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

The CppUTest framework

- xUnit based test framework
  - For C and C++
  - Written in C++
  - Frequently used in embedded systems
  - Portable to old and new platforms
  - Simple to use
  - Small
  - Available as Linux package: cpptest
- https://cpputest.github.io/manual.html
- https://github.com/cpputest/cpputest

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)
```

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
```

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 "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) );
}
IGNORE_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) );
}
.!.
OK (3 tests, 2 ran, 6 checks, 1 ignored, 0 filtered out, 0 ms)
```

Execution flags

```
-c  colorize output, print green if OK, or red if failed
-g  group only run test whose group contains the substring group
-k  package name, Add a package name in JUnit output
-lg print a list of group names, separated by spaces
-ln print a list of test names in the form of group.name, separated by spaces
-n  name only run test whose name contains the substring name
-ojunit    output to JUnit ant plugin style xml files (for CI systems)
-oteamcity output to xml files (as the name suggests, for TeamCity)
-p  run tests in a separate process.
-r# repeat the tests some number (#) of times, or twice if # is not specified.
-sg group only run test whose group exactly matches the string group
-sn name only run test whose name exactly matches the string name
-v  verbose, print each test name as it runs
-xg group exclude tests whose group contains the substring group (v3.8)
-xn name exclude tests whose name contains the substring name (v3.8)
"TEST(group, name)"   only run test whose group and name matches the strings
```

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
```

Brittle tests

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

void add(std::unordered_set<int> &s, int x)
{
  s.insert(x);
}

TEST_GROUP(HashSetTestGroup) { };
TEST(HashSetTestGroup, InsertTest)
{
   std::unordered_set<int> s;
   add(s,1);
   add(s,2);
   add(s,3);
   CHECK_EQUAL(1, *s.begin());
}
```

Brittle tests

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

void add(std::unordered_set<int> &s, int x)
{
  s.insert(x);
}

TEST_GROUP(HashSetTestGroup) { };
TEST(HashSetTestGroup, InsertTest)
{
   std::unordered_set<int> s;
   add(s,1);
   add(s,2);
   add(s,3);
   CHECK_EQUAL(1, *s.begin());
}
```

```
2-brittle.cpp:16: error: Failure in TEST(HashSetTestGroup, InsertTest)
    expected <1>
    but was  <3>
    difference starts at position 0 at: <          3          >
                                                   ^
```

Brittle tests

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

void add(std::unordered_set<int> &s, int x)
{
  s.insert(x);
}

TEST_GROUP(HashSetTestGroup) { };
TEST(HashSetTestGroup, InsertTest)
{
   std::unordered_set<int> s;
   add(s,1);
   add(s,2);
   add(s,3);
   CHECK_EQUAL(1, *s.begin());
}
```

Brittle tests

```
#include "CppUTest/TestHarness.h"    // swap header file order
#include <unordered_set>

void add(std::unordered_set<int> &s, int x)
{
  s.insert(x);
}

TEST_GROUP(HashSetTestGroup) { };
TEST(HashSetTestGroup, InsertTest)
{
   std::unordered_set<int> s;
   add(s,1);
   add(s,2);
   add(s,3);
   CHECK_EQUAL(1, *s.begin());
}
```

Brittle tests

```cpp
#include "CppUTest/TestHarness.h"
#include <unordered_set>

void add(std::unordered_set<int>&s, int x)
{
  s.insert(x);
}

TEST_GROUP(HashSetTestGroup) { }
TEST(HashSetTestGroup, InsertTest)
{
   std::unordered_set<int> s;
   add(s,1);
   add(s,2);
   add(s,3);
   CHECK_EQUAL(1, *s.begin());
}
```

```
$ g++ -Wextra brittle.cpp test.cpp -std=c++20 -lCppUTest
In file included from /usr/include/c++/11/bits/hashtable.h:35,
                 from /usr/include/c++/11/unordered_set:46,
                 from brittle.cpp:2:
/usr/include/c++/11/bits/hashtable_policy.h: In member function ‘void
std::__detail::_Local_iterator_base<_Key, _Value, _ExtractKey, _Hash, _RangeHash, _Unused,
false>::_M_init(const __hash_code_base&)’:
/usr/include/c++/11/bits/hashtable_policy.h:1423:15: error: expected type-specifier before
‘this’
 1423 |       { ::new(this->_M_h()) __hash_code_base(__base); }
      |               ^~~~
/usr/include/c++/11/bits/hashtable_policy.h:1423:15: error: expected ‘)’ before ‘this’
/usr/include/c++/11/bits/hashtable_policy.h:1423:14: note: to match this ‘(’
 1423 |       { ::new(this->_M_h()) __hash_code_base(__base); }
      |              ^
/usr/include/c++/11/bits/hashtable_policy.h: In member function
‘std::__detail::_Hashtable_alloc<_NodeAlloc>::__node_type*
std::__detail::_Hashtable_alloc<_NodeAlloc>::_M_allocate_node(_Args&& ...)’:
/usr/include/c++/11/bits/hashtable_policy.h:1877:20: error: expected type-specifier before ‘(’
token
 1877 |             ::new ((void*)__n) __node_type;
      |                    ^
/usr/include/c++/11/bits/hashtable_policy.h:1877:20: error: expected ‘)’ before ‘(’ token
 1877 |             ::new ((void*)__n) __node_type;
      |                   ~^
      |                    )
/usr/include/c++/11/bits/hashtable_policy.h:1877:21: error: expected primary-expression before
‘void’
 1877 |             ::new ((void*)__n) __node_type;
      |                     ^~~~
In file included from /usr/include/c++/11/bits/hashtable.h:38,
                 from /usr/include/c++/11/unordered_set:46,
                 from brittle.cpp:2:
/usr/include/c++/11/bits/node_handle.h: In member function ‘void std::_Node_handle_common<_Val,
_NodeAlloc>::_M_move(std::_Node_handle_common<_Val, _NodeAlloc>&&)’:
/usr/include/c++/11/bits/node_handle.h:154:16: error: expected type-specifier
  154 |         ::new (std::__addressof(_M_alloc)) _NodeAlloc(__nh._M_alloc.release());
      |                ^~~
/usr/include/c++/11/bits/node_handle.h:154:16: error: expected ‘)’
/usr/include/c++/11/bits/node_handle.h:154:15: note: to match this ‘(’
  154 |         ::new (std::__addressof(_M_alloc)) _NodeAlloc(__nh._M_alloc.release());
      |
```

Brittle tests

```
#include <iostream>
#include <sstream>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os)
{
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}

TEST_GROUP(LogTestGroup) { };

TEST(LogTestGroup, LogContent)
{
   log(std::cerr);
   std::ostringstream os{};
   log(os);
   STRCMP_CONTAINS("4-brittle.cpp", os.str().c_str());
   STRCMP_CONTAINS("7",os.str().c_str());
}
```

Order of tests

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

void increment(int& x)
{
    ++x;
}
static int count = 0;

TEST_GROUP(CountTestGroup) { };

TEST(CountTestGroup, First)
{
   increment(count);
   CHECK_EQUAL( 1, count );
}

TEST(CountTestGroup, Second)
{
   increment(count);
   CHECK_EQUAL( 2, count );
}
```

Hermetic isolation

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os)
{
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}

TEST_GROUP(LogTestGroup) { };

TEST(LogTestGroup, LogContent)
{

   std::ofstream logfile{"logfile.txt"};
   log(logfile);
   CHECK( std::filesystem::exists("logfile.txt"));
}
```

Hermetic isolation

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os)
{
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}

TEST_GROUP(LogTestGroup) { };

TEST(LogTestGroup, LogContent)
{
   std::filesystem::remove("logfile.txt");
   std::ofstream logfile{"logfile.txt"};
   log(logfile);
   CHECK( std::filesystem::exists("logfile.txt"));
}
```

Hermetic isolation

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <filesystem>
#include <thread>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os)
{
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}
TEST_GROUP(LogTestGroup) { };
TEST(LogTestGroup, LogContent)
{
   std::ostringstream os;
   std::thread::id id = std::this_thread::get_id();
   os << id;

   std::string fname{"logfile"};
   fname += os.str();
   fname += ".txt";

   std::filesystem::remove(fname);
   std::ofstream logfile{fname};
   log(logfile);
   CHECK( std::filesystem::exists(fname) );
}
```

Hermetic isolation

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <filesystem>
#include <thread>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os)
{
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}
TEST_GROUP(LogTestGroup) {
  void setup()
  {
     std::ostringstream os;
     std::thread::id id = std::this_thread::get_id();
     os << id;
     fname += "logfile";
     fname += os.str();
     fname += ".txt";
     std::filesystem::remove(fname);
  }
  void teardown()
  {
     std::filesystem::remove(fname);
  }
  std::string fname;
};
TEST(LogTestGroup, LogContent)
{
   std::ofstream logfile{fname};
   log(logfile);
   CHECK( std::filesystem::exists(fname) );
}
```

Fixtures

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

TEST_GROUP(LogTestGroup)
{
  void setup()
  {
    std::cerr << "setup LogTestGroup\n";
  }
  void teardown()
  {
    std::cerr << "teardown LogTestGroup\n";
  }
};

TEST(LogTestGroup, First)
{
  std::cerr << "First\n";
}

TEST(LogTestGroup, Second)
{
  std::cerr << "Second\n";
}
```

Fixtures

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

TEST_GROUP(LogTestGroup)
{
  void setup()
  {
    std::cerr << "setup LogTestGroup\n";
  }
  void teardown()
  {
    std::cerr << "teardown LogTestGroup\n";
  }
};

TEST(LogTestGroup, First)
{
  std::cerr << "First\n";
}

TEST(LogTestGroup, Second)
{
  std::cerr << "Second\n";
}
```

```
$ ./a.out -c
setup LogTestGroup
Second
teardown LogTestGroup
.setup LogTestGroup
First
teardown LogTestGroup
.
OK (2 tests, 2 ran, 0 checks, 0 ignored,
0 filtered out, 0 ms)
```

Fixtures

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

struct MyStruct
{
  MyStruct()  { std::cerr << "MyStruct::MyStruct()"  << '\n'; }
  ~MyStruct() { std::cerr << "MyStruct::~MyStruct()" << '\n'; }
};
TEST_GROUP(LogTestGroup)
{
  void setup()
  {
    std::cerr << "setup LogTestGroup\n";
  }
  void teardown()
  {
    std::cerr << "teardown LogTestGroup\n";
  }
  MyStruct m;
};
TEST(LogTestGroup, First)
{
  std::cerr << "First\n";
}
TEST(LogTestGroup, Second)
{
  std::cerr << "Second\n";
}
```

Fixtures

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

struct MyStruct
{
  MyStruct()  { std::cerr << "MyStruct::MyStruct()"  << '\n'; }
  ~MyStruct() { std::cerr << "MyStruct::~MyStruct()" << '\n'; }
};
TEST_GROUP(LogTestGroup)
{
  void setup()
  {
    std::cerr << "setup LogTestGroup\n";
  }
  void teardown()
  {
    std::cerr << "teardown LogTestGroup\n";
  }
  MyStruct m;
};
TEST(LogTestGroup, First)
{
  std::cerr << "First\n";
}
TEST(LogTestGroup, Second)
{
  std::cerr << "Second\n";
}
```

```
$ ./a.out -c
MyStruct::MyStruct()
setup LogTestGroup
Second
teardown LogTestGroup
MyStruct::~MyStruct()
.MyStruct::MyStruct()
setup LogTestGroup
First
teardown LogTestGroup
MyStruct::~MyStruct()
..
OK (2 tests, 2 ran, 0 checks,
0 ignored, 0 filtered out, 0 ms)
```

Memory check

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <filesystem>
#include <thread>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os) {
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}

TEST_GROUP(LogTestGroup) {
  void setup()
  {
     std::ostringstream os;
     std::thread::id id = std::this_thread::get_id();
     os << id;
     fname += "logfile";
     fname += os.str();
     fname += ".txt";
     std::filesystem::remove(fname);
  }
  void teardown()
  {
     std::filesystem::remove(fname);
  }
  std::string fname;
};
TEST(LogTestGroup, LogContent)
{
   std::ofstream logfile{fname};
   log(logfile);
   CHECK( std::filesystem::exists(fname) );
}
```

Memory check

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <filesystem>
#include <thread>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os) {
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}
std::string fname;
TEST_GROUP(LogTestGroup) {
  void setup()
  {
     std::ostringstream os;
     std::thread::id id = std::this_thread::get_id();
     os << id;
     fname += "logfile";
     fname += os.str();
     fname += ".txt";
     std::filesystem::remove(fname);
  }
  void teardown()
  {
     std::filesystem::remove(fname);
  }
  std::string fname;
};
TEST(LogTestGroup, LogContent)
{
   std::ofstream logfile{fname};
   log(logfile);
   CHECK( std::filesystem::exists(fname) );
}
```

Memory check

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <filesystem>
#include <thread>
#include "CppUTest/TestHarness.h"

void log(std::ostream& os) {
  os << "Log from " << __FILE__ << ", " << __LINE__ << '\n';
}
std::string fname;
TEST_GROUP(LogTestGroup) {
  void setup()
  {
     std::ostringstream os;
     std::thread::id id = std::this_thread::get_id();
     os << id;
     fname += "logfile";
     fname += os.str();
     fname += ".txt";
     std::filesystem::remove(fname);
  }
```

```
4-memory.cpp:36: error: Failure in TEST(LogTestGroup, LogContent)
    Memory leak(s) found.
Alloc num (4) Leak size: 31 Allocated at: <unknown> and line: 0. Type: "new"
    Memory: <0x55ed3bd5d3e0> Content:
    0000: 6c 6f 67 66 69 6c 65 31  33 39 36 39 35 31 35 31 |logfile139695151|
    0010: 31 32 35 34 34 30 2e 74  78 74 00 00 00 00 00    |125440.txt.....|
Total number of leaks:  1
```

.

Exceptions

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

int factorial( int x)
{
  if ( x < 0 )
    throw std::invalid_argument{"less than zero"};
  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, BadArgument)
{
   CHECK_THROWS( std::invalid_argument, factorial(-1) );
   CHECK_THROWS( std::exception,  factorial(-1) );
}
```

Exceptions

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

int factorial( int x)
{
  if ( x < 0 )
    throw std::invalid_argument{"less than zero"};
  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, BadArgument)
{
   CHECK_THROWS( std::out_of_range,  factorial(-1) );
}
```

Exceptions

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

int factorial( int x)
{
  if ( x < 0 )
    throw std::invalid_argument{"less than zero"};
  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, BadArgument)
{
   CHECK_THROWS( std::out_of_range,  factorial(-1) );
}
2-exception.cpp:22: error: Failure in TEST(FactorialTestGroup,
BadArgument)
    expected to throw std::out_of_range
but threw a different type
```

Exceptions

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

int factorial( int x)
{
  if ( x < 0 )
    throw std::invalid_argument{"less than zero"};
  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, BadArgument)
{
   CHECK_THROWS( std::invalid_argument,  factorial(1) );
}
```

```
-exception.cpp:20: error: Failure in TEST(FactorialTestGroup,
BadArgument)
    expected to throw std::invalid_argument
but threw nothing
```

Mocking

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

int factorial( int x)
{
  mock().actualCall("factorial");
  return x <= 1 ?  1 : x*factorial(x-1);
}
TEST_GROUP(Mock)
{
  void teardown()
  {
    mock().clear();
  }
};
TEST(Mock, CallOnce)
{
   mock().expectOneCall("factorial");
   CHECK_EQUAL(1, factorial(1));
   mock().checkExpectations();
}
$ g++ 1-call.cpp test.cpp -lCppUTest -lCppUTestExt
$ ./a.out
.
OK (1 tests, 1 ran, 2 checks, 0 ignored, 0 filtered out, 0 ms)
```

Mocking

```
TEST_GROUP(Mock)
{
  void teardown()
  {
    mock().clear();
  }
};
TEST(Mock, CallOnce)
{
   mock().expectOneCall("factorial");
   CHECK_EQUAL(6, factorial(3));
   mock().checkExpectations();
}
$ ./a.out

2-call.cpp:16: error: Failure in TEST(Mock, CallOnce)
    Mock Failure: Unexpected additional (2nd) call to function:
factorial
    EXPECTED calls that WERE NOT fulfilled:
        <none>
    EXPECTED calls that WERE fulfilled:
        factorial -> no parameters (expected 1 call, called 1 time)

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

Mocking

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

struct MyClass
{
  int func(int x) {
     mock().actualCall("func");
     return 42;
  }
};
TEST_GROUP(Mock)
{
  void teardown()
  {
    mock().clear();
  }
};
TEST(Mock, CallOnObject)
{
   mock().expectOneCall("func");
   MyClass m{};
   CHECK_EQUAL(42, m.func(1));
   mock().checkExpectations();
}
```

Mocking

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

struct MyClass
{
  int func(int x) {
     mock().actualCall("func").onObject(this);
     return 42;
  }
};
TEST_GROUP(Mock)
{
  void teardown()
  {
    mock().clear();
  }
};
TEST(Mock, CallOnObject)
{
   MyClass m{};
   mock().expectOneCall("func").onObject(&m);
   CHECK_EQUAL(42, m.func(1));
   mock().checkExpectations();
}
```

Mocking

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

struct MyClass
{
  int func(int x) {
     mock().actualCall("func").onObject(this).withParameter("x",x);
     return 42;
  }
};
TEST_GROUP(Mock)
{
  void teardown()
  {
    mock().clear();
  }
};
TEST(Mock, CallOnObject)
{
   MyClass m{};
   mock().expectOneCall("func").onObject(&m).withParameter("x",1);
   CHECK_EQUAL(42, m.func(1));
   mock().checkExpectations();
}
```

CppUTest vs GoogleTest

- All CppUTest assertions are fatal, vs. EXPECT_ macros
- No user defined predicates
- No death tests
- No assertions for collections
- No type asserts
- No extra info with << operations
- Much richer mocking support in GMock