**CppUnit Documentation**

Quick summary of how to write unit test with CppUnit

* Need a class to test on with a constructor? – No a constructor is not necessary
* Write your test class and include
  + Creation and registration of test suite
  + class methods for each test in the suite
  + setUp and tearDown overrides to construct and delete instance variables (fixture only)
* Write code we will execute to actually run tests
  + Need a test result collector with listeners for test results and progress
  + Need test runner where we add tests from registry and run
  + Need an outputter that takes collected results and writes them
  + Can output 0 or 1 if tests successful or not

A template of the basic structure for testing and compilation is also available.

Writing a test fixture

* #include <TestFixture.h>
* Create a class for the tests
* Class functiontest : public CPPUNIT\_NS::TestFixture
* TestFixture is inherited from TestCase – wraps TestCase with setUp and tearDown methods that provide common variables across the tests – not necessary
* Initialise fixture by overriding setUp to construct instance variables
* Clean-up variables at end by overriding tearDown
* Implement a method for each test that can use fixture variables
* Methods can be run using TestCaller or collected into a TestSuite – preferred

Test assertions

Tests are based on assertions that must all be true for a test to pass, of which there are several of potential interest

* CPPUNIT\_ASSERT(condition)
  + Asserts that a condition is true
* CPPUNIT\_ASSERT\_EQUAL(expected, actual)
* CPPUNIT\_ASSERT\_THROW(expression, exception)
* Also add \_MESSAGE to provide an additional message upon failure – could be useful

Test suite and executing tests

* #include <TestSuite.h>
* Have to register our TestSuite (in unnamed registry is okay) with CPPUNIT\_TEST\_SUITE\_REGISTRATION (functiontest) – functiontest is the name of our TestFixture/TestCase (in same file we fully write out test methods - eg test.cc)
* In the header for the TestFixture we then initialise our test suite and add the tests we will perform to it
  + CPPUNIT\_TEST\_SUITE(functiontest)
  + CPPUNIT\_TEST(test1)
  + CPPUNIT\_TEST(test2)
  + CPPUNIT\_TEST\_SUITE\_END
* Where test1 and test2 are the methods for the tests we are running
* Now in our executed file we begin to pull things together
* #include <TestRunner.h>
* Define TestRunner CPPUNIT::TestRunner testrunner
* Add tests to our runner with testrunner.addTest(test)
* When we add tests we can do this individually or preferably via the registry to get our test suite - use the following in place of test when adding:
  + CPPUNIT\_NS::TestFactoryRegistry::getRegistry().makeTest()
* Run tests with testrunner.run(testresult) where testresult is a TestResult object (next)

Test results

* #include <TestResult.h>
* Define TestResult object first with CPPUNIT::NS::TestResult testresult
* #include <TestListener.h>
* Register a TestListener to the TestResult to get info on the testing using testresult.addListener – there are a few different types of listener, useful are:
  + TestResultCollector - collects the results of executing a TestCase
    - CPPUNIT\_NS::TestResultCollector collectedresults
    - testresult.addListener(&collectedresults)
  + BriefTestProgressListener – gives output for each test in the suite – more verbose
    - CPPUNIT\_NS::BriefTestProgressListener progress
* It is at this point we define the TestRunner and add our tests
* Use an outputter to receive a summary of the tests once it has finished (next)
* Then we can return collectedresults.wasSuccessful() ? 0:1 as a return of our main function

Different output formats of the TestResultCollector

* CompilerOutputter
  + CPPUNIT\_NS :: CompilerOutputter compileroutputter (&collectedresults, std::cerr);
  + compileroutputter.write ();
  + Outputs in compiler compatible format. This is the outputter for the fraction tutorial class.
* XmlOutputter
  + Outputs in XML format. Example of writing to xml file
  + std::ofstream myfile;
  + myfile.open ("xmltestresults.xml");
  + CPPUNIT\_NS :: XmlOutputter xmloutputter (&collectedresults, myfile, std::string("ISO-8859-1"));
  + xmloutputter.write ();
  + myfile.close ();
  + I don’t know much more about xml files but it works like this to file so shouldn’t be too hard to edit
* TextOutputter
  + Outputs to text stream.

Failed tests

* When there’s a failure the outputter will print the location of the failure giving the file and line number
* It will also print the failed expression
* If the test contains more than 1 assertion and an earlier one fails, it will print this failure and stop before trying the other assertions
* We can also add a message to assertions for more details