COMP 1633 Assignment 1 (Testing)

Given: Monday, January 17, 2022
Electronic Copy Due: Monday, January 31, 2022 at 11:59pm sharp
Hard Copy Due: Tuesday, February 1, 2022 at 1:00pm sharp

Objectives

- To practice developing and executing a comprehensive software test plan.
- To gain experience with `make` and file projects consisting of multiple files.
- To use an automated testing tool (`GoogleTest`) to perform functional testing of a module.
- To continue to develop problem solving skills.
- To expand C++ programming skills.

The Project

There are two main components in this assignment:
1. designing comprehensive functional (black box) test plans for one existing function
2. implementing your test plans using the `GoogleTest` framework

The module to be tested

You are to test the Medusoid addition from the 1631 assignment 2 in the Fall 2021 semester. For anyone that did not take 1631 last fall the assignment specification will be available on BlackBoard and a working executable will be available in the assignments/a1 directory in the 1633 class directory. While the 1631 assignment was done without functions it has been changed so this is now a function. As well, the original assignment used individual integers for the digits. This has been changed to a type called Medusoid_number which is an array of size 6 of type integer. You are being provided with two modules, `m_add.h` and `m_add.o`. The `.h` file provides the function signature for the function and documentation that describes the details of the function.

In order to make the evaluation of this assignment easier when creating the test plan and implementing it using GoogleTest all of your test cases must use the age of 3.

Since you are doing black box (or functional) testing, there is no need for you to have access to the source code. Therefore, you are only being provided with the compiled object module (the .o file) for this function. However, in order to assist you with designing a test plan, the assignment specs and an executable program for the assignment are also being provided.

The Test Plan

You are to develop one comprehensive test plan for the `add` function. Comprehensive means that your test plan must thoroughly test the function.

As a first step, you are to create a **partition diagram** (the tree diagram) for the function. This diagram can be manually drawn, but **must** be neat and readable. Scan or take a photograph of your partition diagram and name this image file `username_a1_partition.jpg`, where the username is your INS account name. Using

the partition diagram, create a test plan. Your plan must be organized in the format shown in class, with six columns:

Test Case #	Reason for Test	Test Data	Expected Result	Actual Result	Pass/Fail

An MS Word document that contains a template table is provided that **must** be used to create your test plan. You are to complete the ID block at the top of the document and fill in the table with your test plan.

A majority of the marks for this assignment will be for the test plan, specifically for its *completeness*, *understandability* and *conciseness* (i.e. redundant test cases should not occur).

Executing the Test Plan

The implementation of your test plan **must** be done using the `GoogleTest` framework. Using the `GoogleTest` framework will be discussed in lab. You are to create a testing module called test_m_add.cpp which will contain an ***appropriately named*** macro corresponding to each test case in your test plan. Appropriately named means that the name completely describes what this test case is testing. The examples provided in the GoogleTest lab use appropriately named test macros. As well, this site describes a number of other conventions that could be used:

https://dzone.com/articles/7-popular-unit-test-naming

You are also required to create a `makefile` for managing the building of your test program. In addition, your `makefile` must include both the "**`all`**" and the "**`clean`**" targets as discussed in the **`make`** lab. There are marks allocated for the creation of a working **`makefile`**.

Implementation Approach

Testing should be done incrementally, just like the implementation of a program. Thus, first create the `test_m_add.cpp` module which should hold one TEST macro for each of your test cases. Start with one test case and implement a test macro for it. Whether you are testing a valid or invalid case, your test should pass - this requires making the *appropriate* check. Should the test fail, you will need to determine what is wrong with your **test** and fix it, as the provided function is known to work, so failure will NOT be due to faulty function code. Once your test passes, you should move on to the next test case, by adding another TEST macro. **The order of your TEST macros MUST BE THE SAME as the order in of the tests in your test plan.**

The process of adding test cases to your test module will be made easier if your `makefile` creates and runs your test program with the `m_add.o` module, as was done in the GoogleTest lab. Be careful with the "clean" target so that the provided `m_add.o` is not deleted!

Documentation

In your `test_m_add.cpp` module you are required to have the normal ID Block. If the names of your test cases are descriptive then no additional documentation is required. However, if your tests are poorly named then complete function documentation is required. Your name should be on both your partition diagram and test plan.

Submission Requirements

Please follow these instructions exactly. Read all instructions carefully before starting.

1. Create a directory called `asg1`.
2. Using a File Transfer program, like **`FileZilla`**, copy the file containing your partition diagram and the Word document containing your test plan to your INS home directory. When connecting to INS using a File Transfer Program you will need to be running OpenVPN and the address of INS needs to be sftp://ins.mtroyal.ca
3. Copy **ONLY** the partition diagram, the test plan document, the `makefile` , all `m_add` files and your `test_m_add.cpp` file into the `asg1` directory.
4. `cd` into `asg1`.
5. Begin a LINUX `script` session.
6. Use `ls -al` to display the contents of the directory.
7. Use `cat` to display the following files in the order specified:
   ```
   test_m_add.cpp
   makefile
   ```
8. Use `make` to build your program, which will also run your test module.
9. Terminate the `script` session.
10. `cd` to the parent directory of `asg1`.
11. Submit your assignment electronically using `submit` (the item to submit will be the `asg1` directory).
12. You will need to print your test plan document (and your partition diagram as well if you have created it electronically). – This will have to be done on a personal printer or you will have to pay at MRU since the line printer cannot print a Word document. **There is NO need to print your typescript as I will view your electronically submitted typescript file.**
13. If this course has shifted back to in-person then you will need to submit a hardcopy of your test plan document and partition diagram (stapled together) in the blue assignment box (**#4**) outside the MACO office.