Basic C++

CppUTest

Dr. Porkoláb Zoltán Károly

gsd@inf.elte.hu

http://gsd.web.elte.hu

The xUnit family

- Automated testing software framework
- Common ancestor: Sunit (1989, Kent Back for Smalltalk)
- Ported to many modern languages
  - Junit
  - Runit
  - xUnix, Nunit, MSTest (for .Net)
  - GoogleTest
  - CppUTest
- Black-box testing and mocking

The xUnit family

- Sharing common architecture
  - Test case: smallest part a test
  - Assertions: validate expected results
  - Test suite: collection of related test cases, sharing a framework to reuse environment for setup and teardown
  - Text fixture/text context: the environment around the test
  - Test runner: runs the test and produce output

First CppUTest

```
#include "CppUTest/CommandLineTestRunner.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 1);
}

int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);
}
```

First CppUTest

```
#include "CppUTest/CommandLineTestRunner.h"

TEST_GROUP(FirstTestGroup) { };   // test suit

TEST(FirstTestGroup, FirstTest)   // test case
{
   CHECK_EQUAL(1, 1);             // assertion
}

int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);   // test runner
}
```

First CppUTest

```
#include "CppUTest/CommandLineTestRunner.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 1);
}

int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);
}

$ g++ -Wall -Wextra first.cpp -lCppUTest -o first
$
```

First CppUTest

```
#include "CppUTest/CommandLineTestRunner.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 1);
}

int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);
}

$ g++ -Wall -Wextra first.cpp -lCppUTest -o first
$ ./first
```

First CppUTest

```
#include "CppUTest/CommandLineTestRunner.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 1);
}

int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);
}

$ g++ -Wall -Wextra first.cpp -lCppUTest -o first
$ ./first
.
```

First CppUTest

```
#include "CppUTest/CommandLineTestRunner.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 1);
}

int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);
}

$ g++ -Wall -Wextra first.cpp -lCppUTest -o first
$ ./first
.
OK (1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)
```

Real CppUTest setup

```
// second.cpp
#include "CppUTest/TestHarness.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 1);
}
//-----------------------------------------------
// test.cpp
#include "CppUTest/CommandLineTestRunner.h"
int main(int ac, char** av)
{
   return CommandLineTestRunner::RunAllTests(ac, av);
}

$ g++ -Wall -Wextra second.cpp
$ g++ -Wall -Wextra test.cpp
$ g++ second.o test.o -lCppUTest -o second
$ ./second
.
OK (1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)
```

Fail CppUTest test

```
// second-fail.cpp
#include "CppUTest/TestHarness.h"

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(1, 0);
}

$ g++ -Wall -Wextra second-fail.cpp test.o -lCppUTest -o second-fail
$ ./second-fail
second-fail.cpp:6: error: Failure in TEST(FirstTestGroup, FirstTest)
    expected <1>
    but was  <0>
    difference starts at position 0 at: <          0         >
                                        ^


.
Errors (1 failures, 1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)
```

General requirements for testing

- Correctness
- Readability
- Completeness
- Demonstrativeness
- Resilience

Correctness

```
#include "CppUTest/TestHarness.h"

int square( int x)
{
  return 0;  // TODO: implementing
}

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(0, square(0));



}

$ g++ square.cpp test.o -lCppUTest -o square
$ ./square
.
OK (1 tests, 1 ran, 1 checks, 0 ignored, 0 filtered out, 0 ms)
```

Correctness

```
#include "CppUTest/TestHarness.h"

int square( int x)
{
  return 0;  // TODO: implementing
}

TEST_GROUP(FirstTestGroup) { };

TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(0, square(0));
   CHECK_EQUAL(1, square(1));
   CHECK_EQUAL(2, square(2));
   CHECK_EQUAL(42,square(42));
}

$ g++ square.cpp test.o -lCppUTest -o square
$ ./square
.
OK (1 tests, 1 ran, 4 checks, 0 ignored, 0 filtered out, 0 ms)
```

Correctness

```
#include "CppUTest/TestHarness.h"

int square( int x)
{
  return x*x;  // TODO: implementing
}
TEST_GROUP(FirstTestGroup) { };
TEST(FirstTestGroup, FirstTest)
{
   CHECK_EQUAL(0, square(0));
   CHECK_EQUAL(1, square(1));
   CHECK_EQUAL(4, square(2));
   CHECK_EQUAL(1764, square(42));
}
$ ./square
.
OK (1 tests, 1 ran, 4 checks, 0 ignored, 0 filtered out, 0 ms)
```

Correctness

- Never depend on unimplemented code or on bug!
- Code review can detect such code
- If the test fails we should know who to blame!
  - Author of the code?
  - Writer of the test?

Readability

```
#include "CppUTest/TestHarness.h"

TEST_GROUP(StorageTestGroup) { };

TEST(StorageTestGroup, Start)
{
   TestStorageSystem storage;
  auto testData = getTestDileMap();
  storage.MapFilesystem(test_data);
  BigSystem system;

  CHECK( system.initialize(5));

  ThreadPool pool(10);
  pool.startThreads();
  storage.setThreads(pool);
  system.setStorage(storage);

  CHECK_EQUAL( 42, system.PrivateKey());
}
```

Readability

```
#include "CppUTest/TestHarness.h"

TEST_GROUP(StorageTestGroup) { };

TEST(StorageTestGroup, Start)
{









  BigSystem system = initTestSystemAndTestData();

  CHECK_EQUAL( 42, system.PrivateKey());
}
```

Readability

- Test should be obvious for future reader
  - Including yourself
- Typical mistakes
  - Too much boilerplate code
  - Not enough context
  - Use the most simple test features you can
- A test is like a novel
  - Setup (Build up the test environment)
  - Action (Actual assertions)
  - Conclusion (TearDown)

Completeness

```
#include "CppUTest/TestHarness.h"




TEST_GROUP(FactorialTestGroup) { };

TEST(FactorialTestGroup, FirstTest)
{
   CHECK_EQUAL(1,   factorial(1));
   CHECK_EQUAL(120, factorial(5));




}

$ ./factorial-test
.
OK (1 tests, 1 ran, 2 checks, 0 ignored, 0 filtered out, 0 ms)
```

Completeness

```
#include "CppUTest/TestHarness.h"

int factorial( int x)
{
  return 1==x ? 1 : 120;
}

TEST_GROUP(FactorialTestGroup) { };

TEST(FactorialTestGroup, FirstTest)
{
   CHECK_EQUAL(1,   factorial(1));
   CHECK_EQUAL(120, factorial(5));




}

$ ./factorial-test
.
OK (1 tests, 1 ran, 2 checks, 0 ignored, 0 filtered out, 0 ms)
```

Completeness

```
#include "CppUTest/TestHarness.h"
#include <limits>

int factorial( int x)
{
  return x <= 1 ? 1 : x * factorial(x-1);
}

TEST_GROUP(FactorialTestGroup) { };

TEST(FactorialTestGroup, FirstTest)
{
   CHECK_EQUAL(  1, factorial(1));
   CHECK_EQUAL(120, factorial(5));
   CHECK_EQUAL(  1, factorial(0) );
   CHECK_EQUAL(479001600, factorial(12) );

   CHECK_EQUAL( std::numeric_limits<int>::max(), factorial(13) ); // overflow



}
```

Completeness

```
#include "CppUTest/TestHarness.h"
#include <limits>

int factorial( int x)
{
  return x <= 1 ? 1 : x * factorial(x-1);
}

TEST_GROUP(FactorialTestGroup) { };

TEST(FactorialTestGroup, FirstTest)
{
   CHECK_EQUAL(  1, factorial(1));
   CHECK_EQUAL(120, factorial(5));
   CHECK_EQUAL(  1, factorial(0) );
   CHECK_EQUAL(479001600, factorial(12) );

   CHECK_EQUAL( std::numeric_limits<int>::max(), factorial(13) ); // overflow

   CHECK_EQUAL( 1,   factorial(0) );    // check: no internal state
   CHECK_EQUAL( 120, factorial(5) );
}
```

Completeness

```
std::list<int> li = { 1, 3, 5, ... };


bool less55_3rd(int x)
{
  static int cnt = 0;
  if ( x < 55 )    ++cnt;
  return 3 == cnt;
}

// find the third occurrence of value less than 55
auto it = find_if( li.begin(), li.end(), less55_3rd);

if ( li.end() != it )
{
  *it = 56;
   it = find_if( ++it, li.end(), less55_3rd);  // works?
}
```

Completeness

```
#include "CppUTest/TestHarness.h"
#include <limits>

int factorial( int x)
{
  return x <= 1 ? 1 : x * factorial(x-1);
}
TEST_GROUP(FactorialTestGroup) { };
TEST(FactorialTestGroup, BasicTest)
{
   CHECK_EQUAL(  1, factorial(1));
   CHECK_EQUAL(120, factorial(5));
   CHECK_EQUAL(  1, factorial(0) );
   CHECK_EQUAL(479001600, factorial(12) );
}
TEST(FactorialTestGroup, OverflowTest)
{
   CHECK_EQUAL(std::numeric_limits<int>::max(), factorial(13)); // overflow
}
TEST(FactorialTestGroup, StatelessTest)
{
   CHECK_EQUAL(   1, factorial(0) );    // check: no internal state
   CHECK_EQUAL( 120, factorial(5) );
}
```

Completeness

```
$ ./factorial-test
.
2-factorial.cpp:18: error: Failure in TEST(FactorialTestGroup, OverflowTest)
    expected <2147483647>
    but was  <1932053504>
    difference starts at position 0 at: <          1932053504>
                                         ^


..
Errors (1 failures, 3 tests, 3 ran, 7 checks, 0 ignored, 0 filtered out, 0 ms)
```

Completeness

```
$ ./factorial-test
.
2-factorial.cpp:18: error: Failure in TEST(FactorialTestGroup, OverflowTest)
    expected <2147483647>
    but was  <1932053504>
    difference starts at position 0 at: <          1932053504>
                                         ^


..
Errors (1 failures, 3 tests, 3 ran, 7 checks, 0 ignored, 0 filtered out, 0 ms)

$ ./factorial-test -c
.
2-factorial.cpp:18: error: Failure in TEST(FactorialTestGroup, OverflowTest)
    expected <2147483647>
    but was  <1932053504>
    difference starts at position 0 at: <          1932053504>
                                         ^


..
Errors (1 failures, 3 tests, 3 ran, 7 checks, 0 ignored, 0 filtered out, 0 ms)
```

Completeness

```
$ ./factorial-test
.
2-factorial.cpp:18: error: Failure in TEST(FactorialTestGroup, OverflowTest)
    expected <2147483647>
    but was  <1932053504>
    difference starts at position 0 at: <          1932053504>
                                         ^


..
Errors (1 failures, 3 tests, 3 ran, 7 checks, 0 ignored, 0 filtered out, 0 ms)


$ ./factorial-test -c -xn OverflowTest
..
OK (3 tests, 2 ran, 6 checks, 0 ignored, 1 filtered out, 0 ms)
```

Completeness

```
#include <vector>
#include "CppUTest/TestHarness.h"

void insert( std::vector<int> &v, int x)
{
    v.push_back(x);
}

TEST_GROUP(InsertTestGroup) { };

TEST(InsertTestGroup, FirstTest)
{
   std::vector<int> v;
   CHECK_EQUAL(  0, v.size());
   int x = 42;
   CHECK_EQUAL(  42, x);
   insert(v,x);
   CHECK_EQUAL(  1, v.size() );
   CHECK_EQUAL( 42, v.back() );
}
```

Completeness

- Cover as many features as we can
- Typical mistakes
  - Write test only for the easy cases
  - Not covering edge cases
  - Not covering error handling
  - Not checking statefull/stateless behavior
- Test only what we are responsible for

Demonstrability

- Many times clients learn the system via tests
- Test should demonstrate how the API works
- Typical mistakes
  - Using private API
  - Using friends (e.g.) for build up the test environment
- Comment, if the test violates the expected API use

Resilience

- Write tests that will long term stable
- Depend only on published API guarantees
- Typical mistakes
  - Flaky tests (re-run gets different results)
  - Brittle tests (depends too many assumptions, impl. Details)
  - Tests depending on the execution order
  - Non-hermetic tests
  - Mocks depending on underlying APIs

Flaky tests

```
#include <thread>
#include "CppUTest/TestHarness.h"



TEST_GROUP(InsertTestGroup) { };

TEST(InsertTestGroup, FirstTest)
{
   Updater u;
   u.updateAsync();

   // 500ms should be enough
   std::this_thread::sleep_for(std::chrono::milliseconds(500));

   CHECK( u.updated() );
}


// RotatingLogFile
```

Assertions

```
CHECK(boolean condition)            - checks any boolean result
CHECK_TEXT(boolean condition, text) - same as above, prints text on failure
CHECK_FALSE(condition)              - checks any boolean result

CHECK_EQUAL(expected, actual) - checks for equality between entities using ==
CHECK_COMPARE(first, relop, second) - checks thats a relational operator holds

LONGS_EQUAL(expected, actual)          - compares two numbers
UNSIGNED_LONGS_EQUAL(expected, actual) - compares two positive numbers
BYTES_EQUAL(expected, actual)          - compares two numbers, eight bits wide
POINTERS_EQUAL(expected, actual)       - compares two pointers
DOUBLES_EQUAL(expected, actual, tolerance) - compares two floating point
FUNCTIONPOINTERS_EQUAL(expected, actual)   - compares two void (*)() function

BITS_EQUAL(expected, actual, mask)   - compares expected to actual bit by bit,
MEMCMP_EQUAL(expected, actual, size) - compares two areas of memory

STRCMP_EQUAL(expected, actual) - checks const char* strings using strcmp()
STRNCMP_EQUAL(expected, actual, length)  - checks strings  using strncmp()
STRCMP_NOCASE_EQUAL(expected, actual) - checks strings, not considering case
STRCMP_CONTAINS(expected, actual) - checks whether actual contains expected

Most of them have _TEXT version to print message
```