## Preface

*The distance is commonly very great between actual performances and speculative possibility, It is natural to suppose that as much as has been done today may be done tomorrow: but on the morrow some difficulty emerges, or some external impediment obstructs. Indolence, interruption, business, and pleasure, all take their turns of retardation; and every long work is lengthened by a thousand causes that can, and ten thousand that cannot, be recounted. Perhaps no extensive and multifarious performance was ever effected within the term originally fixed in the undertaker's mind. He that runs against Time has an antagonist not subject to casualties.* Samuel Johnson (Gibbon's *Miscellaneous Works*)

When I ran across this quote, I was at first jubilant to have found something profound about performance written by Samuel Johnson which I could use as a centerpiece for the preface to this book. But as I read I saw that he was talking much too specifically about human performance to be an appropriate general statement about performance—a statement that could be applied to the performance of a computer program. It took me a few days to see that the point Johnson made addressed the very center of what should be learned about the performance of Lisp systems by anyone who cares to study the material I've presented in this book.

That point is that people work very hard to attain every microsecond of speed that a computer demonstrates, and there are two major problems facing an implementor when he embarks on producing a Lisp system: the first problem is the myriad of decisions to be made, the interactions of various parts of the Lisp system when they are brought together, the unfortunate choice in one aspect of the system turning around and influencing, badly, the performance of another; the second problem is that writing a Lisp system is a monumental undertaking, and this undertaking is executed within the context of living a life as well. And, although an implementor might start out with large goals and spectacular intentions, the time it takes to do the thorough job required to produce an excellent Lisp system will bring many obstacles and intrusions, impediments and obstructions, and in the end, Time will have won out, in that every microsecond the implementor grabs from the hands of Time are bought with hours or days or weeks or months of effort expended by the implementor.

When I began the adventure on which I am reporting in this book, I had the belief that I would simply gather benchmark programs, distribute them to a handful of implementors, and get back the results; my major job would be to distribute the results to all interested parties. When I first sent out the initial benchmarks, there was an uproar because the benchmarks weren't fair, they weren't representative of real Lisp programs, people didn't care about performance now so why bring up this minor concern as a major one, and what was I trying to do, embarrass one group of implementors for the benefit of others?

Throughout the adventure, which lasted four years, I was praised for performing a public service, I was praised for helping find performance and correctness bugs, I was praised for taking the lead in a necessary area—gathering accurate and objective performance information—where others would fear to tread or would be too burdened to tread; and I was accused of favoritism, accused of industrial espionage, even a computer account was closed while I was using it because a system administrator was told that I was possibly gathering proprietary information.

Some people requested that this book not contain any charts, but that the benchmark results be included in running text, the idea being that it would take a significant effort on the part of a reader to make a chart of his own.

But despite the extremes of reaction to my activities, the most common reaction was for the Lisp implementor to look at the results he got running my benchmarks compared with the results from other Lisp implementations, and to turn, quietly and patiently, to his terminal to improve the results. Over the course of the four-year study I've watched the performance of some Lisp systems improve by factors of up to four on some benchmarks, and by factors of two and three overall. These results took the full four years to achieve in some cases, and I think it was the existence of a widely available, common set of benchmarks along with the results of those benchmarks for a number of Lisp implementations that have contributed to these improvements.

It is a gift to be born beautiful or rich or intelligent, to be given, by birth, the possibilities of excellent education, to be endowed with gifts that allow one to make important and stunning contributions. And I respect those individuals who use their talents when those talents have been cultivated 'in the usual manner.' But I admire, much more, people who are born ugly or poor or of average intelligence, who have minimal opportunities for first-class education, who work their

ways through bad schools and bad breaks to make contributions. Perhaps the contributions are less important or less frequent than from those who are blessed, but the contributions are the result of a strong will and advantageous application of available talent and skills.

And so it is with the performance of Lisp systems: I respect the performance of Lisp systems based on special hardware designed by wizards, but I admire the performance of Lisp systems on stock hardware and written by the common implementor, especially when I've watched the performance of those latter systems creep up over the years, by small increments, and accomplished during periods of busy activity in other areas.