

Re: Scope of specifier extern

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-06/msg04778.html

- *From:* Chris Torek <nospam@xxxxxxxx>
 - *Date:* 30 Jun 2006 18:31:36 GMT
-

In article <pan.2006.06.30.11.29.56.925019@xxxxxxxx>
Christian Christmann <plfriko@xxxxxxxx> wrote:

I've a a question on the specifier extern.

OK. Let me note, however, that the phrase in the subject line -- "scope of specifier" -- is not meaningful, in C. Here "specifier" obviously refers to storage-class specifiers. There are five such, in C: "auto", "extern", "register", "static", and the one oddball, "typedef". But none of them have scope; scope is a property exclusively attached to *identifiers*.

This distinction is important, as should become clear in a moment.

There are only two applicable scopes in your example: "block" and "file". (Goto labels have a third scope, "function scope", and identifiers inside prototypes have the fourth, "function prototype scope", but neither of these are used here.)

Code example:

```
void func( void )
{
extern int e;
//...
}
```

Here, the identifier "e" has *block* scope. Its scope is determined by the fact that it is declared inside a function, in the {}-pair that delimits the function's code. The keyword "extern" affects the identifier's *linkage* (and, in this case, storage duration), rather than its scope.

```
int e = 2;
```

Re: Scope of specifier extern

Here, the identifier "e" has *file* scope.

```
int main( void )
{
    func();
    return 0;
}
```

Are both the func() variable 'e' and the global 'e' same objects? Or are they distinct symbols due to different scopes (one is local, the other is global)?

The answer to that question is determined by "linkage", not "scope".

The "scope" concept answers the question "can I `see' that identifier from this position in the source code?" If an identifier is in scope, you can see it; if not, you cannot (so a reference to it gets a diagnostic, usually along the lines of "undeclared identifier".) (More precisely, it has to be in a *visible* scope, not hidden away by some intermediate shadowing declaration.) The "linkage" concept answers the question: "given two identifiers with the same name, do they name the same object or function?"

There are three possible linkages: "internal", "external", and "none". Two identifiers with the same name refer to the same object or function if they have the same linkage, except of course for the obvious case for block-scope automatic objects with "no linkage":

```
void f(void) { int i; ... }
void g(void) { int i; ... }
```

Here the two "i"s have block scope and no linkage, and are different objects.

In your original example, the identifier "e" inside func() has external linkage, and the identifier "e" at file scope also has external linkage. Since the identifiers match and the linkages match, they name the same object.

(As an aside, it may be worth noting that, in ancient K&R-1 C -- pre-C89 -- there *were* at least *some* compilers in which "extern", used inside a function, *did* give an identifier file scope.)

If you were to replace the "int e = 2;" line with "static int e = 2;" you would end up with a single translation unit in which the identifier "e" appears with both external and internal linkage. In this case, the C Standards -- C89 and C99 both -- say the effect

Re: Scope of specifier extern

is undefined.

For a much more complete treatment of scope and linkage and the storage-class specifier keywords, you might also try a google-groups search for "scope" + "linkage" + "duration" in comp.lang.c. :-)

—

In-Real-Life: Chris Torek, Wind River Systems

Salt Lake City, UT, USA (40°39.22'N, 111°50.29'W) +1 801 277 2603

email: forget about it <http://web.torek.net/torek/index.html>

Reading email is like searching for food in the garbage, thanks to spammers.

.