C-Language Variable Attributes

A C program consists of a set of external objects, which are either variables or functions. Those objects can be grouped in different source files or compilation units thus creating a multiple-source file program. Variables must be defined outside any function in contrast to the function arguments and variables defined inside functions, that is, they must be “external” or “global” to any function in contrast to the “internal” variables. Functions in C are always external, because C does not allow functions to be defined inside other functions.
A multiple source file program need not all be compiled at one time and precompiled functions may be loaded from libraries. Communications among the functions of a program may be carried out both trough function calls and through manipulation of external variables.

C variables have the following attributes:
identifier (name), type, size, mutability, storage class (lifetime), scope (visibility), and linkage.
The type or the value of an attribute is specified in a C program through different keywords – modifiers and specifiers.

A variable **mutability** determines whether and how the initial value of a variable can be changed in a program. Keywords: const, register, volatile, static.

An identifier **storage class** (lifetime) determines the period during which that identifier exists in memory. There are two storage classes in C – automatic and static. Automatic variables have a function or a block lifetime (local lifetime). Static variables have a program lifetime. Keywords: auto, register, extern, static,

An identifier **scope (**visibility**)** determines where the identifier can be referenced in a program or where the name can be used. There are four different kinds of scope (lexical scope): file scope, function scope (labels only), block scope, and function declaration scope (argument list only).
In multiple-source file program there is another scope associated with variables and functions with external linkage, which determines the connection between identifiers in separately compiled units. Identifier falls into several different name spaces that do not interfere with one another, that is, the same identifier may be used for different purposes even in the same scope. Example: variable names, function names, typedef names, and enum constants; tags of structures, unions and enumerations; members of each structure or union individually. Keywords: static,extern.

An identifier **linkage** determines for a multiple-source file program whether an identifier is known only in the current source file or in any source file with proper declarations.
There are two kinds of linkage: external and internal. Keywords: static, extern.

The following table summarizes the C-language variable storage class, scope and linkage attributes.

Summary of C-Language Variable Storage class, Scope, and Linkage

Specifier/ Modifier	**Storage class Lifetime**	**Linkage**	**Point of Declaration/ Definition**	**Scope (visibility)**	**How many variables with the same name in compilation unit**

`auto`
`register`
`volatile`
(or none)
	automatic (local)
	--
	block lifetime function lifetime
		internal	inside a block or function (local variable or parameter)	block or function	one for each block or function.
					--
					for nested blocks the inner variable hides the outer ones

(none)
`volatile`
`extern`
	static (global)
	--
	program lifetime
		external	outside a function
			--
			in other source files or functions must be declared as `extern`
			--
			functions are `extern` by default
				from the point of definition / declaration to the end of the file.
				--
				program must be declared `extern` in other source files
					one in a file
					many if declared `extern` in a function or in other files

`static` `volatile`	static -- program lifetime	internal	inside a block or function	block or function -- retains its value when out of scope	one in a block or function
`static` `volatile`	static (global) -- program lifetime	internal	outside a function -- can be applied to a function	from the point of definition to the end of the file.	one in a file one in a program can be use only in one file