Preface

This book was motivated by the desire we and others have had to further the evolution of the core course in computer science. Many departments across the country have revised their curriculum in response to the introductory course in the science of computing discussed in the "Denning Report," (Denning, P. J., D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, J. Turner, and P. R. Young, "Computing as a Discipline," *Comm. ACM* **32**:1, pp. 9–23, January 1989.). That report draws attention to three working methodologies or processes — theory, abstraction, and design — as fundamental to all undergraduate programs in the discipline. More recently, the *Computing Curricula 1991* report of the joint ACM/IEEE-CS Curriculum Task Force echoes the Denning Report in identifying key recurring concepts which are fundamental to computing, especially: conceptual and formal models, efficiency, and levels of abstraction. The themes of these two reports summarize what we have tried to offer the student in this book.

This book developed from notes for a two-quarter course at Stanford — called CS109: *Introduction to Computer Science* — that serves a number of goals. The first goal is to give beginning computer science majors a solid foundation for further study. However, computing is becoming increasingly important in a much wider range of scientific and engineering disciplines. Therefore, a second goal is to give those students who will not take advanced courses in computer science the conceptual tools that the field provides. Finally, a more pervasive goal is to expose all students not only to programming concepts but also to the intellectually rich foundations of the field.

Our first version of this book was based on programming in Pascal and appeared in 1992. Our choice of Pascal as the language for example programs was motivated by that language's use in the Computer Science Advanced Placement Test as well as in a plurality of college introductions to programming. We were pleased to see that since 1992 there has been a significant trend toward C as the introductory programming language, and we accordingly developed a new version of the book using C for programming examples. Our emphasis on abstraction and encapsulation should provide a good foundation for subsequent courses covering object-oriented technology using C++.

At the same time, we decided to make two significant improvements in the content of the book. First, although it is useful to have a grounding in machine architecture to motivate how running time is measured, we found that almost all curricula separate architecture into a separate course, so the chapter on that subject was not useful. Second, many introductory courses in the theory of computing emphasize combinatorics and probability, so we decided to increase the coverage and cluster the material into a chapter of its own.

Foundations of Computer Science covers subjects that are often found split between a discrete mathematics course and a sophomore-level sequence in computer science in data structures. It has been our intention to select the mathematical foundations with an eye toward what the computer user really needs, rather than what a mathematician might choose. We have tried to integrate effectively the mathematical foundations with the computing. We thus hope to provide a better feel for the soul of computer science than might be found in a programming course,

a discrete mathematics course, or a course in a computer science subspecialty. We believe that, as time goes on, all scientists and engineers will take a foundational course similar to the one offered at Stanford upon which this book is based. Such a course in computer science should become as standard as similar courses in calculus and physics.

Prerequisites

Students taking courses based on this book have ranged from first-year undergraduates to graduate students. We assume only that students have had a solid course in programming. They should be familiar with the programming language ANSI C to use this edition. In particular, we expect students to be comfortable with C constructs such as recursive functions, structures, pointers, and operators involving pointers and structures such as dot, `->`, and `&`.

Suggested Outlines for Foundational Courses in CS

In terms of a traditional computer science curriculum, the book combines a first course in data structures — that is, a "CS2" course — with a course in discrete mathematics. We believe that the integration of these subjects is extremely desirable for two reasons:

1. It helps motivate the mathematics by relating it more closely to the computing.
2. Computing and mathematics can be mutually reinforcing. Some examples are the way recursive programming and mathematical induction are related in Chapter 2 and the way the free/bound variable distinction for logic is related to the scope of variables in programming languages in Chapter 14. Suggestions for instructive programming assignments are presented throughout the book.

There are a number of ways in which this book can be used.

A Two-Quarter or Two-Semester Course

The CS109A-B sequence at Stanford is typical of what might be done in two quarters, although these courses are rather intensive, being 4-unit, 10-week courses each. These two courses cover the entire book, the first seven chapters in CS109A and Chapters 8 through 14 in CS109B.

A One-Semester "CS2" Type Course

It is possible to use the book for a one-semester course covering a set of topics similar to what would appear in a "CS2" course. Naturally, there is too much material in the book to cover in one semester, and so we recommend the following:

1. *Recursive algorithms and programs* in Sections 2.7 and 2.8.
2. *Big-oh analysis and running time of programs*: all of Chapter 3 except for Section 3.11 on solving recurrence relations.
3. *Trees* in Sections 5.2 through 5.10.

4. *Lists*: all of Chapter 6. Some may wish to cover lists before trees, which is a more traditional treatment. We regard trees as the more fundamental notion, but there is little harm in switching the order. The only significant dependency is that Chapter 6 talks about the "dictionary" abstract data type (set with operations *insert*, *delete*, and *lookup*), which is introduced in Section 5.7 as a concept in connection with binary search trees.

5. *Sets and relations*. Data structures for sets and relations are emphasized in Sections 7.2 through 7.9 and 8.2 through 8.6.

6. *Graph algorithms* are covered in Sections 9.2 through 9.9.

A One-Semester Discrete Mathematics Course

For a one-semester course emphasizing mathematical foundations, the instructor could choose to cover:

1. *Mathematical induction and recursive programs* in Chapter 2.
2. *Big-oh analysis, running time, and recurrence relations* in Sections 3.4 through 3.11.
3. *Combinatorics* in Sections 4.2 through 4.8.
4. *Discrete probability* in Sections 4.9 through 4.13.
5. *Mathematical aspects of trees* in Sections 5.2 through 5.6.
6. *Mathematical aspects of sets* in Sections 7.2, 7.3, 7.7, 7.10, and 7.11.
7. *The algebra of relations* in Sections 8.2, 8.7, and 8.9.
8. *Graph algorithms and graph theory* in Chapter 9.
9. *Automata and regular expressions* in Chapter 10.
10. *Context-free grammars* in Sections 11.2 through 11.4.
11. *Propositional and predicate logic* in Chapters 12 and 14, respectively.

Features of This Book

To help the student assimilate the material, we have added the following study aids:

1. Each chapter has an outline section at the beginning and a summary section at the end highlighting the main points.

2. Marginal notes mark important concepts and definitions. However, items mentioned in section or subsection headlines are not repeated in the margin.

3. "Sidebars" are separated from the text by double lines. These short notes serve several purposes:

   - Some are elaborations on the text or make some fine points about program or algorithm design.

   - Others are for summary or emphasis of points made in the text nearby. These include outlines of important kinds of proofs, such as the various forms of proof by induction.

   - A few are used to give examples of *fallacious* arguments, and we hope that the separation from the text in this way will eliminate possible misconstruction of the point.

- ✦ A few give very brief introductions to major topics like undecidability or the history of computers to which we wish we could devote a full section.

4. Most of the sections end with exercises. There are more than 1000 exercises or parts spread among the sections. Of these roughly 30% are marked with a single star, which indicates that they require more thought than the unstarred exercises. Approximately another 10% of the exercises are doubly starred, and these are the most challenging.

5. Chapters end with bibliographic notes. We have not attempted to be exhaustive, but offer suggestions for more advanced texts on the subject of the chapter and mention the relevant papers with the most historical significance.

About the Cover

It is a tradition for computer science texts to have a cover with a cartoon or drawing symbolizing the content of the book. Here, we have drawn on the myth of the world as the back of a turtle, but our world is populated with representatives of some of the other, more advanced texts in computer science that this book is intended to support. They are:

The teddy bear: R. Sethi, *Programming Languages: Concepts and Constructs*, Addison-Wesley, Reading, Mass., 1989.

The baseball player: J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, Computer Science Press, New York, 1988.

The column: J. L. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative Approach*, Morgan-Kaufmann, San Mateo, Calif., 1990.

The dragon: A. V. Aho, R. Sethi, and J. D. Ullman, *Compiler Design: Principles, Techniques, and Tools*, Addison-Wesley, Reading, Mass., 1986.

The triceratops: J. L. Peterson and A. Silberschatz, *Operating Systems Concepts*, second edition, Addison-Wesley, Reading, Mass., 1985.

Acknowledgments

We are deeply indebted to a number of colleagues and students who have read this material and given us many valuable suggestions for improving the presentation. We owe a special debt of gratitude to Brian Kernighan, Don Knuth, Apostolos Lerios, and Bob Martin who read the original Pascal manuscript in detail and gave us many perceptive comments. We have received, and gratefully acknowledge, reports of course testing of the notes for the Pascal edition of this book by Michael Anderson, Margaret Johnson, Udi Manber, Joseph Naor, Prabhakar Ragde, Rocky Ross, and Shuky Sagiv.

There are a number of other people who found errors in earlier editions, both the original notes and the various printings of the Pascal edition. In this regard, we would like to thank: Susan Aho, Michael Anderson, Aaron Edsinger, Lonnie Eldridge, Todd Feldman, Steve Friedland, Christopher Fuselier, Mike Genstil, Paul Grubb III, Barry Hayes, John Hwang, Hakan Jakobsson, Arthur Keller, Dean Kelley, James Kuffner Jr., Steve Lindell, Richard Long, Mark MacDonald, Simone Martini, Hugh McGuire, Alan Morgan, Monnia Oropeza, Rodrigo Philander, Andrew Quan, Stuart Reges, John Stone, Keith Swanson, Steve Swenson, Sanjai Tiwari, Eric Traut, and Lynzi Ziegenhagen.

We acknowledge helpful advice from Geoff Clem, Jon Kettenring, and Brian Kernighan during the preparation of the C edition of *Foundations of Computer Science*.

Peter Ullman produced a number of the figures used in this book. We are grateful to Dan Clayton, Anthony Dayao, Mat Howard, and Ron Underwood for help with TEX fonts, and to Hester Glynn and Anne Smith for help with the manuscript preparation.

On-Line Access to Code, Errata, and Notes

You can obtain copies of the major programs in this book by anonymous ftp to host `ftp-cs.stanford.edu`. Login with user name `anonymous` and give your name and host as a password. You may then execute

```
cd fcsc
```

where you will find programs from this book. We also plan to keep in this directory information about errata and what course notes we can provide.

A. V. A.
Chatham, NJ

J. D. U.
Stanford, CA

July, 1994