EVALUATING INTRODUCTORY PROGRAMMING TEXTBOOKS: A GUIDE FOR STUDENTS, INSTRUCTORS, AUTHORS AND PUBLISHERS

Ben Shneiderman
Information Systems Management
University of Maryland
College Park, MD 20742

My father counseled me not to give advice to others. Inspite of that admonition, I have an irrepressible desire to make some highly subjective, potentially pompous remarks about introductory programming language textbooks.

During the past nine years I have evaluated dozens of manuscripts for eight publishers and hundreds of books for teaching term-length introductory courses in FORTRAN, BASIC, PL/I, PASCAL, COBOL and assembly languages. I have co-authored two FORTRAN texts and developed two independent study guides to programming. Each time I see a text, I make judgments by reacting to the material, rather than by comparing the material to a pre-determined set of criteria. In order to provide a "structured" review process (no computer-science oriented paper is complete without a reference to "structure"), I offer the following criteria for evaluating texts. This list is far from complete, but it is a beginning.

I. Basic Pedagogic Issues

Good sequencing of material - An acceptable text must present the material in an orderly way which is understandable to students. Easy material should be presented first, more complex material later. This obvious rule is violated by a surprising number of texts. More subtle violations are "forward references" to concepts not yet discussed and "invalid backward references" to material or terminology not described earlier. Ideas developed in Chapter X should be used in Chapter $X + 1$.

Uniform progression - Each chapter should have approximately the same amount of material of equal difficulty. If, after several easy chapters, students are hit with extremely difficult material, they may not realize the change and fail to invest additional effort, thereby producing frustration and anxiety.

Uniform technical level - Authors who attempt to capture too wide an audience produce confusing schizophrenic texts by laboriously explaining the binary number system or exponentiation in early chapters and using sophisticated numerical analysis examples or context free grammar productions in later chapters. A text should focus on a single, definable audience and be consistent in technical level.

From semantics to syntax - Even though introductory programming students concentrate on syntactic details, the text should attempt to present the semantics of operations first, then the syntax. Texts whose chapter titles are "The IF statement" or "The PERFORM statement"--instead of "Decision making" or "Program Modularity" provide the wrong emphasis. Students must be taught problem solving first - then coding.

Page-one overkill - Too many introductory texts begin with a moderate example of a program and then say "Don't worry if you don't understand all of this program, we're just trying to show a sample" make my angry. Most introductory students are a bit worried about computers and this kind of opening really scares them off. The first program should be completely comprehensible: Pring "HELLO" or sum two numbers.

Writing style - While most authors can produce grammatically correct prose, the quality of writing in introductory texts is atrocious. My favorite example is a text by one of computer science's brightest young men which used "let us note that" eitht times on one page. Other nasties are the excessive use of false connectives such as "however", "thus", "therefore", "it is clear that" and "having examined". Another bad habit is the frequent use of "many", "several", "usually", "often", "most", and "sometimes". It's shocking that algorithmically oriented thinkers can write such fuzzy prose.

Advance organizers - Educational psycholigists have clearly demonstrated the advantage of having chapter introductions which attempt to describe the forthcoming material in terms which students already know. This organizes their reading and establishes expectations for their learning. Every chapter should begin with an advance organizer.

Chapter summaries - A brief synopsis of the chapter highlights emphasizes important issues and reminds students of what facts they should have acquired. These are different from the advance organizers in style and function.

Example programs - There should be numerous complete programs in the text. It is not enough to have program fragments distributed throughout the chapter - complete programs should be shown.

Program output - Novice programmers do not know what to expect as the output of the programs in the text. You must show them explicitly. I was shocked to find that several introductory texts never showed output from the example programs that were presented.

Well motivated examples - Every example should be reasonable and have some clear application. Examples such as A = B + C or PUT LIST BRTX, TRTLLS, X567(ITH), MMMM; are useless. Every opportunity should be used to show students that programming fits in with their other courses. They will understand the examples better, pay more attention and become more motivated.

Good problems - The problems at the end of the chapter provide another opportunity to demonstrate to students that programming is relevant to other courses and that computers can be used to solve useful problems.

Embedded problems - Short simple problems should be embedded in the text. This gives students an opportunity to try out their new knowledge and it provides positive reinforcement to their learning. Answers should be provided.

Incorrect syntax should be isolated - Because of their orientation towards syntax rather than semantics, many authors do not resist the temptation to show incorrect syntax immediately after showing correct syntax. Students will tend to remember the incorrect syntax as often as the correct syntax. Incorrect syntax and other hints about debugging should be shown at the end of each chapter, if they are shown at all. French teachers teach only correct syntax and correct errors later.

II. Presentation Issues

Line printer output - authors who include output from a line printer or terminal should make sure that the material is readable. The printer should be aligned and a new ribbon inserted. If the print is still too light, arrange to have the printer overprint three times.

Typeset programs - If programs are set by the publisher, a typefont which has uniform spacing should be used. Texts which use variable spacing typefonts for showing programs are confusing to students and prevent an adequate discussion of spacing, indenting and general program format issues. If sample output is shown for programs, uniform spacing is again essential to teach proper formatting.

Line printer photo-offset - The availability of text editors has tempted many authors to produce texts on-line. Using line printer output directly as input to photo-offsetting results in an unpleasant book with unfamiliar typefonts. Authors may feel comfortable with this kind of a text, but novice students find it one more anxiety producing barrier. The extra cost and time of typesetting is worth the effort for introductory students.

Illustrations - Introductory texts should make the material appealing. One way of producing a more friendly text is to create pleasant graphics at each chapter introduction. Static photos of computer hardware are chilling, but people oriented photos or drawings can help. Use diagrams to illustrate the material. A picture is worth a thousand words.

III. Supplementary Material

Language Reference Guide - A useful feature is a thorough, precise and concise review of the language. A pedagogically appealing text may not be the best for reference purposes: by creating a reference guide to the language the book becomes useful for debugging. Backward pointers to the text might be useful.

Answers to questions and problems - A subset of the questions and problems should be answered in the back of the text. This gives the author an opportunity to include additional complete programs with output.

Comparison of compiler features - The syntactic differences and implementation details of several popular compilers for the language might be given.

Keypunch guide or terminal introduction - A quick introduction to the keypunch or to terminals might be appropriate. If the programming environment is anticipated, an introduction to the text editor would be worthwhile.

Operating system control cards - Sample job streams would help students in preparing their assignments. If the language is available through many systems, three or four of the more popular approaches could be shown.

Diagnostics - A list of diagnostic messages produced by the compiler could be given. This is essential if the messages from the compiler are good.

Index - A good index is an important asset to a text. Indexes are usually produced in the last minute when authors are sick and tired of proofing; however, effort invested at this point pays off handsomely. Find a good friend or pay somebody to assist in index preparation.