Chris Rodrigues made a comment at the end of class yesterday about lack of transparency in C++ linkage to other languages, so I did some investigation this afternoon…

a few preliminary comments

- If you're interested in this topic, you might also want to read Stroustrup Section 11.3 (and possibly the rest of the chapter). Trying to get proper support when trying to write code that crosses back and forth repeatedly between C++ and C (e.g., using callbacks, etc.) is not necessarily easy; Stroustrup talks about it a bit in the section just mentioned. This note is primarily about going into C functions from C++ and vice-versa.
- Linking C++ to languages besides C and assembly is probably as tricky and painful as is linking C to those languages, since you have to change arguments (e.g., string encoding for Pascal, array information for Fortran, etc.), ensure compatible calling conventions, etc.

First, the C++ to C approach, which is easy and (I think) fairly well understood.

- By wrapping prototype definitions for global C functions in

```
extern "C" {
      // include header here, or add preprocessor-protected
      // construct to header file
}
```

  you tell the compiler to use (and generate) the same name that would be used in C for linking.
- Then you can call the function from C++.

An aside: use "-C" with Gnu's `nm` to see demangled symbol names; Gnu's `ld` demangles by default.

Now the C to C++ approach:

- Using the same construct, you can make the compiler export names for C++ global functions without the mangling used to represent namespaces and types.
- The compiler also squashes the namespace component of the mangling, so:
  - namespace doesn't matter
  - but you can't export two symbols with the same name from two namespaces (one is ok), as
  - C doesn't understand namespaces.
- If you follow the expected usage rules,
  - that is, if you declare the function in ONE file and include that file from everywhere in order to have the function signature available,
  - you retain cross-compilation-module type checking from C++ to C++.

You CANNOT make class functions look like C functions using extern "C". Instead, either

- write a global wrapper function
  - probably with your class definition, with appropriate preprocessor controls to allow inclusion for C compilation
  - you won't be able to inline, since the actual C++ class definitions can't be parsed by the C compiler
- using Gnu tools (or equivalent constructs)
  - create a symbol alias in the file that defines the function (you'll need to determine the mangled name), for example

```
namespace apple {
    class zap {...}; // declare func. bar in class zap
    // add alias to file that defines the function
    extern "C" {
        int foo (double arg) __attribute__
            ((weak, alias ("_ZN5apple3zap3fooEd")));
    }
}
```

  - you can then make any function (be careful with methods…) directly accessible from C code