16 Structures, Unions, and Enumerations

Functions delay binding: data structures induce binding.
Moral: Structure data late in the programming process.

This chapter introduces three new types: structures, unions, and enumerations. A structure is a collection of values (members), possibly of different types. A union is similar to a structure, except that its members share the same storage; as a result, a union can store one member at a time, but not all members simultaneously. An enumeration is an integer type whose values are named by the programmer.

Of these three types, structures are by far the most important, so I'll devote most of the chapter to them. Section 16.1 shows how to declare structure variables and perform basic operations on them. Section 16.2 then explains how to define structure types, which—among other things—allow us to write functions that accept structure arguments or return structures. Section 16.3 explores how arrays and structures can be nested. The last two sections are devoted to unions (Section 16.4) and enumerations (Section 16.5).

16.1 Structure Variables

The only data structure we've covered so far is the array. Arrays have two important properties. First, all elements of an array have the same type. Second, to select an array element, we specify its position (as an integer subscript).

The properties of a *structure* are quite different from those of an array. The elements of a structure (its *members*, in C parlance) aren't required to have the same type. Furthermore, the members of a structure have names; to select a particular member, we specify its name, not its position.

Structures may sound familiar, since most programming languages provide a similar feature. In some languages, structures are called *records*, and members are known as *fields*.