Extract from **`n1256.pdf`**

1 An identifier can denote an object; a function; a tag or a member of a structure, union, or enumeration; a typedef name; a label name; a macro name; or a macro parameter. The same identifier can denote different entities at different points in the program. A member of an enumeration is called an *enumeration constant*. Macro names and macro parameters are not considered further here, because prior to the semantic phase of program translation any occurrences of macro names in the source file are replaced by the preprocessing token sequences that constitute their macro definitions.

2 For each different entity that an identifier designates, the identifier is visible (i.e., can be used) only within a region of program text called its scope. Different entities designated by the same identifier either have different scopes, or are in different name spaces. There are four kinds of scopes: function, file, block, and function prototype. (A *function prototype* is a declaration of a function that declares the types of its parameters.)

3 A label name is the only kind of identifier that has *function scope*. It can be used (in a goto statement) anywhere in the function in which it appears, and is declared implicitly by its syntactic appearance (followed by a : and a statement).

4 Every other identifier has scope determined by the placement of its declaration (in a declarator or type specifier). If the declarator or type specifier that declares the identifier appears outside of any block or list of parameters, the identifier has *file scope*, which terminates at the end of the translation unit. If the declarator or type specifier that declares the identifier appears inside a block or within the list of parameter declarations in a function definition, the identifier has *block scope*, which terminates at the end of the associated block. If the declarator or type specifier that declares the identifier appears within the list of parameter declarations in a function prototype (not part of a function definition), the identifier has *function prototype scope*, which terminates at the end of the function declarator. If an identifier designates two different entities in the same name space, the scopes might overlap. If so, the scope of one entity (the *inner scope*) will be a strict subset of the scope of the other entity (the *outer scope*). Within the inner scope, the identifier designates the entity declared in the inner scope; the entity declared in the outer scope is *hidden* (and not visible) within the inner scope.

5 Unless explicitly stated otherwise, where this International Standard uses the term identifier to refer to some entity (as opposed to the syntactic construct), it refers to the entity in the relevant name space whose declaration is visible at the point the identifier occurs.

6 Two identifiers have the *same scope* if and only if their scopes terminate at the same point.

7 Structure, union, and enumeration tags have scope that begins just after the appearance of the tag in a type specifier that declares the tag. Each enumeration constant has scope that begins just after the appearance of its defining enumerator in an enumerator list. Any other identifier has scope that begins just after the completion of its declarator. Forward references: declarations (6.7), function calls (6.5.2.2), function definitions (6.9.1), identifiers (6.4.2), name spaces of identifiers (6.2.3), macro replacement (6.10.3), source file inclusion (6.10.2), statements (6.8).