements 30, Blocks 30, The `if` statement 31, The `switch` statement 32, The `while` statement 34, The `for` statement 36
1.6 Functions 39
Returning results from functions 39, Function definitions and prototypes 40, The mechanics of the function-calling process 40, Stepwise refinement 41
Summary 42
Review questions 43
Programming exercises 45

2 Data Types in C **51**

2.1 Enumeration types 52
Internal representation of enumeration types 53, Scalar types 54, Understanding `typedef` 55
2.2 Data and memory 56
Bits, bytes, and words 56, Memory addresses 57
2.3 Pointers 59
Using addresses as data values 60, Declaring pointer variables 60, The fundamental pointer operations 61, The special pointer `NULL` 64, Passing parameters by reference 64

2.4 **Arrays 66**
Array declaration 69, Array selection 70, Effective and allocated sizes 71, Passing arrays as parameters 72, Initialization of arrays 72, Multidimensional arrays 75

2.5 **Pointers and arrays 77**
Pointer arithmetic 78, Incrementing and decrementing pointers 81, The relationship between pointers and arrays 82

2.6 **Records 84**
Defining a new structure type 85, Declaring structure variables 85, Record selection 86, Initializing records 86, Pointers to records 87

2.7 **Dynamic allocation 88**
The type **`void *`** 89, Coping with memory limitations 90, Dynamic arrays 91, Dynamic records 93

Summary 94

Review questions 95

Programming exercises 98

3 Libraries and Interfaces 107

3.1 **The concept of an interface 108**
Interfaces and implementations 108, Packages and abstractions 109, Principles of good interface design 110

3.2 **Random numbers 111**
The structure of the **`random.h`** interface 111, Constructing a client program 115, The ANSI functions for random numbers 117, The **`random.c`** implementation 120

3.3 **Strings 123**
The underlying representation of a string 124, The data type **`string`** 125, The ANSI string library 127, The **`strlib.h`** interface 132

3.4 **The standard I/O library 138**
Data files 138, Using files in C 139, Standard files 141, Character I/O 141, Rereading characters from an input file 142, Updating a file 142, Line-oriented I/O 145, Formatted I/O 146, The **`scanf`** functions 146

3.5 **Other ANSI libraries 148**

Summary 150

Review questions 151

Programming exercises 154

PART TWO
Recursion and Algorithmic Analysis **161**

4 Introduction to Recursion **163**

4.1 A simple example of recursion 164
4.2 The factorial function 166
The recursive formulation of `Fact` 167, Tracing the recursive process 167, The recursive leap of faith 171
4.3 The Fibonacci function 172
Computing terms in the Fibonacci sequence 173, Gaining confidence in the recursive implementation 174, Efficiency of the recursive implementation 176, Recursion is not to blame 176
4.4 Other examples of recursion 178
Detecting palindromes 179, Binary search 180, Mutual recursion 182
4.5 Thinking recursively 185
Maintaining a holistic perspective 185, Avoiding the common pitfalls 186
Summary 187
Review questions 188
Programming exercises 190

5 Recursive Procedures **195**

5.1 The Tower of Hanoi 196
Framing the problem 197, Finding a recursive strategy 198, Validating the strategy 200, Coding the solution 201, Tracing the recursive process 201
5.2 Generating permutations 206
The recursive insight 207
5.3 Graphical applications of recursion 208
The graphics library 209, An example from computer art 212, Fractals 217
Summary 222
Review questions 223
Programming exercises 224

6 Backtracking Algorithms **235**

6.1 Solving a maze by recursive backtracking 236
The right-hand rule 236, Finding a recursive approach 237, Identifying the simple cases 238, Coding the maze solution algorithm 239, Convincing yourself that the solution works 243

6.2 **Backtracking and games 245**
The game of nim 246, A generalized program for two-player games 248, The minimax strategy 254, Implementing the minimax algorithm 257, Using the general strategy to solve a specific game 259
Summary 272
Review questions 272
Programming exercises 274

7 Algorithmic Analysis 283

7.1 **The sorting problem 284**
The selection sort algorithm 285, Empirical measurements of performance 286, Analyzing the performance of selection sort 287
7.2 **Computational complexity 288**
Big-O notation 289, Standard simplifications of big-O 290, The computational complexity of selection sort 290, Predicting computational complexity from code structure 291, Worst-case versus average-case complexity 293, A formal definition of big-O 294
7.3 **Recursion to the rescue 296**
The power of divide-and-conquer strategies 296, Merging two arrays 297, The merge sort algorithm 298, The computational complexity of merge sort 300, Comparing N^2 and $N \log N$ performance 302
7.4 **Standard complexity classes 303**
7.5 **The Quicksort algorithm 306**
Partitioning the array 308, Analyzing the performance of Quicksort 311
7.6 **Mathematical induction 312**
Summary 315
Review questions 316
Programming exercises 318

PART THREE
***Data Abstraction* 325**

8 Abstract Data Types 327

8.1 **Stacks 328**
The basic stack metaphor 329, Stacks and function calls 329, Stacks and pocket calculators 330

8.2 **Defining a stack ADT 331**
Defining the types for the stack abstraction 331, Opaque types 333, Defining the `stack.h` interface 334

8.3 **Using stacks in an application 338**

8.4 **Implementing the stack abstraction 342**
Defining the concrete type 342, Implementing the stack operations 342, The advantages of opaque types 344, Improving the `stack.c` implementation 345

8.5 **Defining a scanner ADT 347**
The dangers of encapsulated state 347, Abstract data types as an alternative to encapsulated state 348, Implementing the scanner abstraction 353

Summary 358

Review questions 359

Programming exercises 360

9 Efficiency and ADTs 373

9.1 **The concept of an editor buffer 374**

9.2 **Defining the buffer abstraction 375**
Functions in the `buffer.h` interface 376, Coding the editor application 379

9.3 **Implementing the editor using arrays 380**
Defining the concrete type 381, Implementing the buffer operations 382, The computational complexity of the array implementation 385

9.4 **Implementing the editor using stacks 386**
Defining the concrete structure for the stack-based buffer 387, Implementing the buffer operations 387, Comparing computational complexities 388

9.5 **Implementing the editor using linked lists 391**
The concept of a linked list 392, Designing a linked-list data structure 393, Using a linked list to represent the buffer 394, Insertion into a linked-list buffer 396, Deletion in a linked-list buffer 398, Cursor motion in the linked-list representation 399, Linked-list idioms 402, Completing the buffer implementation 403, Computational complexity of the linked-list buffer 404, Doubly linked lists 407, Time-space tradeoffs 408

Summary 409

Review questions 410

Programming exercises 411

10 Linear Structures **419**

10.1 **Stacks revisited** 420

10.2 **Queues** 424
The structure of the `queue.h` interface 427, Array-based implementation of queues 427, Linked-list representation of queues 433

10.3 **Simulations involving queues** 436
Simulations and models 439, The waiting-line model 440, Discrete time 440, Events in simulated time 441, Implementing the simulation 442

Summary 448

Review questions 449

Programming exercises 451

11 Symbol Tables **457**

11.1 **Defining a symbol table abstraction** 458
Choosing types for values and keys 458, Representing an undefined entry 460, A preliminary version of the symbol table interface 461

11.2 **Hashing** 462
Implementing the hash table strategy 463, Choosing a hash function 468, Determining the number of buckets 470

11.3 **Limitations of the preliminary interface** 471

11.4 **Using functions as data** 473
A general plotting function 473, Declaring pointers to functions and function classes 474, Implementing `PlotFunction` 475, The `qsort` function 476

11.5 **Mapping functions** 481
Mapping over entries in a symbol table 481, Implementing `MapSymbolTable` 484, Passing client data to callback functions 485

11.6 **Iterators** 486
Using iterators 487, Defining the iterator interface 488, Implementing the iterator abstraction for symbol tables 488

11.7 **Command dispatch tables** 492

Summary 496

Review questions 497

Programming exercises 499