Writing your first C library

If you have never written a library before, here is a tutorial for you to get started.

1. **Source files**

Source files are c files with the suffix '.c'. This file contains the actual implementation of the logic. It is as simple as that. To better maintain your code, it is good practice to segregate your source code into multiple source files.

For instance, you can put all your mathematical logic in a file called "my_math.c" and all your error handling logic in another source file "my_error_handlers.c". All your source files are compiled independently and linked after compilation.

2. **Header files**

Header files are c files that end in '.h'. Header files are not different from c files in that they both are written in the same language. Here is some important information to know regarding header files.

- Header files contain declarations of the global variables and function prototypes that are needed to be shared between multiple files. Header files are not meant to be compiled, as opposed to source files. Include these header files in whichever source file you need using the directive *#include*.
- You need to be careful about the storage class of the variables and functions that are to be declared. Remember, functions are implicitly *extern*, if you need to limit a function's visibility, use the keyword *static*.
  Similarly, variables need to be declared *extern* if you use them in multiple files.
- Be sure to use *include* guards in your headers; you will need it to prevent errors like multiple type definitions, etc. You can do this with the help of *#pragma* or the more conventional conditional compilation methods.
- Make sure your function prototype matches the function implementation/ definition.
- Be sure to apply information-hiding as and when required (such as using opaque structures).

3. Error Handling

Your code needs to handle errors well. That is, it has to detect errors, let the user know an error has occurred as well as the type of error, and also take action if necessary (such as error-correction, restarting). This is important for all kinds of software development but particularly essential for some systems (such as critical embedded systems). Use enums to standardize error codes and make them human-readable.

But more importantly, your code itself should be able to withstand all kinds of errors and error injections. If your code crashes or does something unpleasant due to errors/bugs, you may assume your code is not well written. You would ideally also like to log errors to a file.

4. Thorough testing

You need to thoroughly test your code with available tools. Perform unit testing for your functions and also test your library as an integrated module. Test for all edge and improbable cases. Take care your library is fast and memory efficient.

5. Keep pace with updates

If a large number of people are using your library, or if internal teams in your organization use it, it is an unspoken rule that you are responsible for maintenance, debugging, testing and version control. You can onboard more contributors to your library to keep the process smooth. Use git or other tools for version control.