Appendices

Bibliography

This book is a little like the previews of coming attractions at the movies; it's meant to whet your appetite in several directions, without giving you the complete story about anything. To find out more, you'll have to consult more specialized books on each topic.

There are a lot of books on computer programming and computer science, and whichever I chose to list here would be out of date by the time you read this. Instead of trying to give current references in every area, in this edition I'm listing only the few most important and timeless books, plus an indication of the sources I used for each chapter.

Computer science is a fast-changing field; if you want to know what the current hot issues are, you have to read the journals. The way to start is to join the Association for Computing Machinery, 1515 Broadway, New York, NY 10036. If you are a full-time student you are eligible for a special rate for dues, which as I write this is $25 per year. (But you should write for a membership application with the current rates.) The Association publishes about 20 monthly or quarterly periodicals, plus the newsletters of about 40 Special Interest Groups in particular fields.

Read These!

If you read no other books about computer science, you must read these two. One is an introductory text for college computer science students; the other is intended for a nonspecialist audience.

Abelson, Harold, and Gerald Jay Sussman with Julie Sussman, *Structure and Interpretation of Computer Programs,* MIT Press, Second Edition, 1996.

> The introductory computer science text at MIT, this book uses Lisp as the vehicle for an intense study of everything from data structures to machine architecture. Although it's not a book about artificial intelligence as such, this is the definitive presentation of the artificial intelligence view of what computer science in general is about, and the best computer science book ever written.

Hofstadter, Douglas R., *Gödel, Escher, Bach: an Eternal Golden Braid,* Basic Books, 1979.

> This book won the Pulitzer Prize for its success in explaining to readers who aren't computer scientists some of the deepest ideas of computer science, and it makes a strong case for the view that those ideas also have a lot to teach us about human intelligence.

Chapter 1: Automata Theory

The reference I used in thinking about this chapter was

Minsky, Marvin, *Computation: Finite and Infinite Machines,* Prentice-Hall, 1967.

> Part of the interest of this particular text is that its author is a leading figure in artificial intelligence research, and so the question of whether the insights of automata theory apply also to human intelligence is always visible as a motivating force in the presentation of the theory. Minsky's bibliography will refer you to the original papers by Turing, Kleene, Church, and so on as well as some left-field references to biological information processing from people like Lettvin and McCulloch.

Chapter 2: Discrete Mathematics

This chapter touches on several topics. An overall introduction for computer scientists is

Liu, Chung Laung, *Elements of Discrete Mathematics,* McGraw-Hill, Second Edition, 1985.

> This book requires no advanced mathematical background, but it does require that the reader feel comfortable with mathematical notation and the notion of formal proof. The topics include both purely mathematical ones, like set theory, combinatorics, and modern algebra, and related computer science ones like computability, formal languages, automata theory, analysis of algorithms, and recursion. This list is not unlike the one in the book you're reading now, and in fact Professor Liu expresses a goal similar to mine: to show computer science undergraduates the relevance of mathematics to their work. The difference is that I use actual programs to illustrate the ideas whenever possible, whereas his is a "straight" math book. (Of course another difference is that he treats all these topics in much more depth. But don't be scared away; he starts simply.)

On the topic of mathematical logic, there is a range of books that vary in accessibility. Among the most pleasant are

Smullyan, Raymond, *What Is the Name of This Book?* Prentice-Hall, 1978
—, *The Lady or the Tiger?* Knopf, 1982
—, *5000 B.C. and Other Philosophical Fantasies,* St. Martin's, 1984
—, *Alice in Puzzle-Land,* Penguin, 1984.

> These are books of puzzles based on logic, but they go beyond the simple propositional inference puzzles like the one in the text. Smullyan starts with some of the classic puzzle categories, like the Liars and Truth-Tellers puzzle, and builds up to an exposition in puzzle form of topics like self-reference, modal logic, and Gödel's Theorem.

"Logic programming" is the use of mathematical logic formalisms as a programming language. It is also called "declarative programming" because instead of issuing commands to the computer, the programmer makes statements about things known to be true. The algorithm by which the programming system makes inferences from these statements is not explicitly provided by the programmer, but is built into the language.

The most widely known logic programming language, although not the only one, is Prolog. An accessible introductory text is

Ennals, Richard, *Beginning Micro-Prolog,* Harper & Row, Second Edition, 1984.

> I list this book here because it's a Prolog text and therefore relevant to mathematical logic, but for me the main interest of the book is that it argues for the use of Prolog in teaching kids, as an alternative to Logo. The book gives examples of logic programming at work in various curriculum areas.

Chapter 3: Algorithms and Data Structures

To a software engineer, the issues in this chapter are among the central ones in computer science. That's not my own way of thinking, so it's possible that my presentation doesn't give the field all the pizazz that an enthusiast would manage. To compensate for that, you should read

Bentley, Jon, *Programming Pearls,* Addison-Wesley, 1986.

> This is a collection of monthly articles written by Bentley for the *Communications* of the Association for Computing Machinery. It requires virtually no formal mathematics and is extremely readable. If the book has a moral, it's "Think first, program later." It makes its case with a number of true-to-life examples of projects in which orders of magnitude were saved in the execution time of a program by rethinking its fundamental structure.

Chapter 4: Programming Language Design

There are textbooks in "comparative programming languages," but I'm going to stick to the strategy of the chapter by using Pascal as the example. *Structure and Interpretation of Computer Programs,* one of my must-reads, will be useful as a contrast here, giving the Lisp point of view.

Jensen, Kathleen, and Niklaus Wirth, *Pascal User Manual and Report,* Springer-Verlag, Third Edition, 1985.

> This is the official report of the international committee responsible for the design of the language. The book has two parts, a reference manual and the committee report itself. The latter includes some explicit discussion of the design decisions in the language.

Chapter 5: Programming Language Implementation

I really didn't have a reference for this chapter; I just sort of forged ahead on my own! But here's the book I *should* have read first:

Friedman, Daniel P., Mitchell Wand, and Christopher T. Haynes, *Essentials of Programming Languages,* MIT Press, 1992.

> This book uses the Scheme dialect of Lisp as the basis for a study of programming language interpreters. It's harder reading than most of the books in this bibliography, but it encourages the reader to think very deeply about how programming languages work.

Chapter 6: Artificial Intelligence

I'll list two references here; one on language understanding in general and one that contains a paper about the Student program specifically.

Winograd, Terry, *Language as a Cognitive Process, Volume 1: Syntax,* Addison-Wesley, 1983.

> A planned second volume on semantics was not published. This is a technically meaty book, but considering its depth it is quite readable.

The book strikes a good balance among technical programming issues, psychological issues, and the ideas of mainstream linguistics. It includes an extensive bibliography. When I attended Terry's course at Stanford in which he first presented the material that became this book, it was the first time I experienced a course that ended with a standing ovation for the instructor. The book shows the same clarity of explanation and the same enthusiasm.

Minsky, Marvin L., *Semantic Information Processing,* MIT Press, 1969.

This is a collection of early research reports. I include it here because one of its chapters is a paper by Bobrow on STUDENT, and you won't be able to find the more complete description in Bobrow's unpublished thesis. Other chapters describe similar microworld-strategy projects of the same vintage.

Computers and People

Last but far from least, some of the most fascinating reading connected with computer science is found outside of the technical literature, in the reactions of psychologists, philosophers, and sociologists to the computer as a social force. You owe it to yourself to understand the human context of your work; you owe it to everyone else to be strongly aware of the social implications of what you do.

Turkle, Sherry, *The Second Self: Computers and the Human Spirit,* Simon and Schuster, 1984.

A sociologist's view of the computer culture, this book explores both the psychology of computer experts and the ways in which a computer-rich environment has changed the thinking of non-experts not only about technology but about what it means to be human.

Weizenbaum, Joseph, *Computer Power and Human Reason: From Judgment to Calculation,* W. H. Freeman, 1976.

Weizenbaum is a computer scientist, and this book is in part a technical argument about the limitations of what computers can do. But it is more importantly a call to computer scientists to take responsibility for the uses to which their inventions are put. Weizenbaum argues that there

are things we *shouldn't* do with computers, even if we *can* learn how to overcome the technical obstacles. Computer-based weapons of war are an obvious example, but Weizenbaum is also worried about things like automated psychotherapy, which was just a daydream when the book appeared but has since become a reality to a limited extent. Many computer scientists find this book offensive, and it is certainly possible to find flaws in the details. But the critics rarely present an alternative with an equally strong social conscience.

Dreyfus, Hubert L., *What Computers Still Can't Do: A Critique of Artificial Reason,* MIT Press, 1992.

Dreyfus is a philosopher who uses the phenomenological ideas of Heidegger and others to suggest a fundamental flaw in the assumptions AI researchers make about human intelligence. To try to sum it up in one sentence, the sort of thinking that people do in solving a puzzle is very different from the much more profound intelligence we use in carrying out our more customary activities. AI programs mimic the former but not the latter. This is a revision of an earlier book, taking into account more recent developments in AI research.

Weinberg, Gerald M., *The Psychology of Computer Programming*, Van Nostrand Reinholt, 1971.

This book studies programming as a social activity, programming as an individual activity, and the programming environment. In my opinion, its main contribution is the idea of "egoless programming," which means more or less that when your friend finds that impossible bug in your program for you, you should feel happy rather than threatened. Weinberg offers several good ideas for how to act as part of a programming community. On the other hand, I'm less enthusiastic about his manager's-eye view of the programmer as a cog in the machine, rather than as a creative artist. But overall I think this book is well worth reading; it's also entertainingly written.

Credits

Index of Defined Procedures

This index lists example procedures whose definitions are in the text. The general index lists technical terms and primitive procedures.

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

General Index

This index lists technical terms and primitive procedures. There is also an index of defined procedures, which lists procedures whose definitions are in the text.

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y