CPSC 2720 – Assignment 2 (Spring 2017)

Overview

In this assignment, you will:

- Write implementation for the various GUI components that were tested in the first assignment.
- Keep track of your progress using version control.
- Run code coverage to ensure all code is tested.
- Document your implementation using `doxygen`.

Instructions

1. Fork the repository at https://bitbucket.org/ulethsecourse/asn2spring2017. Make sure that your forked repository is public (not private!) or your assignment will not be graded.
2. Create a local clone of your BitBucket repository.
3. Compile and run the provided code. You should not have to change the `main.cpp` file. Initially the program will do nothing as there are no tests registered to run.

   It is suggested you do this using Code::Blocks by creating a console project and adding the files from the repository.

   a. You want to have the Code::Blocks configuration file in your local clone of your repository. To do this, when you create the project, check that the "Resulting filename" field does not have a folder with the project's name (Code::Blocks will do this by default, but remove it).

   b. You will need to link in the `cppUnit` library.

      i. Open the *Build options* for the project.

      ii. Go to the *Linker settings* tab.

      iii. Type `–lcppunit` in the *Other linker options* textbox

   c. You will need to tell CodeBlocks where to find files

      i. Open the *Build options* for the project.

      ii. Go to the *Search directories* tab

      iii. In the *Compiler* tab, add the `include` and `test` directories where the header files are.

   d. Confirm that the compiler being used is `g++11` (not `g++`) and that the `--std flag` is set correctly.

      i. Check under `Settings->Compiler->Toolchain executables` that the compiler and linker are set to `g++11`.

      ii. Check that under `Settings->Compiler->Compiler Settings->Other compiler options` has `--std=c++11`.

4. **Write implementation for the methods of the concrete classes**. Use the unit tests you created in the previous assignment to verify your implementation.
    a. All of your `.cpp` files are to be in the `src` directory.
    b. Your unit tests should fail initially, as there is no implementation (yet) for the methods.
        i. For completing this assignment, you may want to incrementally add one unit test file (e.g. `TestImage.cpp`) to the Code::Blocks project at a time, and get all the tests to pass before adding the unit tests for the next class and then implementing that class.
    c. For implementing AsciiWindow.h, this code:

```
Window* w = new AsciiWindow(10,10);
Widget* t = new Textbox(Coordinate{1,1}, 5,1, "Hello");
Widget* i = new Image(Coordinate{5,5}, 3,3);
w->addWidget(t);
w->addWidget(i);
w->draw();
```

    produces output like:

```
----------
|                |
| H e l l o      |
|   W o r l      |
| d              |
|                |
|         I I I  |
|         I I I  |
|         I I I  |
|                |
|                |
----------
```

    d. Make sure that your implementation does not have memory leaks. You can either:
        i. Run `valgrind` from the command line on your executable (e.g. `valgrind gui`)
        ii. In Code::Blocks, select `Valgrind->Run MemCheck`.
    e. A `Makefile` is provided that will allow you to run code coverage (`make 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.
        i. 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`)
5. If you run into problems, please file a bug report in the original repository (https://bitbucket.org/ulethsecourse/asn2spring2017). This is preferred to sending the instructor an email, as this will allow other students who run into the same problem to get a more immediate answer.

Grading

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

- All public methods of all concrete and abstract classes are tested by unit tests. You are expected to write unit tests for all public methods (except destructors). In some cases you will need to use a derived class to test the methods in a base class. Also, you are not expected to test the exception classes themselves.
- Version control history shows an iterative progression in completing the assignment.
- Implementation shows understanding of software engineering principles.
- Unit tests cover all (or almost all) of the `.cpp` files.
- The implementation does not contain memory leaks.
- Tests are appropriately documented using the principles discussed in class (e.g. all classes and methods) so that `doxygen` can extract the documentation. As this documentation can be generated, you need not generate it yourself (i.e. don't have it in your repository).

Submission

Submit the URL of your forked repository to Moodle. The grader will clone your repository as of the deadline for grading.

- You can submit the URL of your repository as soon as you have created it (i.e. before you have started), so you don't forget to submit it later.