

Crash Course in using CppUnit

Introduction

This document will introduce you to a testing framework called **CppUnit**. CppUnit is a C++ port of the **JUnit** testing framework developed by Erich Gamma and Kent Beck. This document describes the latest, stable version of CppUnit (version 1.8.0). The main purpose of CppUnit is to support developers in doing their unit testing of C++ programs. This document assumes that you already have the MinGW and MSYS packages installed.

Installing CppUnit

Installation of CppUnit can be broken down into the following steps:

1. Download the [local copy of CppUnit](#). Start a MSYS shell by double-clicking on the MSYS icon on your Windows desktop. If you have set the HOME variable, the current directory in the MSYS shell will be the one given by the HOME variable (otherwise, it will be some default directory set by MSYS). We will assume that the CppUnit archive file is saved to this directory. Now, unpack the CppUnit archive file as follows:

```
tar -zxvf cppunit-1.8.0.tar.gz
```

The above command will create a new directory called cppunit-1.8.0. You can type "ls" to check.

2. Compile the CppUnit source files:

```
cd cppunit-1.8.0
./configure
make
make install
```

3. Test that CppUnit has been installed properly.

```
cppunit-config --version
```

If you see 1.8.0 as the output, then CppUnit has been installed properly. You are now ready to use CppUnit for your development work.

Using CppUnit

Overview

In this section, we will provide you with some general steps on how to go about using CppUnit. The next section illustrates the key steps using an example. My advice is to skim through this section quickly for an overview. Then, as you go through the example in the next section, refer back to this section frequently to get the whole picture.

Assuming that you want to test a class called Parser. The following are the general steps to use the CppUnit framework to test this class:

1. Write a class (let's call it TestParser) to test the Parser class. This class must inherit the class TestCase which is defined by the CppUnit framework.
2. Create a constructor for this class, passing a name that is representative of the set of tests for this class as the parameter.
3. Create a *fixture*. A test fixture is a set of sample objects that you want to (re)use during testing. For example, you might create a few sample source files for the Parser to parse. CppUnit provides a setUp and a tearDown method to manage the fixture. Therefore, you can create file objects in setUp to open the source files and release these resources in the tearDown method. The important thing to note is that setUp and tearDown will be called for every 'test' that you run.
4. Each 'test' you perform is represented by the implementation of a method in the test class. For example, if you want to test whether the parser extracts the tokens correctly, you can implement a method called testGetToken. The collection of test methods you implement forms a test suite.
5. In each test method you create, use the assertion mechanism provided by CppUnit to compare the results of running the test and the expected results. This will enable you to create repeatable tests as well as saving you lots of time from visually inspecting the results.
6. Finally, use the textual version of the TestRunner tool to run the tests and collect the results. As each test is run, CppUnit will provide feedback on whether the test ran successfully, or the test failed, or an exception has occurred.

Example

In this section, we will describe how you can use CppUnit using an example (the reference section provides the link to download the sample files used for this guide). Take a few minutes to examine the following two classes (.h and .cpp files) to see what they are doing:

- Course.h

```
#ifndef Course_h
#define Course_h

#include <string>

class Course {
public:
    // Default constructor
    Course();

    // Constructor
    Course(std::string nm, int gr);

    // method to get the name of the course
    std::string getCourseName();

    // method to get the grade of the course
```

```

    int getCourseGrade();

private:
    std::string course_name;    // name of this course
    int grade;                 // grade of this course
};
#endif

```

- Course.cpp

```

#include "Course.h"

// default constructor
Course::Course() {
    course_name = "";
    grade = -1;
}

// constructor
Course::Course(std::string nm, int gr):course_name(nm) {
    grade = gr;
}

// method to get the name of the course
std::string Course::getCourseName() { return course_name; }

// method to get the grade of the course
int Course::getCourseGrade() { return grade; }

```

- Student.h

```

#ifndef Student_h
#define Student_h

#include <iostream>
#include <string>
#include "Course.h"

const int MAXNUM = 20;    // Maximum number of courses allowed per student

class Student {
public :
    // Constructor
    Student(std::string nm, std::string no);

    // Method to return student's name
    std::string getStuName();

    // Method to return student's number

```

```

std::string getStuNumber();

// Method to assign a grade to a course
void assignGrade(std::string co, int gr);

// Method to return the grade of a course
int getGrade(std::string co);
private:
    std::string name;           // name of the student
    std::string number;       // the student's number
    Course course_grades[MAXNUM]; // courses taken by student
    int no_of_courses;        // the current number of courses taken
};
#endif

```

- Student.cpp

```

#include "Student.h"

// Constructor
Student::Student(std::string nm, std::string no):name(nm), number(no) {
    no_of_courses = 0;
}

// Method to return student's name
std::string Student::getStuName() { return name; }

// Method to return student's number
std::string Student::getStuNumber() { return number; }

// Method to assign a grade to course
void Student::assignGrade(std::string co, int gr) {
    // check whether the maximum number of courses have been taken
    if (no_of_courses == MAXNUM) {
        std::cout << "You have exceeded the maximum number of courses !\n";
        return;
    }
    // create a new course
    Course c(co, gr);
    course_grades[no_of_courses++] = c;
}

// Method to return the grade of a course
int Student::getGrade(std::string co) {
    int i = 0;

    while (i < no_of_courses) {
        //check if course name the same as co
        if (course_grades[i].getCourseName() == co)
            return (course_grades[i].getCourseGrade());
    }
}

```

```

        i++;
    }
    return(-1);
}

```

Basically, there are two classes: Course and Student. Each Course contains a name eg. CS3215 and an integer grade which ranges from 0 to 100. Each Student has a name, a number as well as a list of course grades. You can add the grade that a student scores at a particular course using the assignGrade method and retrieve the grade of a particular course using the getGrade method. Now, compile the files as follows:

```

g++ -c Course.cpp
g++ -c Student.cpp

```

You will see two files - Course.o and Student.o created in the directory. Just leave them alone for the moment. Next, create the test class. The following are the test files we wrote for the Student class (called TestStudent.h and TestStudent.cpp):

- TestStudent.h

```

#ifndef TestStudent_h
#define TestStudent_h

#include <iostream>
#include <string>

// Note 1
#include <cppunit/TestCase.h>
#include <cppunit/TestSuite.h>
#include <cppunit/TestCaller.h>
#include <cppunit/ui/text/TestRunner.h>

#include "Student.h"

class StudentTestCase : public CppUnit::TestCase { // Note 2
public:
    // two constructors - Note 3
    StudentTestCase() {}
    StudentTestCase(std::string name) : CppUnit::TestCase(name) {}

    // method to test the constructor
    void testConstructor();

    // method to test the assigning and retrieval of grades
    void testAssignAndRetrieveGrades();

    // method to create a suite of tests
    static CppUnit::Test *suite ();
};
#endif

```

- TestStudent.cpp

```

#include "TestStudent.h"

// method to test the constructor
void StudentTestCase::testConstructor() { // Note 4
    // create a student object
    Student stu("Tan Meng Chee", "94-1111B-13");

    // check that the object is constructed correctly - Note 5
    std::string student_name = stu.getStuName();
    CPPUNIT_ASSERT(student_name == "Tan Meng Chee");
    std::string student_number = stu.getStuNumber();
    CPPUNIT_ASSERT(student_number == "94-1111B-13");
}

// method to test the assigning and retrieval of grades
void StudentTestCase::testAssignAndRetrieveGrades() {
    // create a student
    Student stu("Jimmy", "946302B");

    // assign a few grades to this student
    stu.assignGrade("cs2102", 60);
    stu.assignGrade("cs2103", 70);
    stu.assignGrade("cs3215", 80);

    // verify that the assignment is correct - Note 6
    CPPUNIT_ASSERT_EQUAL(60, stu.getGrade("cs2102"));
    CPPUNIT_ASSERT_EQUAL(70, stu.getGrade("cs2103"));

    // attempt to retrieve a course that does not exist
    CPPUNIT_ASSERT_EQUAL(-1, stu.getGrade("cs21002"));
}

// method to create a suite of tests - Note 7
CppUnit::Test *StudentTestCase::suite () {
    // create a test suite
    CppUnit::TestSuite *suiteOfTests =
        new CppUnit::TestSuite("StudentTestCase");

    // add the tests
    suiteOfTests->addTest
        (new CppUnit::TestCaller<StudentTestCase>
         ("testConstructor", &StudentTestCase::testConstructor));

    suiteOfTests->addTest
        (new CppUnit::TestCaller<StudentTestCase>
         ("testAssignAndRetrieveGrades",
          &StudentTestCase::testAssignAndRetrieveGrades));
    return suiteOfTests;
}

```

```

}

// the main method - Note 8
int main (int argc, char* argv[]) {
    CppUnit::TextUi::TestRunner runner;
    runner.addTest(StudentTestCase::suite());
    runner.run();
    return 0;
}

```

Notes for the preceding code:

- Note 1
Remember to include these four header files: TestCase.h, TestSuite.h, TestCaller.h and TestRunner.h in all the header files (.h) of the test classes you are implementing.
- Note 2
Every test class that you wrote must inherit the class CppUnit::TestCase.
- Note 3
These constructors are quite standard. You can just cut and paste for every test class you create.
- Note 4
This is an example of a method written to test one of the methods of the Student class. In this case, it's the constructor.
- Note 5
The assert macro is one of the most common statements you will use. The argument is a boolean expression that must evaluate to either a true or false value.
- Note 6
The assertEquals method can also be used to check whether a test passed or failed. It takes in two arguments and compare them. If they are not equal, an exception will be raised to indicate that the test has failed.
- Note 7
This class method is used to assemble a suite of tests. It's also fairly standard. Just cut and paste and change accordingly to suit your needs. In particular, change StudentTestCase to the name of your test class and substitute those various test method names with your own.
- Note 8
Again, this main method is fairly standard. So, you can just cut and paste for your test classes. Just change the class name accordingly.

Now, compile your test programs:

```
g++ -c TestStudent.cpp
```

```

In file included from TestStudent.cpp:1:
TestStudent.h:7:30: cppunit/TestCase.h: No such file or directory
TestStudent.h:8:31: cppunit/TestSuite.h: No such file or directory
TestStudent.h:9:32: cppunit/TestCaller.h: No such file or directory
TestStudent.h:10:40: cppunit/ui/text/TestRunner.h: No such file or directory
....

```

When you compile the test program, the above error messages will appear. This is because the compiler is unable to locate the CppUnit header files. The correct way to compile this (assuming you install CppUnit the way we described earlier) is:

```
g++ -I/usr/local/include -c TestStudent.cpp
```

At the end of this step, you would have TestStudent.o. Now, put everything together to create an executable test program (execute the following as a single command):

```
g++ -o tester Course.o Student.o TestStudent.o -L/usr/local/lib -lcppunit
```

The above command will create the executable called **tester**. The -L option is to tell the compiler additional directories it can search for libraries specified with the -l option. The -l option specifies the name of a library (in this case, the CppUnit libraries) to be used by the linker in resolving references. To run the test suite, simply type:

```
tester
```

Exercise

The best way to learn CppUnit is to use it. So, here's a small exercise you can do to get some hands-on practice. Let's say we now extend the Student class by adding a method to find the average grade of all the courses taken by the student. You can add the following piece of code to Student.h and Student.cpp:

```
// In Student.h under public
// Method to return the average grade
float findAveGrade();

// In Student.cpp
// Method to return the average grade
float Student::findAveGrade() {
    float sum = 0.0, average;

    // sum up the marks in all the courses
    for (int i = 0; i < no_of_courses; i++)
        sum += course_grades[i].getCourseGrade();
    average = sum / no_of_courses;
    return(average);
}
```

Now write a method in the TestStudent class to test this newly created method. Give it a try and see whether you really know how to use CppUnit

References

- Download the [sample files](#) use in this guide.
-