EPILOGUE

It is time to take stock. Although we have touched on many aspects of computer programming in the last eight chapters, much has been left unsaid. In some cases this was due to lack of space, but most of the omissions were intentional. There are many good books on languages, algorithms and numerical methods available to those who want to learn programming in greater depth. Our goal was not to teach languages or algorithms, but to teach you to program well.

Programmers have a strong tendency to underrate the importance of good style. Eternally optimistic, we all like to think that once we throw a piece of code together, however haphazardly, it will work properly the first time and ever after. Why waste time cleaning up something that is almost certain to be correct? Besides, it probably will be used for only a few weeks.

There are really two answers to the question. The first is suggested by the word "almost." A slap-dash piece of code that falls short of perfection can be a difficult creature to deal with. The self-discipline of writing it cleanly the first time increases your chances of getting it right and eases the task of fixing it if it is not. The programmer who leaps to the coding pad or the terminal and throws a first draft at the machine spends far more time redoing and debugging than does his or her more careful colleague.

The second point is that phrase "only a few weeks." Certainly we write code differently depending on the ultimate use we expect to make of it. But computer centers are full of programs that were written for a short-term use, then were pressed into years of service. Not only pressed, but sometimes hammered and twisted. It is often simpler to modify existing code, no matter how badly written, than to reinvent the wheel yet again for a new application. Big programs — operating systems, compilers, major applications — are never written to be used once and discarded. They change and evolve. Most professional programmers spend much of their time *changing* their own and other people's code. We will say it once more — clean code is easier to maintain.

One excuse for writing an unintelligible program is that it is a private matter. Only the original programmer will ever look at it, and surely he need not spell out everything when he has it all in his head. This can be a strong argument, particularly if you don't program professionally. It is the same justification you use for writing "qt milk, fish, big box" for a grocery list instead of composing a proper sentence. If the list is intended for someone else, of course, you had better specify what kind of fish you want and what should be inside that big box. But even if only you personally want to understand the message, if it is to be readable a year from