

# Preface

---

*The last thing one discovers in writing a book  
is what to put first.*

—Blaise Pascal (1623–1662)

This book is written for the prospective computer scientist, applied mathematician, or engineer who wants to learn the ideas that underlie computer science. I have attempted to give elementary introductions to those ideas and techniques that are necessary to understand and practice the art and science of computing. The topics come from the fields of mathematics, logic, and computer science itself.

The choice of topics—and the depth and breadth of coverage—reflects the desire to provide students with the foundations needed to successfully complete courses at the upper division level in undergraduate computer science programs. The book is the outgrowth of a computer science course at Portland State University that has evolved over twenty years from a one-term course in discrete mathematics for upper-division students into a one-year course in discrete structures, logic, and computability for sophomores.

The book can be read by anyone with a good background in high school mathematics and an introductory knowledge of computer programming. Therefore it could also be used at the freshman level or at the advanced high school level. Although the book is intended for future computer scientists, applied mathematicians, or engineers, it may also be suitable for a wider audience. For example, it could be used in courses for students who intend to teach computer science.

This book differs in several ways from current books about discrete mathematics or foundations of computing. It presents an elementary and unified introduction to a collection of topics that have not been available in a single source. A major feature of the book is the unification of the material so that it doesn't fragment into a vast collection of seemingly unrelated ideas. This is accomplished by organization and focus.

The book is organized more along the lines of technique than on a subject-by-subject basis. The focus throughout the book is on the computation and construction of

objects. Therefore many traditional topics are dispersed throughout the text to places where they fit naturally with the techniques under discussion. For example, to read about properties of—and techniques for processing—natural numbers, lists, strings, graphs, or trees, it's necessary to look in the index or scan the table of contents to find the several places where they are found.

The logic coverage is much more extensive than in current books at this level. Logic is of fundamental importance in computer science not only for its use in problem solving but also for its use in formal specification of programs, formal verification of programs and for its growing use in many areas such as databases, artificial intelligence, robotics, automatic reasoning systems, and logic programming languages.

Logic is also dispersed throughout the text. For example, we introduce informal proof techniques in the first section of Chapter 1. Then we use informal logic without much comment until Chapter 4, where inductive proof techniques are presented. After the informal use of logic is well in hand, we move to the formal aspects of logic in Chapters 6 and 7, where equivalence proofs and inference-based proofs are introduced. Formal logic is applied to proving correctness properties of programs in Chapter 8, where we also introduce higher forms of logic. We introduce automatic reasoning and logic programming in Chapter 9. Logic programming is used to construct simple machine interpreters in Chapters 11, 12, and 13.

The coverage of algebraic structures in Chapter 10 differs from that in other texts. We give an elementary introduction to algebras and algebraic techniques that apply directly to computer science. In addition to the important ideas of Boolean algebra, we also introduce abstract data types as algebras, and we look at some computational algebras and a few other algebraic ideas that are directly applicable to computing problems.

In Chapters 11 through 14 the computing topics of languages, machines, and computation are presented at a new, more elementary, level.

## Some Notes

- Each chapter begins with a chapter guide, which gives a brief outline of the topics to be covered in each section.
- Each chapter ends with a chapter summary, which gives a brief description of the main ideas covered in the chapter.
- Algorithms in the text are presented in a variety of ways. Some are simply a few sentences of explanation. Others are presented in a more traditional notation. For example, we use assignment statements like  $x := t$  and control statements like **if**  $A$

**then  $B$  else  $C$  fi, while  $A$  do  $B$  od, and for  $i := 1$  to  $10$  do  $C$  od.** We avoid the use of begin-end or  $\{-\}$  pairs by using indentation. We'll also present some algorithms as logic programs after logic programming has been introduced.

- The word “proof” makes some people feel uncomfortable. It shouldn't, but it does. Maybe words like “show” or “verify” make you cringe. Most of the time we'll discuss things informally and incorporate proofs as part of the prose. At times we'll start a proof with the word “Proof,” and we'll end it with QED. QED is short for the Latin phrase *quod erat demonstrandum*, which means “which was to be proved.”
- Most problems in computer science involve one of the following five questions:

Can the problem be solved by a computer program?

If not, can you modify the problem so that it can be solved by a program?

If so, can you write a program to solve the problem?

Can you convince another person that your program is correct?

Can you convince another person that your program is efficient?

One goal of the book is that you obtain a better understanding of these questions together with a better ability to answer them. The book's ultimate goal is that you gain self-reliance and confidence in your own ability to solve problems, just like the self-reliance and confidence you have in your ability to ride a bike.

- A laboratory component for a course is a natural way to motivate and study the topics of the book. The course at Portland State University has evolved into a laboratory course. The ideal laboratory component uses an interactive, exploratory language such as Prolog or one of the various mathematical computing systems. The labs seem to work quite well when experiments are short and specific so that instant feedback is obtained when trying to solve a problem.

## Notes for the Second Edition

- Almost every section has been rewritten to clarify and update the exposition. In addition, each section now contains many subtopic headings to identify specific areas of discussion.
- Several hundred new exercises have been added to the book so that there are now over 1700 exercises. Answers are provided for about half of the exercises.
- The exercises at the end of each section are now listed by topic and ordered by difficulty within each topic. There is also a collection of proofs and/or challenges at the end of each set of exercises.
- The number of examples has been increased to make more connections between ideas

and applications.

- The last section of the book on evaluation of expressions has been dropped in favor of an introduction to complexity classes.
- Special attention has paid to the report by the ACM/IEEE Joint Task Force on Computing Curricula entitled “Computing Curricula 2001.” The book covers all topics listed in the report for the area of discrete structures (DS), which includes logic. The book also covers topics from the following areas listed in the report: recursion (PF5); basic algorithm analysis (AL1); basic computability theory (AL5); the complexity classes P and NP (AL6), automata theory (AL7); formal methods (SE9); knowledge representation and reasoning (IS3); programming paradigms (PL10).
- I hope that this edition has no errors. But I do wish to apologize in advance for any errors found in the book. They were not intentional, and I would appreciate hearing about them. As always, we should read the printed word with some skepticism.

## Using the Book

Although the book is designed as a textbook, it can also be used as a reference book because it contains the background material for many computer science topics. As with most books, there are some dependencies among the topics. For example, all parts of the book depend on the introductory material contained in Chapter 1 and Section 2.1. But you should feel free to jump into the book at whatever topic suits your fancy and then refer back to unfamiliar definitions. Here are a few more topics with associated dependencies:

- Inductive Definitions: They are used throughout the text after being introduced in Section 3.1.
- Recursively Defined Functions: They are discussed in Section 3.2 and depend somewhat on inductive definitions in Section 3.1.
- Logic: Informal proof techniques are introduced in Section 1.1. The technique of proof by induction is covered in Section 4.4, which can be read independently with only a few references back to unfamiliar definitions. Chapter 6 and Chapter 7 should be read in order. Then Chapters 8 and 9 can be read in either order.
- Analysis: Chapter 5 introduces some tools that are necessary for analyzing algorithms. It uses proof by induction, which is discussed in Section 4.4.
- Algebra: Chapter 10 uses recursively defined functions as discussed in Section 3.2, and it uses proof by induction as discussed in Section 4.4.
- Languages: Chapters 11 and 12 provide the necessary tools for courses in

programming languages and compilers and should be read in order. These chapters depend on the material in Sections 1.2, 3.3, 9.2, and 10.1.

- Computation: Chapters 13 and 14 should be read in order. There are some references to topics in Sections 9.2 and 10.1 and Chapters 11 and 12.

## Course Suggestions

The topics in the book can be presented in a variety of ways, depending on the length of the course, the emphasis, and student background. The major portion of the text has been taught for several years to sophomores at Portland State University as a year long course. Here are a few suggestions for courses of various lengths and emphases.

### Discrete mathematics with no formal logic

10-week course: 1–4, 5.1–5.3, 10.1–10.2.

15-week course: 1–5, 10, 11.1–11.2, 12.1–12.2

20-week course: 1–5, 10.1–10.4, 11.1–11.2, 12.1–12.2, 13.1, 14.3

### Discrete mathematics with some formal logic

10-week course: 1, 2.1–2.3, 3.1–3.2, 4, 5.1, 5.3, 6.1–6.2, 7.1–7.2, 10.1–10.2.

15-week course: 1, 2.1–2.3, 3, 4, 5.1–5.3, 6, 7.1–7.2, 8.1–8.2, 10.1–10.2, and one of 2.4, 5.4, 10.3, 10.4, or 10.5.

20-week course: 1–7, 8.1–8.2, 10.1–10.4, 11.1–11.2, 12.1–12.2, 13.1.

### Discrete mathematics and formal logic

10-week course: 1, 2.1–2.2, 3.1–3.2, 4, 5.1, 5.3, 6, 7.1–7.2, 10.1–10.2.

15-week course: 1, 2.1–2.3, 3, 4, 5.1–5.3, 6, 7.1–7.2, 8, 9.1, 10.1–10.2.

20-week course: 1–9, 10.1–10.3.

30-week course: 1–10, 11.1–11.2, 12.1–12.2, 13.1

### Discrete mathematics, logic, and computability

10-week course: 1, 2.1–2.2, 3.1–3.2, 4, 5.1, 5.3, 6, 7.1–7.2, 10.1–10.2.

15-week course: 1, 2.1–2.3, 3, 4, 5.1–5.3, 6, 7.1–7.2, 8, 9.1, 10.1–10.2.

20-week course: 1, 2.1–2.3, 3, 4, 5.1–5.2, 6, 7, 8.1–8.2, 10.1–10.2, 11.1–11.2, 12.1–12.2, 13.

30-week course: 1–11, 12.1–12.2, 12.4, 13–14.

### Logic

10-week course: 6.1–6.3, 7, 8.1–8.2, 9.1 with some references to 1.1–1.2, 2.1.

15-week course: 6–9 with some reference to 1.1–1.2, 2.1.

## Computability

10-week course: Review 10.1. Cover 11.1–11.3, 12.1–12.2, 13, 14.1–14.2.

15-week course: Review 10.1. Cover 11–14.

## Acknowledgments

Many people helped me create this book in the first place and many people have influenced the content and form of the second edition. Thanks go especially to the students and teachers who have kept my email quite interesting over the past several years with questions, suggestions, and criticisms. They have all influenced this book. I thank Michael Stranz at Jones and Bartlett for maintaining the entrepreneurial spirit. I also thank my family for their constant help and support.

J.L.H.  
*Portland, Oregon*