The biggest change we made was to break the job into five small functions, each one of which can be assimilated separately, then treated as a black box that does some part of the job. Once it works, we need no longer concern ourselves with *how* it does something, only with the fact that it does. We thus have some assurance that we can deal with the program a small section at a time without much concern for the rest of the code. There is no other way to retain control of a large program.

Write and test a big program in small pieces.

Recursion represents no saving of time or storage. Somewhere in the computer must be maintained a list of all the places a recursive routine is called, so the program can eventually find its way back. But the storage for that list is shared among many different uses. More important, it is managed automatically; many of the burdens of storage management and control flow are placed on the compiler, not on the programmer. And since bookkeeping details are hidden, the program can be much easier to understand. Learning to think recursively takes some effort, but it is repaid with smaller and simpler programs.

Not every problem benefits from a recursive approach, but those that deal with data that is recursively defined often lead to very complicated programs unless the code is also recursive. A list, for example, can be said to consist of two elements, where each element is either an atom or a list. To trace through an arbitrary list requires an indefinite amount of storage to keep track of how to get back. The recursion mechanism provides this simply and concisely.

Even if you cannot use recursion in such a situation, perhaps because you must stick to Fortran, or because it is too inefficient (don't believe that until you've tried it), you will find it valuable to do the original design as a recursive program. Then unfold the recursion, simulating the recursive storage with your own explicitly indexed data structures. The resulting program should be cleaner and easier to understand than if you start from scratch.

Use recursive procedures for recursively-defined data structures.

The discipline of breaking a large job into appropriate small pieces is often called "structured design" and sometimes "composite design." Whatever you choose to call it, that discipline is necessary. To summarize some of the points made in this chapter about program structure,

(1) The only way to write and maintain a big program is as a set of small functions, subroutines, or procedures. No module should have to know much about the total problem, nor deal with more than a handful of immediate neighbors.