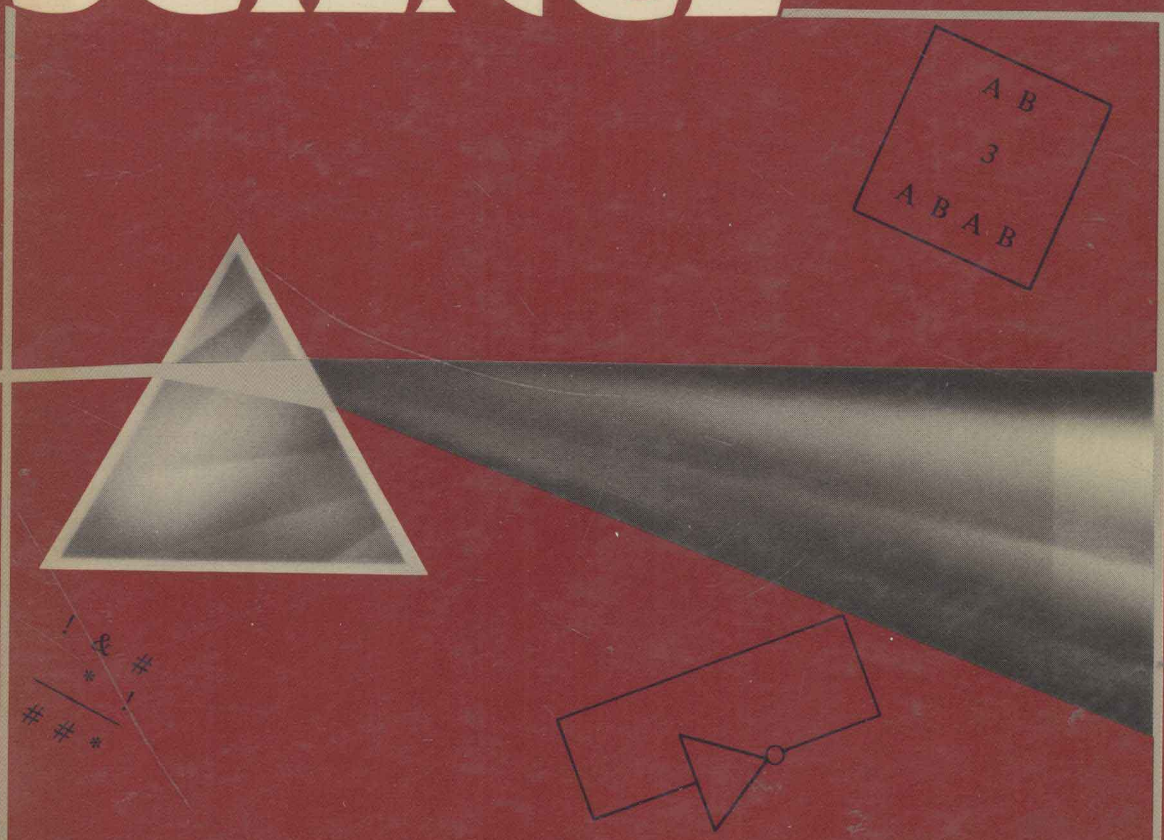
PRINCIPLES OF COMPUTER SCIENCE



CULLEN SCHAFFER

Principles of Computer Science

CULLEN SCHAFFER

Rutgers University

PRENTICE HALL, Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

SCHAFFER, CULLEN, (date)
Principles of computer science.

Includes index.
1. Computers. 2. Electronic data processing.
I. Title.
QA76.S345 1988 004 87-29061
ISBN 0-13-709759-X

Editorial/production supervision
and interior design: *Joan McCulley*
Cover design: *Photo Plus Art*
Manufacturing buyer: *Rick Washburn*

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-709759-X 025

PRENTICE-HALL INTERNATIONAL (UK) LIMITED, *London*
PRENTICE-HALL of AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL HISPANOAMERICANA, S.A., *Mexico*
PRENTICE-HALL of INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL of JAPAN, INC., *Tokyo*
SIMON & SCHUSTER ASIA PTE. LTD., *Singapore*
EDITORIA PRENTICE-HALL do BRASIL, LTDA., *Rio de Janeiro*

For Guthrie and Cecily,
whom I admire

Preface

Frankly speaking, the volume at hand is an odd one—a kind of eccentric friend whom I am glad to have had as a companion these many months, but yet am rather apprehensive in introducing to a larger acquaintance.

The book opens with a detailed account of how to build a computer from transistors, and yet it relegates external memory and I/O devices to a few pages at the end. It introduces the halting problem and uncomputability; complexity analysis and recursion; and yet it presumes virtually no mathematics, not even algebra. Finally, as if it were not enough to treat computer science topics from Boolean logic to artificial intelligence, the narrative ranges off at times into biology, philosophy, psychology, and economics.

I think it is safe to say the approach is unique. The question, of course, is whether it has any *other* merits.

One, certainly, is that it concentrates on some of the most exciting, important ideas in the history of technology. These are implicit, of course, in other introductory texts, but all too often scintillating ideas are couched in inscrutable equations, with relevance and ramifications written only between the lines.

Furthermore, the first goal of most texts is to convey *practical* information, much of which is rather less than earthshaking. Most people appreciate the utility of a keyboard; few care to read about it.

The topics treated here are of practical value, but they have been chosen primarily on grounds of intellectual significance. I have asked myself what ideas we computer scientists have reason to be proud of and then attempted to present these at an introductory level.

The result is a knockout lineup. As the deep insights of our discipline are revealed, students ought to experience not just understanding, but awe; and if I occasionally overstep the usual bounds of textbook matter and prose, it is with the idea of exciting this reaction. I would hope the tenor of the presentation is so fitted to the brilliance of the material that students will be moved to burst out from time to time into unprintable, but appreciative, expletives. Certainly, if they do not, it is the writing and not the material to blame.

* * * * *

To get at the powerful ideas of computer science, a surprisingly simple model of the computer suffices. By paring the machine down to essentials, dispensing with everything but a serviceable processor and some internal memory, I have managed to say nearly everything about how it works in a few chapters.

I expect students will get a certain satisfaction from learning what makes a computer tick, from transistors to CPU; but the project of building a computer also provides a framework into which other topics may be introduced coherently: formal logic, through Boolean algebra; the central role of feedback in self-regulating systems; and the basis of arithmetic operations in the place-value scheme.

Moreover, the ascent from transistors to logic gates, from logic gates to flip-flops, and from flip-flops to memory systems and higher—this itself is an awesome and valuable introduction to the power of hierarchical design. Indeed, one of the deep lessons of the computer, and perhaps of technology as a whole, is that the agglomeration of simple, well-defined constituents may yield disproportionate power in the whole.

The intent of the first chapters, then, is not merely to explain computer architecture, but rather, as my title would suggest, to elucidate the principles underlying it.

Once the components of a model computer have been plausibly designed, a few elementary machine-language examples lead quickly and, I should hope, shockingly to the halting problem proof of uncomputability. This naturally raises important questions about the limits of mechanical information processing. But it is equally valuable here as a testament to the abstract approach which distinguishes computer science from computer programming.

This distinction is often drawn in textbook prefaces, but I think it is rarely impressed upon students in the chapters following. To my mind, the halting problem proof is the strongest and best medicine for adept programmers, who are all too apt to treat analysis as a kind of artificial complication.

To drive the point home, the text continues with an analytic look at algorithms and data structures, considering searching and sorting problems with an emphasis on efficiency. I expect students to leave this section with an intuitive appreciation of the enormous difference between functions of various orders—log n, linear, n log n, and n^2—as well as with a practical respect for what may be gained through the creative organization and handling of information.

Again, no mathematics is presumed. Rather than treating logarithms as a prerequisite to complexity analysis, I have used complexity examples to motivate logarithms, and, with apologies to the mathematics department, I think students will be grateful for the more concrete approach.

From a general consideration of data structures, the text focuses on the special role of stacks in subroutines and recursion. These are both of major importance: subroutines because they allow us to apply hierarchical design principles to software as well as to hardware; recursion because it provides an alternative, equally powerful model of computation.

Both also suggest ways of raising the machine to a higher level of understanding, a topic with which most of the rest of the book is concerned. Translation and bootstrapping are the principal topics, but I have also included a description of a high-level language, a simplified one designed to give programmers a fresh view of the essential algorithmic elements and nonprogrammers a feel for the ease of high-level programming, unclouded by the syntactic complexities of practical languages.

With the essence of the computer bared, then, from transistors to translators, the text finishes its tour where most begin, by introducing practical, auxiliary features such as disk drives, I/O devices, and operating systems, along with some of the jargon avoided elsewhere in the text. This ties together many otherwise disparate subjects and relates the sophisticated, abstract notion of a computer developed in the text to the pedestrian contraption with which many students are already quite familiar.

From these pragmatic concerns, however, the text turns for its conclusion to the question of artificial intelligence. If the preceding chapters are not wholly uninterpretive, this last one is frankly opinionated. It will have served its purpose well if it provokes a heated response.

*   *   *   *   *

It may be discerned from this description that what I have written is not simply a collection of related chapters. This text embodies a development, an organizing scheme. It aims to show students not only the ideas of our discipline, but how these are interwoven to *form* a discipline. Realizing that I cannot hope to include all the facts in a single volume, I have striven instead to provide an overview, a framework into which students may later fit details.

This makes the book a useful one, I think, for computer science majors, who will be bombarded with such details in other courses.

For two reasons, it should also find a place in the general introductory course, which constitutes a kind of invitation to the field. First, if we concentrate on Pascal in such classes, the ranks of our majors will swell with students who believe computer science is the study of programming. Second, conversely, if the introductory course does *not* offer a substantial taste of the best computer science has to offer, we risk losing the very brightest students, the ones we pine for in upper-level classes.

Finally, although it may seem odd to some, my own intent is to use the book in a Rutgers University course meant strictly for nonmajors. Existing literacy texts stress vocabulary, programming, and the use of applications packages. The fact is, however, that students who take a one-semester introductory course are extremely unlikely to have any use for programming skills after the course has concluded. They will, of course, have occasion to use word processing and spreadsheet programs, but these have been geared to the mass market and require little teaching.

This book gives substance to my deep conviction that technical subjects merit the respect and treatment normally reserved for works in the humanities. We teach literature not for any immediate, pragmatic benefit, but because of its inherent beauty and the difficulty, sometimes, of apprehending that beauty without guidance. In the same way, I would like to try to convey something of the feeling of computer science to nontechnical students.

* * * * *

With these words, then, I send my eccentric friend out to greet the world. The text is an odd one, admittedly, but it is written with a good deal of affection and respect for the material it treats. I entrust it to the consideration of my like-hearted colleagues.

Cullen Schaffer
New Brunswick, New Jersey

Introduction

Ten years ago, a book about computers might have begun with a definition. Today, most people already know more about the machine than a definition can tell. A hefty percentage have been taught to *use* a computer in some way—either to program it or to use a store-bought program like a word processor. Many own a computer or have considered buying one.

Computers are becoming as familiar as the family dog and, if we are to believe the advertising copy we read, nearly as friendly. Would a book called *Principles of Canine Science* begin with a definition? Nonsense. Everyone knows what a dog is, and most people know quite a bit more: what dogs eat, how to teach them tricks, what the difference is between a collie and a poodle, and so on.

But, for all this knowledge, how much *do* we know?

For one thing, a dog's internal workings are more complex, by far, than our most fabulous mechanical and electronic inventions. From intricate cells to major organs like the heart and stomach, the body of a dog is a miracle of design—a miracle few people appreciate in any depth.

Moreover, knowing how a dog digests food and pumps blood is only the beginning—it still leaves us with questions on a higher level. Why does the dog sleep? What kind of control system keeps all its parts functioning smoothly together? Despite the simplicity of these questions, it is fair to say that *no one*—neither scientist nor dog lover—knows the full answer.

On a third level, consider behavior. In part, this is based on instinct. We all know that dogs have a built-in tendency to chase cats and bury bones. Charting the full range of instinctual behavior, however, is a major scientific problem. Another is explaining why dogs have the instincts they do—how these contribute to its survival.

Behavior is more than instinct, however. Intelligence permits a dog to learn and to be trained. This fact leads to theoretical questions about *how* dogs learn. It also suggests practical questions about how to teach them—to perform tricks, guide blind people, or sniff out bombs.

Finally, all these points lead to questions on a yet higher level. How has evolution made the dog what it is? Through further evolution or breeding, what will it become? How far may a dog be trained? What is the limit of canine intelligence?

* * * * *

Understanding, it seems, is a matter of many levels. When we say that everyone knows about dogs, we really mean only that all people know a little about some of the middle levels. A scientist could fill a book with what most people do *not* know about the family dog.

Moreover, there would be good reasons for reading such a book. First, subjects like physiology, evolution, and animal behavior are fascinating in their own right. The fact of life grows all the more astounding and marvelous when we examine the details.

Second, a many-leveled understanding is important for practical purposes. To breed dogs efficiently, we need to know genetics. To train them, we ought to understand canine instinct and intelligence.

Last, and perhaps most important, much of what we might learn about dogs would apply beyond the canine world. Genetics, respiration, instinct—once we have learned about these things, we may apply our knowledge to animals in general, even to ourselves. Research into how dogs fight disease can lead to advances in human medicine.

Furthermore, beyond the applications in other parts of biology, knowledge about dogs might provide insight in unrelated fields. Engineers, for example, have a great deal to learn from the dog about design and control. They have yet to build a machine that walks and climbs as easily as the dog does over uneven surfaces.

Also, the dog is a marvel of planning and compromise. At any moment, it is bombarded with information and torn by conflicting desires. A hungry, tired dog may be called by its owner just as a cat runs by. Will the dog chase, eat, sleep, heed the call? Somehow, all the information and urges must be managed and coordinated so that the dog acts sensibly.

An understanding of the dog's decision process might well profit *people* who are bombarded with information and torn by conflicting pressures: people who manage and coordinate complex systems like factories and governments.

As a last example, consider evolution. Once we have the idea and understand how it applies to dogs, we may begin to consider the evolution of economies, peoples, or ideologies. Attempting to understand dogs may lead us to principles which are of general use, far beyond the original context.

This, in fact, is our main point: it is not only the dog we are interested in, but the *principles* behind it.

* * * * *

Naturally, all we have said is relevant to computers as well as canines. When we say that most people are familiar with computers, we mean that they know a little bit about the middle levels. This is as true of most programmers as it is of the English major who knows only how to write essays with a word processor.

Most of this book is an attempt to explain the other levels and to show how they are related. That is, we want to provide a broad understanding of what computers are and how they may be used.

We will begin by showing, in detail, how a computer is constructed. It is an odd fact that, although the computer is the most complex invention in the history of humankind, we can give a fairly complete explanation of its design in just a few chapters.

Part of the reason is that the computer is designed on many levels. By saying just a little about each level it is possible to show how the whole machine operates. At first, we will discuss the simple electronic parts that go into a computer. Then we will explain how the parts are assembled into components, how the components are used to build systems and, finally, how the systems go together to make a working computer.

All these lower levels concern the physiology of the computer, its inner workings. Once we have considered these thoroughly, we will move on to higher-level questions about the computer's behavior. In particular, we will want to ask what it can and cannot do.

Given that the computer can undertake quite a variety of useful projects, we will also need to consider *practical* questions of efficiency. As we will see, computers are information-handling machines. Efficiency depends on organizing and using information effectively.

Finally, on a still higher level, we will look into the peculiar fact that computers may be used to increase their own power. This leads to some exciting, perhaps crucial questions: How powerful can computers get? Can they match our own information-handling skills? Can they think?

The reasons for studying computers at all these levels are exactly those we gave a moment ago. First, the subject material is inherently intriguing. The computer is very nearly a miracle and it becomes all the more astounding and marvelous when we examine it closely.

Second, a many-leveled understanding is important for practical purposes. Knowing how the computer is constructed often aids efficient programming. Likewise, knowing the limits of what computers can do may keep us from attempting impossible projects.

Last, and perhaps most important, much of what we learn in studying computers applies in other fields as well. In part, this is because techniques for designing and controlling complex systems carry over into other kinds of engineering.

But, as we will see, the ideas behind computers also apply in many seemingly unrelated fields: biology, economics, and, especially, pyschology.

And now we come back, again, to our main point: It is not only the computer we are interested in, but the *principles* behind it. These are what make this book not a programming guide, but an introduction to computer science. It is the idea of the computer and the analysis of that idea which forms our subject of study.

Contents