JQR Calc Projects

Cyber Solutions Development - Tactical

October 20, 2021

Abstract

This series of projects is designed to prepare CCD Trainees to take the BSLE. They may also be used to demonstrate many of the Basic JQR Line Items. They are challenging, but not needlessly difficult, and should prove to be a good exercise for developers of all levels.

1 Orientation

This repository is designed to provide structure, a testing framework, and some basic examples for CCD Trainees. At the time of this writing, there are 5 total projects. These 5 are the primary BSLE preparation projects. This means that, if done properly with guidance from a senior mentor, a trainee should have a strong foundation going into the exam. You will most likely have to complete a small project afterwards to complete the remainder of the JQR demonstrate requirements, and may be used by a mentor as a way to remediate any weak areas identified via the BLSE.

The primary projects in order are:

- SimpleCalc - Designed as C familiarization, as well as to help the mentor set the tone and expectations (style guide adherence, for example) for the remaining projects.
- FileCalc - Introduces file IO concepts, basic serialization, and binary specifications
- DataStructures1 - The (mostly) catch all data structures section. Some of these are needed for the more complex problems.
- ThreadCalc - Introduces threading and threadpool concepts.
- NetCalc - Introduces networking and socket IO concepts. Also serves as the "capstone" before taking the BSLE.

These are considered "core competency" projects and are designed to train the more common tasks the trainee will encounter while working on the BSLE and real-world projects. While the remaining line items are important, Projects 1-5 should be considered fundamental, and should not be skipped or lightly passed over.

In addition to the Project Folders, which are labeled "X_Project", there is also:

- 0_Common - This is more of a "catch-all" project. While it does not have a specific area, it does contain it's own usage instructions. Because the Core projects are designed to build on each other, this should contain most of the code, and is designed to be used and linked against like a library (further instructions can be found in this folder).
- GeneratorsGraders - Because many of the Core projects deal with various flavors of file parsing, we needed a way to generate these files. This folder contains those generator scripts.
- Fillable Basic JQR - We have included a blank and fillable version of the most recent (hopefully) version of the CCD Basic JQR. If this looks out of date, please talk to your mentor.
- build.sh - The top level build script for your project. When this is run, it should build all the projects you have completed so far.
- CMakeLists.txt - The top level CMakeList for your projects. This will be used by the build script to appropriately build all of your sub-projects. Some sections have been added for you as examples, but as your progress into the more advanced projects, you will need to add sections.
- local_tester.sh - Basically what the CICD pipeline will be running. You can either call this directly, or it will be used in the built in tests in your binary. We highly recommend using this to test with; just because it passes locally doesn't mean it will pass in CICD, but we can guarantee if it doesn't pass locally it won't pass CICD.
- run_tests.sh - Use this to run the tests that were built for your binaries. These will either be Unit Tests (in the case of DataStructures1) or will use the "local_tester.sh" script.
- uncovered_jqrs.md - These are the remaining "Demonstrate" JQR Line Items that we do not expect will be covered in these projects. Talk to your Mentor to develop a plan to complete these (and any other outstanding JQRs as determined by your Mentor).

There are also some other files (which may be hidden) that can assist the Trainee. They are needed by the testing framework, and it is left as an exercise to the Trainee to figure out if/how they should best be used. It is essential that CSD developers develop and continually refine the ability to perform efficient research and apply critical thinking and independent problem solving as a habit, rather than relying only on guidance and resources provided by their organization.

There are some general guidelines to how work should be done in this repo. While your specific mentor may have further guidance or instructions, in general please try to adhere to basic "DevOps" principles. Some (but not all) guidance is listed below:

- Do work on feature branches, Do Not work off of 'master'
- Coordinate with your mentor to create Project Labels
- Create a 'Draft' Merge Request when you begin a new branch. This allows comments and suggestions to be made about work in progress.
- When Merges are ready for Review, change the label, remove the 'Draft', and Coordinate with your mentor for code review.

2 Expectations of Trainees

Trainees are expected to stay engaged with their Mentor and with the Projects. If you are struggling, please ask for help sooner rather than later. Although your Mentor should be keeping tracking of your progress, please keep them in the loop. Many mentors are very busy, and you keeping them informed allows them to track and plan for code review and general guidance timelines.

You must work within the guidelines and limitations of the project specifications, but any other design choice not specified or otherwise limited by instructions are yours to make. These choices should be deliberate, and you should be prepared to explain your reasoning. You will be expected to do the same on the BSLE. If you are unable to determine which design choice is best, please reach out to your mentor. If you find yourself stuck, do some research into the corresponding "Describe" JQRs for that section; it should help. If you find yourself asking "what is the" or "how do I" questions, please research the problem! This is how you progress as an engineer. I guarantee there will be answers to technical questions; nothing in these projects in novel.

Although there is room for you to flex your problem solving, the Specs must be adhered to, and depending on the mentor, they may have their own expectations. At the very least, your code must be passing the CICD pipeline to be merged into the 'master' branch.

Please read the following section to see what type of support you can expect from your mentor. If you are not receiving this, please talk to either a Crew Lead or Senior Developer in your organization.

3 Expectations of Mentors

The primary purpose of these Projects are relieve the stress of trainee curriculum development, while still allowing for some creative freedom. There are basic specs to follow, and some simple adherence testing. These should help to significantly decrease the amount of hands on Q/A time the trainees need to complete their JQRs.

However, the CICD testing does not mean code review isn't required. On the contrary, these projects probably require more intense and more frequent code review. Your trainees will be interacting with (most likely) new concepts, and may resort to simply trying to get to a "working" solution. If you are a mentor, you know what a BSLE panel is. Please hold your trainees to that standard on whichever project they are working on before they move on. Properly done early projects will save you and your trainee lots of time down the line.

While review throughout your trainees whole dev cycle is highly encouraged, at the very least please be prompt on reviewing Merge Requests marked as ready. Please coordinate with your trainee if you are very limited on time, and try to plan for these reviews. They are extremely important, and if done properly, can mean the difference between a pass and a fail on your trainees BSLE.

4 Rough Timeline Estimations

The current standard for JQR completion is 6 months from the date you start them. While this can vary greatly from trainee to trainee based on experience, below is a timeline that breaks down roughly the time each project should take to meet the 6 month mark:

- SimpleCalc: 2 Weeks - 1 Month
- FileCalc: 3 Weeks - 1 Month
- DataStructures1: 1 - 3 Weeks
- ThreadCalc: 3 Weeks - 1 Month
- NetCalc: 2 Weeks - 1 Month

Again, this is very rough timeline, and is designed to serve more as left and right limits than a pacing chart. If you and your mentor are capable of working faster, please do. If you are going slower than this pace, please work with your mentor. Basically, talk to your mentor to coordinate your timeline and stay on track.

This timeline leaves some flexibility for Describe JQRs, JQRs that were not covered in the projects, and the life events that cut into Project time. If you are struggling to find time to consistently work on the projects, talk to your mentor. Do not allow yourself to be in the situation where you are at 5 or 6 months in and feel unprepared to take the exam.

5 How to Use the Tools in this Repo

5.1 CMake

On the blank repo, no CMakeLists.txt files will be present. This will break the root level build.sh. This script should work and all projects should be build into a root build/ directory. This should be done in addition to the build scripts in the project folders. You will find that this is not additional work, but is required to make the root-level build work. Look into add_subdirectory().

5.2 CICD

The CICD pipeline is dependent on build.sh and local_tester.sh working.

The lint command for the pipeline are:

```
find ./ -type f -name "*.c" | xargs clang-format --dry-run -Werror
    --style=file

find ./ -type f -name "*.h" | xargs clang-format --dry-run -Werror
    --style=file
```

This will give feedback against the style in .clang-format. It is largely compliant with Barr-C, but there are some items that are not fixed by it. Refer to the Barr-C specification.

5.3 Testers

The run_tests.sh script is dependent on you registering the tests with CMake. Read about CTest and CMake enable_testing() and add_test() if you are having trouble.

The testers provided int he GeneratorsGraders and Tests directories should be interfaced through the local_tester.sh script locally. The tests are very forgiving on output. Talk to your mentor before digging into the tests. They are functional tests, not unit tests, so they are largely dependent on output to stdout/stderr.

A common point of failure is the names of the binaries. The setup() functions in the tests are good places to start checking if you must dive into the tests.

6 Basic Example Project

It is expected that some Trainees are new to C, and that most Trainees are new to the BARR-C Standard. To aid in this transition, we have included some helpful resources:

- The BARR-C Standard The codestyle that CSD generally uses, and what is used for the BSLE. Following the standard does help.
- "Writing Better C" This was a talk given as training to CSD-T. It walks through some C history, libraries for common tasks, and common pitfalls of new C developers. It

also contains a sample C project. It generally complies with the BARR-C standard, and is designed to demo the concepts from the talk.