Reviews

Great Ideas in Computer Science: A Gentle Introduction, by Alan W. Biermann. 2nd edition. ISBN 0-262-52223-2, 1997. Available from MIT Press, Cambridge, MA.

This book is the second edition of Professor Biermann's 1990 text. The text supports a course in computer science for non-computer science students. This is a tall order for any teacher; just the choosing of topics is a monumental task. Perhaps that is the place to start the review.

The choice of topics is taken from "Computing as a Discipline" by Peter Denning, *et al.*[*] This article can be taken as a reasonable taxonomy of what computer science is all about. The article lists nine subareas of computer science:

Algorithms and Data Structures
Architecture
Artificial Intelligence and Robotics
Database/Information Retrieval
Human-Computer Interaction
Numerical/Symbolic Computing
Operating Systems
Programming Languages
Software Methodology and Engineering

Given a decision to cover these nine categories, there remains a question of how to cover each topic, what examples to use, what pedagogy to use, what programming to cover, *etc*. My general impression of the text is that it admirably covers the nine areas except perhaps human-computer interaction (HCI).

The text contains fifteen chapters. The first four chapters cover basic computer science programming issues. A first impression is that there is a lot of `Pascal` code in this text, but the author wants to teach elementary programming as well as computer science concepts. The language of choice is `Pascal`; the programs supporting the text are available. The choice of `Pascal` is understandable in 1985 but it is not so clear in 1997.

Certainly, `Pascal` is easier for the non-programmer to grasp than `C`. The motivating problems in these chapters are well thought out and lead the student naturally to topics in computer science. These problems come from Denning categories of algorithms, database systems, numerical algorithms, and software methodology.

Chapters five through fifteen introduce problems from the remaining topics except HCI. Since it is hard to escape HCI considerations on any PC today, the lack of an HCI chapter does not necessarily mean that HCI issues are not addressed. Several chapters (7–10) deal with the computer as an electronic device and the computer as a virtual machine (the view of the system as presented by the operating system). For the most part, the examples chosen seem natural for the intended audience.

For the remainder of this review, the comments are restricted to those I think I am sufficiently well-versed enough to comment and are germane to this journal's audience: questions relating to computational science and engineering. These comments are further constrained to touch on thought patterns or pedagogical technique.

Ch 3. Within the limits of the material, a good attempt is made to make sense of computer arithmetic. For a nontechical group, the term *error* must be handled carefully since it is a "loaded" term. The author misses a good chance to put the term in perspective by looking at $dV/dr = 500 - 3\pi r^2$ for several values of π, like 22/7, 3.14, 3.14159, *etc*.

Ch 5. This chapter is the one I disagree most with: the issue of correctness and validation is never addressed. The subject of the chapter (simulation) is covered by giving several simulations. The author skips entirely the development of the equations. The reality of the equations is never questioned. I feel this is a major pedagogical mistake.

[*]Peter J. Denning (Chairman), Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young, "Computing as a Discipline," *Communications of the ACM*, volume 32, number 1, January, 1989.

ISSN 1058-9244/97/$8
Scientific Programming, Vol. 6, pp. 399–400 (1997)

Ch 9. The author does a good job in explaining formal language notation, which can be difficult to communicate clearly. He does assume, however, that the students understand the syntax/semantics division, what parsing is, what grammars are, *etc*. My experience is that they have little understanding for the structure of natural language, let alone formal language.

Ch. 12–14. These chapters are treated as independent and I believe that is a mistake at this level.

There are occasional organization problems. For example, grammar rules are used in Chapter 1 but are not explained well until Chapter 9. In such an encyclopedic work, one should expect a few such problems.

I do have two general criticisms:

1. The text is lacking in pointers to what might be called *classical problems*. Computer science, like any endeavor, has an excitement generated by the problems and the solutions to those problems. The $\mathcal{NP}$-complete problems, parallelism, computability, programming languages, *etc.* generate their own excitement: where does the student go to find out more?
2. The text's tactic of raising questions, by using *what*-style questions, are a good device. Unfortunately, those questions are not always answered by the text. But more to the point, many chances to ask the *why is it so* question are passed up, depriving the reader of stimulation.

I recommend this text for instructors who are dealing with a high school or college level, non-computer science audience. The author has taken on a very difficult task and I believe that for a non-computer science *student* audience this text is worth serious consideration. For the non-student — say a professional scientist or engineer — it is not so clear as to the worth of the text. It certainly points out various applications of computer science ideas, but to the computer literate scientists, engineer, or mathematician, the presentation may be a too low a level to be satisfying.

D. E. Stevenson
442 R. C. Edwards Hall
Department of Computer Science
Clemson University
PO Box 341906
Clemson, SC 29634-1906, USA
E-mail: steve@cs.clemson.edu