What happens when you want to #include the same .h file into two different .c files?   
  
Remember that, in general, each .c file in a project is passed to the compiler and compiled in isolation. So if file1.c includes shared.h and that shared.h actually generates any code or data then file1.o will include a copy. Now you compile file2.c which also includes shared.h and the output file2.o ALSO now has copies of the generated data and functions. Hopefully the linker will recognise things as being identical but the chaces are you are going to get "xxxx is mutiply defined" errors during the link.

Private declarations belong in the source file itself, not in a header. This is the only way of ensuring that no other module knows about the private declarations

I suppose you are including the header file into more than one source file, but only define (implement) the static function in one of them.   
  
If the function is static, this means that it is only visible inside the source file where it is defined, and thus it's function prototype needs not be in the header file at all. For me the primary reason for header files is tha they are "contracts" between proucers and consumers of eg functions. In your case the producer and the consumer is the same source file, so the contract can be in that file itself.   
  
You can of-course place such a prototype in a header file, but you will need a separate such heaer file for each source file that declares static functions.

Oh, are you trying to tell us that M$ sets the benchmark in good programming practices?

Well, yes, actually I do believe that. While people decry Microsoft their own source code is very good quality. The reason people call Windows "buggy" is that it does not isolate driver code enough and when Tom, Dick or Harry get a copy of the DDK and write their own webcam or scanner driver it has the ability to crash the kernel (in fact the same is true of anything that is insmod/modprobe'd into the Linux kernel too). The end user's experience is that "Windows has crashed" but more often than not it's 3rd party driver code that has crashed.   
  
In fact a lot of the coding standard we adhere too is based on the writings of Steve Maguire in Writing Solid Code:   
  
<http://www.amazon.com/Writing-Solid-Code-Microsofts-Programming/dp/1556155514>   
  
He outlines the techniques that Microsoft use internally to avoid the more obvious code faults. If you like it's a bit like a cut-down, less stringent MISRA rule book. That is a very very good book and I highly recommend it to anyone programming in C. While I've had office "purges" over the years and had to discard a lot of books to make room, this remains one of the core books (like K&R, the Borland C reference and Charles Petzold on programming Windows) that I'll retain for all time.   
  
Cliff

Jan,   
  
You may want to take a look at <avr/io.h> in AVR-LibC Wink  
  
Even \Program Files\Microsoft Visual Studio\VC98\include\windows.h is little more than a bunch of #include's   
And, as has already been stated, the technique is used a LOT in the Linux kernel tree.   
It may not be "good practice" but it's what C programmers have actually been doing in large projects since time immemorial. In fact that includes me in large in-house projects (one with 57,000 .c files) - some are "meta headers" that just group the inclusion of a bunch of other headers.   
  
Cliff