CPSC 3720 – Warm-Up
(Exposure)

Overview

In this assignment, you will refresh your knowledge from CSPC 270 by:

- Creating a class that performs one or more calculations.
- Keep track of your progress using version control.
- Write passing unit tests for the class.
- Determine how well your code is tested using code coverage.
- Maintain a coding style with a style checker.
- Check for memory leaks using a memory checker.
- Use static analysis to detect bugs and avoid dangerous coding practices.
- Generate documentation for your code using Doxygen.
- Use continuous integration to automate the running of software engineering tools.

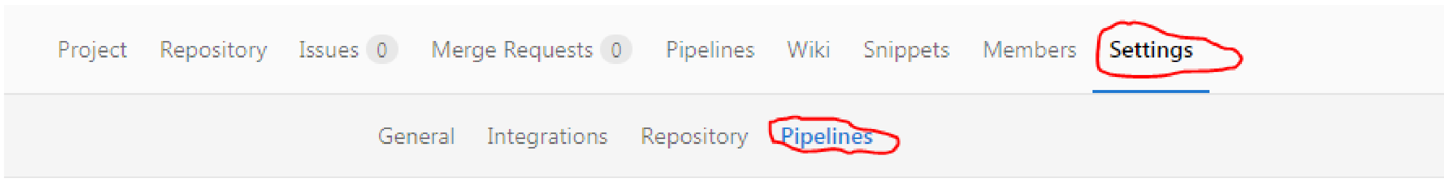
Instructions

Setup

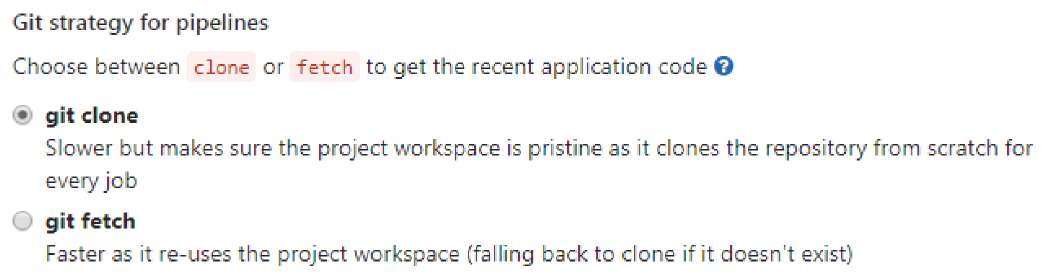
1. Fork the assignment repository so that you have your own GitLab repository for completing the assignment.

    a. If you do not do this step, the marker will not find your assignment repository, and **you will receive an automatic 0 for the assignment.**

2. Set up your GitLab repository for running continuous integration on your project.



    a. Set the *Git Strategy* to "`git clone`"



Git strategy for pipelines

Choose between `clone` or `fetch` to get the recent application code

[✓ git clone]
Slower but makes sure the project workspace is pristine as it clones the repository from scratch for every job

git fetch
Faster as it re-uses the project workspace (falling back to clone if it doesn't exist)

Completing the Assignment

1. Create a local clone of your assignment repository.
    a. Run the command `git remote` and verify that there is a remote called `origin`.
        i. `origin` is the link to your repository of GitLab and is where you will be pushing your changes.
2. Create a class that implements the two methods specified in the provided `.h` file.
    a. One method calculates the exposure of a photograph given the values for aperture, shutter speed and ISO.
    b. One method indicates if a photograph will have the correct exposure or be overexposed or underexposed for a given lighting situation. The range of possible lighting situations (20+) has been reduced into four groups: night, dim light, medium light and bright light.
3. Test your implementation by creating unit tests.
    a. You don't need to test the class thoroughly, but your tests should demonstrate complete testing of at least one equivalence partition.
4. An example `Makefile` is provided to help you build and test your program, run static analysis, memory leak checking, style checking and code coverage.
    a. The Makefile has the following targets:
        i. `tests`: Builds and runs the unit tests.
        ii. `allTests`: Runs all testing targets.
        iii. `memcheck`: Runs `valgrind –memcheck` to check for memory leaks.
        iv. `coverage`: Runs `lcov` to generate HTML reports of the unit testing code coverage.
            1. The HTML reports are to be located in the `coverage` directory.
        v. `style`: Runs `CPPLINT` to check for coding style violations.
        vi. `static`: Runs `cppcheck` to check for bugs and bad programming practices.
        vii. `docs`: Generates HTML documentation in `docs/code` using `Doxygen` for your application.

Grading

Based on your demonstrated understanding of unit testing, version control, and good software engineering practices, you will be graded. Examples of items the grader will be looking for include (but are not limited to):

- Unit tests test the methods.
    - Unit tests show the use of equivalence partitioning and boundary value analysis in the creation of test cases.
    - Statement coverage is as close to 100% as you can get when looking at the `lcov` report.
        - Depending on the implementation, you should be able to reach 90% or better line coverage.
        - As there are only two methods, function coverage should be 100%.

- Proper use of version control.
    - Version control history shows an iterative progression in completing the assignment.
    - Version control repository contains no files that are generated by tools (e.g. object files, binary files, documentation files)
- The status of the most recent build in your repository's GitLab pipeline nearest the deadline (but not after the deadline) is green (i.e. passes).
    - Memory leak checking, static analysis, and style analysis show no problems with your code.
- Source code contains no "dead code" (i.e. code that is commented out).

Submission

There is no need to submit anything, as GitLab tracks links to forks of the assignment repository.

- Ensure that the permissions are correctly set for your repository on GitLab so the grader has access. **You will receive an automatic 0 (zero) for the assignment if the grader cannot access your repository.**

Hints

- Remember that `double x = 1/5` will be return in `x=0`, as integer division is performed. Doing `double x = 1.0/5` will give you `x=0.2.`

- Several exposure calculators found online can be used to validate the exposure calculation and determine testing values for each equivalence partition (`correct`, `underexposed`, and `overexposed`) of each lighting situation. Some include:
    - https://www.omnicalculator.com/other/exposure
    - http://endoflow.com/exposure/
    - https://rechneronline.de/exposure/