CPSC 2720 - Assignment 1

Overview

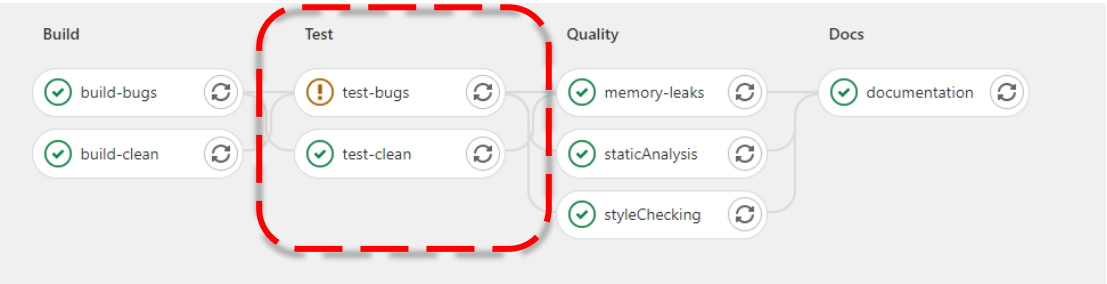
In this assignment, you will:

- Write unit tests for a provided library that represents windows and widgets (i.e. images and textboxes) in a GUI.
- Keep track of your progress using version control.
- Report bugs in the library.
- Use various software engineering tools to help create quality software (static and style analysis, memory leak checking, continuous integration)

Completing the Assignment

1. Fork the assignment at http://gitlab.cs.uleth.ca/course2720/assignments/asn1, if you have not done so already.
2. Download the generated documentation from the documentation job of the build pipeline on GitLab.
3. Read through the generated documentation (or the header files) for all of the methods in the following classes to understand the expected output of the methods.
    a. AsciiWindow
    b. Coordinate
    c. Image
    d. Textbox
    e. Widget
    f. Window
4. Determine what tests you will need to create.
    a. What methods need to be tested?
        i. **You are to test all of the public methods of the concrete classes, except for the following:**
            1. **destructors**
            2. **constructors of exceptions**
            3. **Window.draw() and for its subtypes -** this just prints to the screen. A test of this method is provided for you by using `diff`.
    b. What are the equivalence partitions for the methods?
        i. What are the "normal" cases?
        ii. What are the "error" cases?
    c. What are the boundary conditions?
5. Iteratively write unit tests (i.e. write tests for each class in a method-by-method fashion).
    a. Start by writing unit tests that test the "clean" library (`libgui-clean.a`) which is *intended* to be bug-free. This will help to make sure that you don't have any problems with your unit tests.
    b. **There are 8 unique and intentional implementation bugs in the "buggy" library** (`libgui-bugs.a`) and your unit tests should identify as many of them as you can.
        i. If a unit test passes for the "clean" library but fails for the "buggy" library, then you can be sure that you found one of the seeded bugs. These bugs must be reported by creating an issue on **your** assignment repository (http://gitlab.cs.uleth.ca/USER2720/asn1).

When you find bugs, the build pipeline will look like this:



ii. **Do not** change any of the `.h` files. You are to write tests according to the specification of the methods given. If you think you found a bug in the "clean" library, create an issue on the assignment repository (http://gitlab.cs.uleth.ca/course2720/assignments/asn1).

iii. **Do not** create your own implementation of the class files to make your unit tests pass. Some of your unit tests should not pass for the "buggy" library.

c. Unit tests are to be organized by the class they test (i.e. each class-under-test has a corresponding test fixture file).

6. File bug reports in **your** repository for the issues in the `libgui-bugs.a` discovered by your unit tests. Be specific enough that it will be clear to the marker what bug you found. Ensure that you are reporting bugs on **your own repo**, and not on the class repo which you have forked from earlier.
   a. Try your best to report only **unique** bugs. For example, assume that superclass `A` has two subclasses `B` and `C`. Class `A` has the method `foo()`, which is inherited by classes `B` and `C`. If your unit tests reveal the same bug in `B.foo()` and `C.foo()` then the fault is likely in `A.foo()` and should only be reported **once** (not twice). However, you can indicate in the bug report that the failure occurs in both `B` and `C`.

Notes

- A `Makefile` is provided which:
  - Builds a testing executable using both the "clean" (`test-clean`) and "buggy" (`test-bugs-unit` and `test-bugs-diff`) libraries.
  - Checks for memory leaks (`make memcheck`)
  - Runs static analysis (`make static`)
  - Runs style checking (`make style`)
- 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 may find a bug that causes a test to `SEGFAULT`, which stops the remaining tests from running. You can use `DISABLED_` to skip that test (e.g. `TEST(MyTests, DISABLED_SegFaultTest)`).

Grading

You will be graded based on your demonstrated understanding of:

1. Unit testing

2. Version control
3. Bug reporting
4. Good software engineering practices.

Items the grader will be looking for are:

- The number of intentional bugs that you find. You will be allowed to miss one bug and still get full marks for bug finding.
- All public methods of all concrete classes are tested by unit tests.
- Use of equivalence partitioning and boundary-value analysis in the creation of test cases.
- Test cases have a clear and single purpose (e.g. a test case does not test both positive and negative input values).
- Version control history shows an iterative progression in completing the assignment. In other words, you should have more than a few commits.
- 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.
- The build as of the deadline does not fail. A grace period will be given for build failures where it is obvious that a simple build error was corrected shortly after the deadline.
- Bug reports are clear, concise, and describe the problem in such a way that a developer could replicate the problem, and use the format discussed in class.
- Your code is clean and readable (e.g. does not contain commented out sections).

Submission

There is no need to submit anything, as GitLab tracks links to forks of the assignment repository. However, you **must** ensure the following instructions are followed:

- Make sure that **the marker (`mark2720`) has access** to your repository, **otherwise you will receive a 0**. There is a script that should automatically add `mark2720` as a member of the repository, but it is your responsibility to confirm that it is doing its job.
- Using **a different project repository than the one created by forking the assignment repository will result in an automatic 0 (zero)** as the marker will not be able to find your project.