Mixing C and C++: extern C

1 MAY 2017 BY PHILLIP JOHNSTON • LAST UPDATED 20 APRIL 2022

When transitioning from C to C++, you are not likely to refresh your entire code base. Instead, you will need to maintain a mix of C and C++ code, hopefully getting the two sets of code to work together.

One common situation is that you have a C++ library with C-style interfaces. While it seems like you should be able to `#include` the headers and rely on the linker, you will find that your program fails to compile and link.

Or, taken in the opposite direction: you want to use a C function inside your C++ library. Still doesn't work!

The magic bullet here is `extern "C"`.

C and C++ use different function name mangling techniques. C++ allows function overloading, which means the linker needs to mangle the function name to indicate which specific prototype it needs to call.

By declaring a function with `extern "C"`, it changes the linkage requirements so that the C++ compiler does not add the extra mangling information to the symbol.

```
extern "C" void foo(int bar);
```

If you have a library that can be shared between C and C++, you will need to make the functions visible in the C namespace. The easiest way to accomplish this is with the following pattern:

```
#ifdef __cplusplus
extern "C"
{
#endif
```

```
//C code goes here

#ifdef __cplusplus
} // extern "C"
#endif
```

This pattern relies on the presence of the `__cplusplus` definition when using the C++ compiler. If you are using the C compiler, `extern "C"` is not used.

Are there any rules for mixing C and C++?

Of course there are rules! I recommended reviewing the C++ FAQ entry "How to mix C and C++" for a more thorough review.

To summarize the rules for mixing C and C++:

- You must use your C++ compiler when compiling main() (e.g., for static initialization)
- Your C++ compiler should direct the linking process (e.g., so it can get its special libraries)
- Your C and C++ compilers probably need to come from the same vendor and have compatible versions (e.g., so they have the same calling conventions)

The other option mentioned in the FAQ is to compile all of your code with the C++ compiler:

> *BTW there is another way to handle this whole thing: compile all your code (even your C-style code) using a C++ compiler. That pretty much eliminates the need to mix C and C++, plus it will cause you to be more careful (and possibly —hopefully!— discover some bugs) in your C-style code. The down-side is that you'll need to update your C-style code in certain ways, basically because the C++ compiler is more careful/picky than your C compiler. The point is that the effort required to clean up your C-style code may be less than the effort required to mix C and C++, and as a bonus you get cleaned up C-style code. Obviously you don't have much of a choice if you're not able to alter your C-style code (e.g., if it's from a third-party).*