



Applied Research Laboratories
The University of Texas at Austin



**CppUnit: An Open
Source C++ Unit
Testing Framework**

Presentation by Eric Hagen

- ◆ CppUnit is the port of JUnit, an open source Java framework for unit testing
- ◆ CppUnit provides a library for making easy to use and implement unit tests
- ◆ Tests can be run either automatically or supervised
 - Output for automatic tests is in XML or text format
 - a GUI is available for Windows supervised testing and debugging

Why is unit testing important?

- ◆ A well maintained set of unit tests represents the most practical design possible, provides evolving documentation with classes and gives the developer confidence in their code
- ◆ Proper unit testing allows for easy and quick testing of extended and/or modified code
- ◆ Unit testing adds stability to code
 - With multiple small, quick tests, many test scenarios can be explored

- ◆ The class Fraction is a simple model of mathematical fractions
- ◆ We want to test Fraction to make sure that it is working as expected
- ◆ The Fraction class can add, subtract and reduce Fractions
 - We want to test these operations as well as test to make sure that an invalid fraction (1/0) is not allowed
- ◆ Finally, we want to modify Fraction and then add test cases to check the modifications

```
// CppUnit-Tutorial
// file: Fraction.h
#ifndef FRACTION_H
#define FRACTION_H

#include <iostream>

using namespace std;

// some helper functions (actually don't belong here)
// calculates the greatest common factor (for reducing)
unsigned int gcf (unsigned int, unsigned int);
// calculates the least common denominator (for expanding)
unsigned int lcd (unsigned int, unsigned int);

// definition of an exception-class
class DivisionByZeroException
{
};

// simple definition of a fraction-class
class Fraction
{
public:
    // constructor
    Fraction (int = 0, int = 1) throw (DivisionByZeroException);

    // copy-constructor and assignment-operator
    Fraction (const Fraction&);
    Fraction& operator= (const Fraction&);

    // comparing operators
    bool operator== (const Fraction&) const;
    bool operator!= (const Fraction&) const;
};
```

Header for the Fraction class

```
    // comparing operators
    bool operator== (const Fraction&) const;
    bool operator!= (const Fraction&) const;

    // arithmetic operators
    friend Fraction operator+ (const Fraction&, const Fraction&);
    friend Fraction operator- (const Fraction&, const Fraction&);

    // output on stdout
    friend ostream& operator<< (ostream&, const Fraction&);

private:
    // method for reducing
    void reduce (void);

    // variables for saving the numerator and denominator
    int numerator, denominator;
};

#endif
```

Header for the Fraction class continued

```
// CppUnit-Tutorial
// file: fraction.cc
#include "Fraction.h"

unsigned int gcf (unsigned int n1, unsigned int n2)
{
    unsigned int TMP;

    if (n1 < n2)
    {
        TMP = n1;
        n1 = n2;
        n2 = TMP;
    }
    if (n2 > 0)
        return (gcf (n2, n1 % n2));
    else
        return (n1);
}

unsigned int lcd (unsigned int n1, unsigned int n2)
{
    return n1 * n2 / gcf (n1, n2);
}
```

Fraction source file

```
.  
Fraction :: Fraction (int n, int d) throw (DivisionByZeroException)  
{  
    // throw exception if denominator 0  
    if (d == 0)  
        throw DivisionByZeroException ();  
    if (d < 0)  
    {  
        numerator = -n;  
        denominator = -d;  
    }  
    else  
    {  
        numerator = n;  
        denominator = d;  
    }  
    reduce ();  
}
```

Fraction source file continued

```
Fraction :: Fraction (const Fraction& v)
{
    numerator = v.numerator;
    denominator = v.denominator;
}

Fraction& Fraction :: operator= (const Fraction& v)
{
    numerator = v.numerator;
    denominator = v.denominator;
    return *this;
}

bool Fraction :: operator== (const Fraction& v) const
{
    return numerator == v.numerator && denominator == v.denominator;
}
```

Fraction source file continued

```
bool Fraction::operator!= (const Fraction& v) const
{
    return !operator== (v);
}

void Fraction::reduce (void)
{
    unsigned int teiler = gcf (abs (numerator), abs (denominator));

    numerator /= (int) teiler;
    denominator /= (int) teiler;
}

Fraction operator+ (const Fraction& a, const Fraction& b)
{
    unsigned int v = lcd (abs (a.denominator), abs (b.denominator));

    Fraction r (a.numerator * ((int) v / a.denominator) + b.numerator * ((int) v / b.denominator), v);
    r.reduce ();
    return r;
}
```

Fraction source file continued

```
void Fraction::reduce (void)
{
    unsigned int teiler = gcf (abs (numerator), abs (denominator));

    numerator /= (int) teiler;
    denominator /= (int) teiler;
}

Fraction operator+ (const Fraction& a, const Fraction& b)
{
    unsigned int v = lcd (abs (a.denominator), abs (b.denominator));

    Fraction r (a.numerator * ((int) v / a.denominator) + b.numerator * ((int) v / b.denominator), v);
    r.reduce ();
    return r;
}

Fraction operator- (const Fraction& a, const Fraction& b)
{
    unsigned int v = lcd (abs (a.denominator), abs (b.denominator));

    Fraction r (a.numerator * ((int) v / a.denominator) - b.numerator * ((int) v / b.denominator), v);
    r.reduce ();
    return r;
}

ostream& operator<< (ostream& out, const Fraction& v)
{
    return out << v.numerator << "/" << v.denominator;
}
```

Fraction source file continued

```
// CppUnit-Tutorial
// file: fractiontest.h
#ifndef FRACTIONTEST_H
#define FRACTIONTEST_H

#include <cppunit/TestFixture.h>
#include <cppunit/extensions/HelperMacros.h>
#include "Fraction.h"

using namespace std;

class fractiontest : public CPPUNIT_NS :: TestFixture
{
    CPPUNIT_TEST_SUITE (fractiontest);
    CPPUNIT_TEST (addTest);
    CPPUNIT_TEST (subTest);
    CPPUNIT_TEST (exceptionTest);
    CPPUNIT_TEST (equalTest);
    CPPUNIT_TEST_SUITE_END ();

public:
    void setUp (void);
    void tearDown (void);

protected:
    void addTest (void);
    void subTest (void);
    void exceptionTest (void);
    void equalTest (void);

private:
    Fraction *a, *b, *c, *d, *e, *f, *g, *h;
};

#endif
```

FractionTest Header

◆ TextFixture

- Wrapper class which servers to create the basic conditions that are needed for the test suite

◆ HelperMacros

- Macros which help eliminate extra coding

- CPPUNIT_TEST_SUITE

 - Begins the test suite

- CPPUNIT_TEST_SUITE_END

 - Ends the test suite

- CPPUNIT_TEST

 - Macro that helps integrate our tests into the suite

```
// CppUnit-Tutorial
// file: fractiontest.cc
#include "fractiontest.h"

CPPUNIT_TEST_SUITE_REGISTRATION (fractiontest);

void fractiontest :: setUp (void)
{
    // set up test environment (initializing objects)
    a = new Fraction (1, 2);
    b = new Fraction (2, 3);
    c = new Fraction (2, 6);
    d = new Fraction (-5, 2);
    e = new Fraction (5, -2);
    f = new Fraction (-5, -2);
    g = new Fraction (5, 2);
    h = new Fraction ();
}

void fractiontest :: tearDown (void)
{
    // finally delete objects
    delete a; delete b; delete c; delete d;
    delete e; delete f; delete g; delete h;
}

void fractiontest :: addTest (void)
{
    // check subtraction results
    CPPUNIT_ASSERT_EQUAL (*a + *b, Fraction (7, 6));
    CPPUNIT_ASSERT_EQUAL (*b + *c, Fraction (1));
    CPPUNIT_ASSERT_EQUAL (*d + *e, Fraction (-5));
    CPPUNIT_ASSERT_EQUAL (*e + *f, Fraction (0));
    CPPUNIT_ASSERT_EQUAL (*h + *c, Fraction (2, 6));
    CPPUNIT_ASSERT_EQUAL (*a + *b + *c + *d + *e + *f + *g + *h, Fraction (3, 2));
}
```

FractionTest source file

```
void fractiontest :: subTest (void)
{
    // check addition results
    CPPUNIT_ASSERT_EQUAL (*a - *b, Fraction (-1, 6));
    CPPUNIT_ASSERT_EQUAL (*b - *c, Fraction (1, 3));
    CPPUNIT_ASSERT_EQUAL (*b - *c, Fraction (2, 6));
    CPPUNIT_ASSERT_EQUAL (*d - *e, Fraction (0));
    CPPUNIT_ASSERT_EQUAL (*d - *e - *f - *g - *h, Fraction (-5));
}

void fractiontest :: exceptionTest (void)
{
    // an exception has to be thrown here
    CPPUNIT_ASSERT_THROW (Fraction (1, 0), DivisionByZeroException);
}

void fractiontest :: equalTest (void)
{
    // test successful, if true is returned
    CPPUNIT_ASSERT (*d == *e);
    CPPUNIT_ASSERT (Fraction (1) == Fraction (2, 2));
    CPPUNIT_ASSERT (Fraction (1) != Fraction (1, 2));
    // both must have equal valued
    CPPUNIT_ASSERT_EQUAL (*f, *g);
    CPPUNIT_ASSERT_EQUAL (*h, Fraction (0));
    CPPUNIT_ASSERT_EQUAL (*h, Fraction (0, 1));
}
```

FractionTest source file continued

```
#!/ CppUnit-Tutorial
// file: ftest.cc
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>

int main (int argc, char* argv[])
{
    // informs test-listener about testresults
    CPPUNIT_NS :: TestResult testresult;

    // register listener for collecting the test-results
    CPPUNIT_NS :: TestResultCollector collectedresults;
    testresult.addListener (&collectedresults);

    // insert test-suite at test-runner by registry
    CPPUNIT_NS :: TestRunner testrunner;
    testrunner.addTest (CPPUNIT_NS :: TestFactoryRegistry :: getRegistry ().makeTest ());
    testrunner.run (testresult);

    // output results in compiler-format
    CPPUNIT_NS :: CompilerOutputter compileroutputter (&collectedresults, std::cerr);
    compileroutputter.write ();

    // return 0 if tests were successful
    return collectedresults.wasSuccessful () ? 0 : 1;
}
```

Our main function, ftest

- ◆ **TestResult**
 - Acts as an event manager
 - Informs the listener (TestResultCollector) which instance of tests it needs to listen to for the results
- ◆ **TestResultCollector**
 - Will be informed about the test process after being added to the manager
 - Listens for data

- ◆ TestRunner
 - Executes all of the tests that are on the registry
- ◆ CompilerOutputter
 - Displays the results of each test to look like a compiler
 - If you were running an IDE you could click the line number and go to the failed test
- ◆ 0 is returned if all the tests are passed
- ◆ 1 is returned if one or more of the tests failed

```
# CppUnit-Tutorial
# file: makefile
# next line has to be changed to the installation-path of CppUnit
CPPUNIT_PATH=/home/ehagen

ftest: ftest.o fractiontest.o Fraction.o
    gcc -o ftest ftest.o fractiontest.o Fraction.o -L${CPPUNIT_PATH}/lib -lstdc++ -lcppunit -ldl

Fraction.o: Fraction.cc Fraction.h
    gcc -c Fraction.cc

fractiontest.o: fractiontest.cc
    gcc -c fractiontest.cc -I${CPPUNIT_PATH}/include

ftest.o: ftest.cc
    gcc -c ftest.cc -I${CPPUNIT_PATH}/include

clean:
    rm -f *.o ftest
```

Makefile to create ftest executable

```
bash-3.00$ make
gcc -c fractiontest.cc -I/home/ehagen/include
gcc -c Fraction.cc
gcc -o ftest ftest.o fractiontest.o Fraction.o -L/home/ehagen/lib -lstdc++ -lcppunit -ldl
[4]+  Done                  emacs ftest.cc
bash-3.00$ ./ftest
OK (4)
```

Command line entries

```
void fractiontest :: exceptionTest (void)
{
    // an exception has to be thrown here
    CPPUNIT_ASSERT_THROW (Fraction (1, 1), DivisionByZeroException);
}
```

Intentional fail of exceptionTest

CppUnit – What if we fail?

```
bash-3.00$ make
gcc -c fractiontest.cc -I/home/ehagen/include
gcc -o ftest ftest.o fractiontest.o Fraction.o -L/home/ehagen/lib -lstdc++ -lcppunit -ldl
bash-3.00$ ./ftest
fractiontest.cc:51:Assertion
Test name: fractiontest::exceptionTest
assertion failed
- Expected exception: DivisionByZeroException not thrown.

Failures !!!
Run: 4   Failure total: 1   Failures: 1   Errors: 0
bash-3.00$ █
```

ExceptionTest fails as expected

```
// arithmetic operators
friend Fraction operator+ (const Fraction&, const Fraction&);
friend Fraction operator- (const Fraction&, const Fraction&);
friend Fraction operator* (const Fraction&, const Fraction&);

// output on stdout
friend ostream& operator<< (ostream&, const Fraction&);
```

Addition of a multiplication operator to the Fraction header class

```
Fraction operator* (const Fraction& a, const Fraction& b)
{
    Fraction r (a.numerator * b.numerator, a.denominator * b.denominator);
    r.reduce ();
    return r;
}
```

Addition of multiplication operator to
the Fraction class source

```
class fractiontest : public CPPUNIT_NS :: TestFixture
{
    CPPUNIT_TEST_SUITE (fractiontest);
    CPPUNIT_TEST (addTest);
    CPPUNIT_TEST (subTest);
    CPPUNIT_TEST (multTest);
    CPPUNIT_TEST (exceptionTest);
    CPPUNIT_TEST (equalTest);
    CPPUNIT_TEST_SUITE_END ();

public:
    void setUp (void);
    void tearDown (void);

protected:
    void addTest (void);
    void subTest (void);
    void multTest (void);
    void exceptionTest (void);
    void equalTest (void);

private:
    Fraction *a, *b, *c, *d, *e, *f, *g, *h;
};
```

Addition of multTest to the
FractionTest header

```
void fractiontest :: subTest (void)
{
    // check addition results
    CPPUNIT_ASSERT_EQUAL (*a - *b, Fraction (-1, 6));
    CPPUNIT_ASSERT_EQUAL (*b - *c, Fraction (1, 3));
    CPPUNIT_ASSERT_EQUAL (*b - *c, Fraction (2, 6));
    CPPUNIT_ASSERT_EQUAL (*d - *e, Fraction (0));
    CPPUNIT_ASSERT_EQUAL (*d - *e - *f - *g - *h, Fraction (-5));
}

void fractiontest :: multTest (void)
{
    // check multiplication results
    CPPUNIT_ASSERT_EQUAL (*a * *b, Fraction (2, 6));
    CPPUNIT_ASSERT_EQUAL (*a * *b, Fraction (1, 3));
    CPPUNIT_ASSERT_EQUAL (*b * *c, Fraction (2, 9));
    CPPUNIT_ASSERT_EQUAL (*c * *d, Fraction (-5, 6));
    CPPUNIT_ASSERT_EQUAL (*g * *h, Fraction (0, 1));
    CPPUNIT_ASSERT_EQUAL (*g * *h, Fraction (0, 5));
}

void fractiontest :: exceptionTest (void)
{
    // an exception has to be thrown here
    CPPUNIT_ASSERT_THROW (Fraction (1, 0), DivisionByZeroException);
}
```

Addition of multTest to FunctionTest
source

Making the new test suite

```
bash-3.00$ make
gcc -c fractiontest.cc -I/home/ehagen/include
gcc -c Fraction.cc
gcc -o ftest ftest.o fractiontest.o Fraction.o -L/home/ehagen/lib -lstdc++ -lcppunit -ldl
[1]- Done          emacs fractiontest.cc
[5]+ Done          emacs fractiontest.cc
bash-3.00$ ./ftest
OK (5)
bash-3.00$ █
```

MultTest successfully runs, along with the other four tests. Notice, no changes were needed in the Makefile or the main function (ftest)

What if we fail, again?

```
void fractiontest :: multTest (void)
{
    // check multiplication results
    CPPUNIT_ASSERT_EQUAL (*a * *b, Fraction (2, 7));
    CPPUNIT_ASSERT_EQUAL (*a * *b, Fraction (1, 3));
    CPPUNIT_ASSERT_EQUAL (*b * *c, Fraction (2, 9));
    CPPUNIT_ASSERT_EQUAL (*c * *d, Fraction (-5, 6));
    CPPUNIT_ASSERT_EQUAL (*g * *h, Fraction (0, 1));
    CPPUNIT_ASSERT_EQUAL (*g * *h, Fraction (0,5));
}
```

First assert set to intentionally fail in the FunctionTest source,
*a * *b should be 1/3 not 2/7

What if we fail, again?

```
bash-3.00$ make
gcc -c fractiontest.cc -I/home/ehagen/include
gcc -o ftest ftest.o fractiontest.o Fraction.o -L/home/ehagen/lib -lstdc++ -lcppunit -ldl
bash-3.00$ ./ftest
fractiontest.cc:51:Assertion
Test name: fractiontest::multTest
equality assertion failed
- Expected: 1/3
- Actual   : 2/7

Failures !!!
Run: 5   Failure total: 1   Failures: 1   Errors: 0
bash-3.00$ █
```

MultTest fails as expected but the other four tests run correctly

What can CppUnit do for us?

- ◆ CppUnit is a flexible framework for unit testing and should be adaptable to many situations
- ◆ To test CppUnit two real GPSTk classes were tested within its framework
- ◆ Triple
 - A mathematical class which provides three dimensional vector operations
- ◆ DayTime
 - A GPS class which can convert between many formats of time

- ◆ Triple has 3 constructors 7 operators and 9 member functions
- ◆ We want to test to ensure the quality of this class to the best of our ability while using minimal time and energy
- ◆ To the best of our knowledge, there was no previous testing implemented for Triple
- ◆ CppUnit's framework provides a standard base from which a test suite can be built

- ◆ In the TripTest test suite there are 12 separate test cases each with assertions to validate many different paths
- ◆ The TripTest header is clear and concise
- ◆ The TripTest source is explicit and modular
- ◆ Each member function is tested separately
- ◆ During the course of this testing three errors in the class Triple were found

```
bash-3.00$ ./Ttest
TripTest.cpp:85:Assertion
Test name: TripTest::unitvTest
equality assertion failed
- Expected: (nan, nan, nan)
- Actual  : (0, 0, 0)

TripTest.cpp:96:Assertion
Test name: TripTest::cosvTest
equality assertion failed
- Expected: nan
- Actual  : 1

TripTest.cpp:139:Assertion
Test name: TripTest::azTest
assertion failed
- Expected exception: CPPUNIT_NS::Exception not thrown

Failures !!!
Run: 12   Failure total: 3   Failures: 3   Errors: 0
```

- ◆ The DayTime class converts between different GPS time formats
- ◆ DayTimeConversionTest was a previously implemented test in which the developer created their own framework to test the DayTime class
- ◆ DayTimeConvTest is a DayTime test within the CppUnit framework which is designed to mimic DayTimeConversionTest
- ◆ The purpose of DayTimeConvTest is to prove that CppUnit is flexible and versatile

- ◆ DayTimeConvTest succeeds at mimicking DayTimeConversionTest, all tests ran successfully
- ◆ DayTimeConvTest has three tests within its test suite for construction, mutation, and random dates
- ◆ DayTimeConvTest is clear and modular and, because it was built within a framework for unit testing, is easier for other developers to understand when compared to a test within its own unique framework

GPSTk Class Testing - Day Time

```
bash-3.00$ make
make: `DayTest' is up to date.
bash-3.00$ ./DayTest
January 06, 1980 00:00:00.0000000000000000
August 22, 1999 00:00:00.0000000000000000
January 01, 2000 12:00:00.0000000000000000
January 06, 1980 00:00:00.0000000000000000
August 22, 1999 00:00:00.0000000000000000
January 01, 2000 12:00:00.0000000000000000
October 21, 2008 18:06:20.021072924137115
August 05, 2000 20:59:47.115217745304108
July 12, 2001 10:56:48.639393746852875
June 27, 2007 17:30:28.454549610614777
January 19, 2007 22:43:42.727905213832855
March 23, 1999 21:53:23.528550267219543
October 10, 2009 23:02:12.498590648174286
August 24, 2004 03:02:40.644029080867767
November 10, 1996 01:32:21.421723365783691
June 23, 1996 20:59:02.714774608612061
September 03, 2008 18:18:18.607237637042999
November 09, 2007 03:15:36.733563244342804
June 03, 2000 13:21:53.398334383964539
December 15, 2005 17:08:24.519103467464447
April 08, 2003 12:18:04.847585856914520
July 20, 2012 02:29:15.049355328083038
May 04, 2000 03:46:28.333015143871307
July 27, 2014 14:14:22.607847154140472
April 08, 2007 14:27:59.499734938144684
June 16, 2001 05:11:00.235740244388580
OK (3)
bash-3.00$ █
```

- ◆ For meaningful equivalence assertions, it seems an equivalence operator is needed within the class
- ◆ At an initial glance, it does not seem possible to pass parameters to the testing function directly
- ◆ Normally, CppUnit finds divisions by zero and records them as NaNs but in one case, for unknown reasons, this was not the case

- ◆ We were able to test an existing class, without modifications, for stability and reliability
- ◆ We were able to add functionality to the Fraction class and then easily test that new functionality by adding to our test suite
- ◆ We were able to test two GPSTK classes and were able to identify errors within class code
- ◆ CppUnit was able to successfully imitate DayTimeConversionTest within our framework
- ◆ CppUnit gives the developer the power to assure quality code and product to the user