Common Library

Cyber Solutions Development - Tactical

October 17, 2021

Abstract

Your task is to build a library as you work the other projects. Your library should implement the basic functionality of the Calc requirements in a way that is usable to fulfill the various projects on the same codebase.

1 Requirements

You must at least implement the mathematics calculation in a manner that is usuable by both the 32-bit requirement in SimpleCalc and the 64-bit requirements of the later projects.

Preferably, all of your pure functions will be implemented in this library, and your various projects will consist only on main.c files.

1.1 Basic Requirements

1. Written in C.
2. Produces both a .so shared library and .a static library.

   NOTE: this part is highly recommended to be done with CMake versus gcc in the build.sh script.
3. Different components can be built into various libraries if preferred.
4. No namespace collisions with standard libraries or common open-source libraries. (mentor dependent)
5. Used across at least SimpleCalc and two other projects.
6. Other projects should compile this library without the grader having to enter the 0_Common directory.

1.2 Tips for the Common Library

1. Read all requirements before starting this project. Attempt to save duplication of effort later on by implementing code in a way that might be reusable between projects.
2. Utilize CMake's add_subdirectory(), allowing the projects to be built in the root project directory or their individual directories. This can help with the requirement to have other projects build the library.
3. If you are finding the projects to be easy, look into implementing unit testing. (see notes to grader)
4. Look up common open-source libraries (ie OpenGL, SQLite, SDL, OpenSSL, etc). See how they do things, and look for other projects that build against them.

2 Deliverables

Your code should have the following file structure:

```
0_Common
 |__ src
 |    |__ source-files
 |__ include
 |    |__ header-files
 |__ references
 |    |__ documentation
 |__ CMakeLists.txt
```

Your code should build and compile with the following shell script ran from the Common directory. An example of this would be:

```
// build.sh
mkdir build
cd build
cmake ..
make
```

You are free to make this as fancy as you want, but your build script needs to:

1. be called build.sh
2. run properly when used with the command `bash ./build.sh`

   NOTE: this does not conflict with the requirement to have other projects build this library. It should be build from both this directory and those projects.
3. live in the base folder of your project (ie 0_Common/build.sh)
4. do whatever steps needed to produce a binary named 0_Common/build/simplecalc

3 Notes to grader

The purpose of this assignment is NOT to write fancy C code. Use this assignment to achieve the following objectives:

1. Hammer down on the coding standard.
2. Building projects with CMake. Have the trainee utilize CMake with best practices to solve the intent of the requirements for the library.
3. Proper error handling (resource cleanup, printing errors to stderr, etc)
4. Not implementing any code that shouldn't be in a library. Make sure they follow conventions for stdout versus stderr and think about how the code might be used by another project.
5. Discuss unit testing with the trainee. Have them use a testing library (cmocka, cunit, gtest, cppunit, etc.) to unit test at least one function in the library.
6. Discuss best practices and how producing code libraries can save effort.

Although you ultimately control the trainees JQR completion status, allow them to complete the projects using whichever means they want (within reason). Engineers need to be able to break down problems, and iterate on solutions. If we say "you will do X and Y and Z to solve this" we risk putting the trainees in a box where they can't fully flex their problem-solving muscles.

4 JQR Sections Potentially Covered - Mentor/Trainee dependent

1. These projects are designed to give each trainee room to think creatively. Although the reference specification may target specific JQR requirements, it is up to the trainee to pick which JQR items they want to solve, and it should be indicated somewhere in the trainees documentation. It is up to the mentor to determine which JQRs have been satisfied and which need more work.

2. These JQRs are satisfied in the reference solution, which is designed to be a sane and straightforward solve, but not particularly creative.

This section will vary widely from trainee to trainee. Refer to project specs for line items as projects are completed.