Mixed Language Programming

Introduction

Make yourself familiar with source code package provided for the exercises. The given code mimics a library project "mylib", which consists of an API for Fortran (lib.f) and C/C++ (lib.h). The source code does not represent a specific application and avoids any fancy project structure (*include* or *src* subfolders, etc.) only for the sake of simplicity.

1. Compile the project by typing `make`. If something does go through, consider changing the compiler to a more recent version or to comment the code causing the issue (likely in *lib.f* and related to *FUNC_TYPE*). Execute the `test_isoc` and `test_noisoc` sample code. Determine the following information: **Compiler** : _____ **--version** : _____
2. Open *lib.f* and determine where *lib_func_void* is implemented; note down the name of the translation unit/file. **Unit/file** : _____
3. What is the effect of *FSYMBOL* (see *lib.h*)? **FSYMBOL(x)** : _____

Traditional Techniques

There are legacy techniques, which are not standardized across compilers and platforms, and certain compiler flags (symbol name configuration, etc.) may be required to get a basic level of interoperability between Fortran and C. These techniques are mainly meant to interoperate with code written prior to ISO_C_BINDING (e.g., Fortran 77). Of course, ISO_C_BINDING and legacy techniques can be combined. For example, an explicit Fortran interface may be based on ISO_C_BINDING and an implicit Fortran 77 "interface" can be just "magically" present.

1. Open the *test_noisoc.f* and *CALL lib_func_ex1()*. Implement the afore mentioned function in either C (*libimpl.c*) or C++ (*libimpl.cpp*) – the function takes no arguments and does not return a value but prints *"Hello World"* to the console. **Return type** : _____ **Fn. signature** : _____
   A function prototype declaration shall become available in *libimpl.h* (C/C++).
2. Find the implementation of *lib_f77_f2_f2f()*. **Unit/file** : _____
   Implement a similar function in a different language (compared the existing implementation). **Froeb (orig.)** : _____ **Froeb (new)** : _____
   Use a double-precision accumulator (no further thoughts about Kahan compensation) to calculate the Froebenius norm! Remember to **not** use ISO_C_BINDING.

ISO_C_BINDING

This intrinsic module standardizes binding with C code (and thereby any language settling with C conventions). Modern Fortran language features e.g., PROCEDURE POINTERs are accessible as well. Confusion about mapping structured types, passing strings, and so on are defined in a compiler/platform portable fashion.

1. Revisit the previous exercise and provide a declaration (prototype/interface) in C (*lib.h*) and Fortran (*lib.f*) using ISO_C_BINDING for the latter. The function can simply reuse the previous implementation to calculate the Froebenius norm.
2. Make yourself familiar with *lib_fiso_get()*. Open *test_isoc.f90*, declare a PROCEDURE POINTER variable, initialize this variable with "function #1", call this function, and print the result. **"func #1"(100) :** _____
3. What is a key element to enable using C function pointers in Fortran when ISO_C-BINDING is not an option? Use YES or NO

   INTEGER(8) **:** _____
   C_INTPTR_T : _____
   STOP : _____
4. Are below subroutine arguments mapped as *void** or as *void***?

   TYPE(C_PTR), INTENT(IN) :: arg1 _____
   TYPE(C_PTR), INTENT(OUT) :: arg2 _____
   TYPE(C_PTR), INTENT(IN), VALUE :: arg3 _____