Chapter 10

External Definitions

A C program consists of a sequence of external definitions. An external declaration becomes an external definition when it reserves storage for the object or function indicated. Within the entire program, all external declarations of the same identifier with external linkage refer to the same object or function. Within a particular translation unit, all external declarations of the same identifier with internal linkage refer to the same object or function. The syntax is shown below:

external declaration:
  function−definition
  declaration

The syntax for external definitions that are not functions is the same as the syntax for the corresponding external declarations. The syntax for the corresponding external function definition differs somewhat from that of the declaration, since the definition includes the code for the function itself.

External Function Definitions

Function definitions have the form:

function−definition:
  declaration−specifiers$_{opt}$ *declarator declaration−list*$_{opt}$
  compound statement

The form of a declarator used for a function definition can be:
  pointer$_{opt}$ *direct−declarator (parameter−type−list*$_{opt}$ *)*
  pointer$_{opt}$ *direct−declarator (identifier−list*$_{opt}$ *)*

In this syntax, the simplest instance of a direct−declarator is an identifier. (For the exact syntax, see "Declarators".)

The only storage−class specifiers allowed in a function definition are **extern** and **static**.

If the function declarator has a *parameter−type−list* (see "Declarators"), it is in function prototype form (as discussed in "Function Declarators and Prototypes"), and the function definition cannot have a *declaration−list*. Otherwise, the function declarator has a possibly empty *identifier−list*, and the *declaration−list* declares the types of the formal parameters. **register** is the only storage−class specifier permitted in declarations that are in the *declaration−list*. Any identifiers in the *identifier−list* of the function declarator that do not have their types specified in the *declaration−list* are assumed to have type **int**.

Each parameter has block scope and automatic storage duration. ANSI C and traditional C place parameters in different blocks. See "Scope" for details. Each parameter is also an *lvalue*, but since function calls in C are by value, the modification of a parameter of arithmetic type cannot affect the corresponding argument. Pointer parameters, while unmodifiable for this reason, can be used to modify the objects to which they point.

Argument promotion rules are discussed in "Function Calls".

The type of a function must be either **void** or an object type that is not an array.

External Object Definitions

A declaration of an object with file scope that has either an initializer or static linkage is an *external object definition.*

In ANSI C, a file−scope object declaration with external linkage that is declared without the storage−class specifier **extern**, and also without an initializer, results in a definition of the object at the end of the translation unit. See the discussion in "Preprocessor Changes" for more information.