CPSC 2720 – Assignment 2 (Spring 2019)

Overview

In this assignment, you will:

- Write implementation for the various geometric shapes that were tested in the first assignment.
- Keep track of your progress using version control.
- Run code coverage to ensure all code is tested.
- Use various software engineering tools to help create quality software (static and style analysis, memory leak checking, continuous integration).

Instructions

1. Fork the repository at http://ares-mat17.cs.uleth.ca/gitlab/cpsc2720/Geometry/asn2. As it is a CS department server, you will only be able to do this on the campus network (or via VPN).
2. Set your notification settings for this repository to "Watch" so you will receive email notification if there are any changes to repository (e.g. clarifications are added to the instructions).
3. Fork the repository so you have your own copy.
4. Set the project visibility for your forked repository to "Private".
5. Add the marker as a member of your project with the permission "Reporter". You will be provided with their CS department user name in the lab and/or on Moodle. This is needed so the marker can grade your assignment.
6. Setup your GitLab repository for running continuous integration for your project.
    a. Set the *Git Strategy* to "`git clone`"
    b. Set the Timeout to 5 (i.e. 5 minutes). Your CI job will be small, so this should be lots of time and will prevent any infinite loops from tying up the CI server

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. Copy the contents of the `include` directory and `test` directory from your previous assignment to the corresponding directories in this assignment.
3. Create a `Code::Blocks` console project and add the copied files to the project.
    a. Using `Code::Blocks` makes development easier, but you could complete the assignment using only the provided `Makefile`.
4. Generate the project documentation using `doxygen` or look at comments in the header (`.h`) files to see the specification of the methods.
    a. Use `make docs` to do this.

5. Read through the generated documentation for all of the methods in all of the classes to understand the expected output of the methods.
6. Add the `.h` files to your `Code::Blocks` project.
7. Use your unit tests from Assignment #1 to verify your implementation.
    a. Use the Test-Driven Development approach to complete the assignment. This will help you to focus on implementing one shape class at a time.
        a. Add one unit test file (e.g. TestQuad.cpp) to the `Code::Blocks` project.
        b. Write the implementation for the methods of the software-under-test (SUT) (e.g. the Quadrilateral class).
            i. Two floating-point values are considered equal if they are within an error bound of 0.0001 (i.e. 4 decimal places).
            ii. The following pages may be useful, as they contain geometric formulas:
                1. http://www.math-salamanders.com/image-files/geometry-terms-and-definitions-geometry-cheat-sheet-4-2d-shapes-formulas.gif
                2. http://www.math-salamanders.com/image-files/high-school-geometry-help-geometry-cheat-sheet-5-3d-shape-formulas.gif
        c. Get all the tests to pass.
        d. Continue adding test fixtures and implement the SUT until all of the classes are completed and all tests pass.
8. Use code coverage to make sure that all (i.e. 100%) of your implementation is tested.

Notes

- A `Makefile` is provided which:
    - Builds and runs a testing executable (`make tests`).
    - Checks for memory leaks (`make memcheck`)
    - Runs static analysis (`make static`)
    - Runs style checking (`make style`)
    - Runs code coverage (`make coverage`)
    - Runs all of the checks (`make all`)
- A continuous integration configuration file (`.gitlab-ci.yml`) is provided for you. It is not expected that you will need to change this file.
- You will not be able to run the code coverage until you have code to examine (i.e. `.cpp` files in both `src` and `test`)

Grading

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

- Version control history shows an iterative progression in completing the assignment. You are expected to have a minimum of **eight new commits** in your repository (e.g. one for each new source code file).

- Version control repository contains no files that are generated by tools (e.g. object files, binary files, documentation files)
- Memory leak checking, static analysis and style analysis show no problems with your code.
- Implementation shows understanding of software engineering principles.
- Unit tests produce 100% statement coverage (function coverage can be less than 100%).

Submission

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

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

Appendix

Updating the Assignment Files

The following information is to be used in the case that the assignment is updated with clarifications or corrections.

1. Create an upstream remote to the original assignment repository from your local repository:

   ```
   git remote add upstream
   http://ares-mat17.cs.uleth.ca/gitlab/cpsc2720/Geometry/asn2.git
   ```

   This command creates a link from your local repository to the original assignment repository. You will not have permissions to push to the assignment repository, so you will get an error if you try.

2. To get the updates from the assignment repository, you can pull them into your local repository.

   ```
   git pull upstream master
   ```

   a. If there are any merge conflicts, you will need to resolve them.

3. Commit the changes to your local repository.