ALAN J. PERLIS

THE PROGRAMMING LANGUAGES OF THE FUTURE WILL BE STRANGER THAN WE CAN IMAGINE

ROLE OF LANGUAGE IN PROBLEM SOLVING – I
Robert Jernigan, Bruce W. Hamill, and David M. Weintraub (Editors), *The Johns Hopkins University Applied Physics Laboratory*
Published by North-Holland Publishing Co., Amsterdam, 1985, 405 pp., $55.50

This book contains the edited proceedings of a meeting whose concern was that of the book's title. The meeting, sponsored by APL, concentrated attention on computer programming languages and their roles in solving problems on computers. The specialization was crucial, and everything in the volume must be evaluated relative to it. There has already appeared in this journal an excellent review of the meeting, listing the titles and intents of the individual presentations and panel meetings (B. W. Hamill, "Symposium on the Role of Language in Problem Solving," *Johns Hopkins APL Tech. Dig.* **6**, 149-158 (1985)). Therefore, my review need not include the customary listing of contributors and summaries of their papers.

In the book, a number of important questions are raised, treated, and debated. In this reviewer's opinion, most were not clarified. It is unlikely that a reader interested in the burning issue of the conference will finish the book and say, "Aha, now I know the question and I have seen the answer." The book is not likely to become an oft-cited reference, but it is worth reading because it epitomizes the insights that most programming language people have on the topic. Furthermore, scattered throughout the book are a number of pithy nuggets about programming languages that are worth remembering.

While all languages have some common properties, natural and programming languages are profoundly different. Programming languages may ape English, but in no way, constructive or observable, are they evolving toward English or French or Hindi. The difference between programming language and natural language arises from that between sender and receiver when one is human and the other is machine: Our computers don't have enough state to capture the dynamics of our thoughts as we progress through the exercise of problem solving (let alone the heroics of dissecting emotions). Since programming languages are meant to be processed on computers, they must share the latter's limitations. Sadly, programming languages do affect the way we solve problems. One participant kept insisting that we need to make our programming languages more like natural languages. We do, but the debate should limit itself to the nature of the approximation and not confuse aping with infancy.

Programming languages are far from useless. As the book testifies eloquently, an enormous range of our thinking does find natural expression in these languages, often in ways superior to those we would have used in purely human commerce. Programming languages are evolving and improving. They are far more than notations, and we have a variety to choose from when problem solving. In many cases, the choice of language is dominated by social and economic issues more than linguistic ones. People being what they are, history and traffic have created an honest-to-goodness tribal structure partitioned by zealous worship of our own programming language. This book contains the usual chest-beating chants (expressed in intellectual terms, to be sure) asserting the superiority of my language over yours. FORTRAN, APL, LISP, Ada, COBOL, and PROLOG are all mentioned fondly, and the reader who seeks knowledge of their advantages at the problem-solving level will find the book a good source.

When we solve problems, we do so within the framework of a symbolic model that may emerge as part of the solution or, as is usually the case, may be one that is already established within the community. The latter is usually preferred because some of the work of translation can be bypassed. In his paper, Carlton-Foss illustrates some of the models employed in physics. Insofar as the computer is used in problem solving within a model, we seek programming languages that fit the model's computational needs. Physics supports so many rich models that FORTRAN has gained wide acceptance among physicists, not because it affects our thinking but because it doesn't—it is neutral and primitive. Thoughts are not communicated in FORTRAN, but the translation into FORTRAN programs of computations arising from within models is usually straightforward though tedious. There is a dark side to this "thoughtless" use of FORTRAN: It has prevented the diffusion into physics of the models arising from the complexity of computation itself. Only within the last half decade have such models become a major tool in the study of collective phenomena. Complexity itself is everywhere in science, and the computer, host to an expanding universe of communicating programs, is the natural environment for its study.

If dependency on FORTRAN is harmful, what is beneficial? In his paper, Boudreaux reveals his discovery of LISP in terms much like Balboa's on seeing the

Professor Perlis is the Eugene Higgins Professor of Computer Science at Yale University.

Pacific: He sees it as the water of a vast ocean of program possibilities so free of currents that exploration is isotropic! Of course, he is aware that 11100 (binary) years before, an entire culture set sail on that ocean in search of an artificial intelligence. Colonies have been planted and subcultures established on islands in a vast megalonesia. On visiting these islands (see the papers by Amarel, Carbonell, Rada, Rich, and Temin), a tourist is not tortured, eaten, or killed; he is rewarded with maps and maps of maps and maps of ... and then sent out to sea on a spinning top. The natives are not vicious, just wistful; it is the nature of that ocean to drown one in possibilities a mere few feet offshore of one's goal. Still, anyone who studies seriously the development of programming languages must support these explorations. LISP is the best programming language yet developed in which to absorb cognitive skills that we understand well enough to describe operationally. Artificial intelligence is the search for cognitive skills; thus, it is the richest source of stimuli for programming language evolution. The papers mentioned above are a few examples of the inevitable symbiosis between programming languages and artificial intelligence.

The reader would do well to treasure some of the nuggets scattered throughout the book. A new tool does not solve all problems; it merely frees us to concentrate on other ones (Hirshfield). What then must a student of programming know in order to structure a problem? Simply stated, a student must have an appreciation of what the machine can do (Hirshfield). Little is to be gained by building a machine that causes users to suffer the delusion that they are addressing members of their own species (Boudreaux). Though several patterns of programming language development emerge, one is quite striking: Each succeeding generation transfers one or more difficult cognitive skills from the programmer to the computer (Boudreaux).

The text of J. W. Carr's banquet address hammers home an important truth about programming languages —and everything else associated with our use and appreciation of computers: Hardware drives the field. Very-large-scale integrated circuits will have a greater effect on the future of programming languages than all the languages we have used so far. Fortunately, many of our great programming languages (APL, LISP, PROLOG, FORTRAN) have been created by a few people each, so one has reason to hope that the 64000 processor machine and gigabyte memories will stimulate the creation of new languages that process many more of our cognitive skills.

To paraphrase a great English physicist of the last century, the computer and the programming languages of the future will not only be stranger than we imagine, but stranger than we can imagine.