CPSC 2720 – Assignment 2

Overview

In this assignment, you will:

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

Instructions

1. Fork the repository at http://gitlab.cs.uleth.ca/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. Set the project visibility for your forked repository to "Private".
    a. This means that only those you allow will be able to access your work.
4. Add the marker (`mark2720`) and lab instructor (`wilsonn`) as members of your project with the permission "`Reporter`".
5. 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 follow directories from your Assignment #1
    a. Your unit tests into the `test` directory.
    b. The header files into the `include` directory.
3. **Write implementation for the methods of the concrete classes** (e.g. Quadrilateral, Cone, Circle).
    a. You may find the following pages useful, as they contain geometric formulas:
        i. http://www.math-salamanders.com/image-files/geometry-terms-and-definitions-geometry-cheat-sheet-4-2d-shapes-formulas.gif
        ii. http://www.math-salamanders.com/image-files/high-school-geometry-help-geometry-cheat-sheet-5-3d-shape-formulas.gif
4. Use your unit tests from Assignment #1 to verify your implementation.

    a. All the unit tests should fail initially, as there is no implementation (yet) for the methods.
    b. Use the Test-Driven Development methodology to incrementally 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 at a time
        b. Get all the tests to pass.
        c. Continue adding test fixtures and implement that class until all of the classes are completed and all tests pass.
        d. You may want to use `DISABLED_` to skip tests (e.g. `TEST(MyTests, DISABLED_CircleTest)`) to disable tests while you are creating your implementation.
5. A `Makefile` is provided that allows you to run code coverage to see if your unit tests are covering all of your source code. Based on the results, you may need to update your unit tests.
    a. To run the code coverage tool `lcov`, use the command:

        ```
        make coverage
        ```

        This will create the directory `coverage` which contains HTML files showing how much of your code your unit tests exercised.
    b. Open the file `index.html` in a web browser.
    c. You are expected to have 90% or better coverage for the concrete classes.

Grading

You will be graded based on your demonstrated understanding of:

1. Complete testing of your implementation as shown by code coverage.
2. Version control,
3. Good software engineering and development practices.

Examples of items the grader will be looking for include (but are not limited to):

- All methods of all concrete classes are tested by unit tests.
- Version control history shows an iterative approach to completing the assignment.
- Implementation shows understanding of software engineering design principles.
- Source code is appropriately documented using the principles discussed in class (e.g. all classes and methods) so that `doxygen` can extract the documentation.
- `valgrind` shows that no memory blocks were lost in the implementation.

Notes

- A `Makefile` is provided which:
    - Builds a testing executable (`make testShapes`)
    - Runs code coverage (`make coverage`)
    - Checks for memory leaks (`make memcheck`)
    - Runs static analysis (`make static`)
    - Runs style checking (`make style`)
    - Creates documentation (`make docs`)
    - 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.

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 grader has access. You will receive an automatic 0 (zero) for the assignment if the grader cannot access your repository.

- Creating an independent copy of the repository (i.e. not a fork) will result in an automatic 0 (zero) as the marker will not be able to find it.

Appendices

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 so you can pull in the updates:

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

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

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

   ```
   git pull upstream  master
   ```