122COM: Unit testing

David Croft
david.croft@coventry.ac.uk

March 9, 2016

Abstract

This week we are looking at implementing automated unit testing for your programs.

1 Introduction

Testing is a vital part of writing software. It is, however, time consuming and often dull. This week you will be looking at writing automated unit tests to check that the code you wrote in previous weeks is working as expected.

This week will be taught in Python and C++. It should be clear by this point which language you are expected to use for your answers.

You will have two weeks to work on these exercises, however you will have phase test 2 during the lab session for one of those weeks.

The source code files for this lab are available at:

```
https://github.com/covcom/122COM_testing.git
```

```
git@github.com:covcom/122COM_testing.git
```

2 Unit testing

You should be familiar with the idea of automated testing code from the excercises in the previous weeks. In a number of the the lab exercises (searching, sorting, data types, pointers, c++) the code provided in the github repos would also test your code when you had completed the task.

The approach used in the previous weeks was very basic, the tests were simply written into the main functions of the program and ran whenever the program ran. This week we are looking at implementing automated testing in a more professional and capable manner.

2.1 Python

Going to be using the built-in unittest module. Not the most capable of the options but still good and available on all platforms.

To run the unit tests you need to run one of the following commands. The first command will definitely work (if you code is correct), the second command requires that you put `unittest.main()` in your `TESTCASES.py` file as well.

```
>> python3 -m unittest TESTCASES.py      >> python3 TESTCASES.py
```

2.2 C++

Going to be using cxxtest. Not the most capable of the options out there but straightforward and available on most platforms. Unfortunately things are not as simple as they were with Python.

As with Python we have the code we want to test in one file and the tests we want to run in another. Firstly we are going to need to run the `cxxtestgen` program which takes our tests file and produces a "test runner" program for us. Secondly we are going to need to compile the test runner. Thirdly we need to actually execute the test runner program.

```
>> bin/cxxtestgen --error-printer TESTCASES.h -o runner.cpp

>> g++ --std=c++11 runner.cpp -o runner

>> ./runner
```

3 ADT testing

Pre-lab work:

1. You will find versions of the stack and queue code from the data structures weeks in the lab_stack and lab_queue files.
   - If you want to replace the provided code with the stack and queue code that you wrote then go ahead.
2. A complete series of test cases has been provided for the stack code in the test_lab_stack files.
   - BIT and MC should be using the Python version. Everyone else should be using the C++ version but can do the Python version as well if they want.
3. Run the unit tests for lab_stack. A bug has been introduced into the stack code, use the unit test results to identify, locate and fix the bug.

Extended work:

As you have seen over the last few topics: trying to compile, profile or test C++ code can involve multiple complicated commands. Since it's a hassle to remember all the commands and to keep typing them, various solutions have been found. One of the most widely used of which is the make file.

4. Have a look at `makefile`. It contains all the commands necessary to produce, test and run the unit tests for this week.
    - To use the make file just type `make` from the command line.
5. Investigate makefiles.
    - Make file syntax is often confusing and obscure so feel free to ask questions and investigate online. While they can be a pain to learn they are a very useful skill.

Lab work:

6. A minimal framework of a set of queue test cases has been provided in test_lab_queue. Write a complete set of test cases for your queue implementation from earlier in the module, or use the implementation provided.
7. Write from scratch a series of unit tests for your set implementation from the data structures week.

Recap:

8. If you have completed all the tasks then make sure that you have finished all the green and yellow tasks from the previous topics.
    - In particular this is a good opportunity to finish the performance week work which you may have skipped due to phase test 1.