`googletest` ref sheet (2013-03-15)

Links
http://code.google.com/p/googletest/
http://code.google.com/p/googletest/wiki/Documentation

Basic test
```cpp
// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
  EXPECT_EQ(1, Factorial(0));
}

// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(1, Factorial(1));
  EXPECT_EQ(2, Factorial(2));
  EXPECT_EQ(6, Factorial(3));
  EXPECT_EQ(40320, Factorial(8));
}
```

Test names
- **NO UNDERSCORES**[1]
- prefix name with `DISABLED_` to disable it

Runner
```cpp
int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

Exec flags
(no flags) : runs all tests.
`--help`
`--gtest_list_tests` list available tests
`--gtest_filter=*` Also runs all tests. (match all)
`--gtest_filter=FooTest.*` Runs everything in test case FooTest
`--gtest_filter=*Null*:*Constructor*` Runs any test whose full name contains either "Null" or "Constructor"
`--gtest_filter=-*DeathTest.*` Runs all non-death tests
`--gtest_filter=FooTest.*-FooTest.Bar` Runs everything in test case FooTest except FooTest.Bar
`--gtest_also_run_disabled_tests` also run `DISABLED_` tests
`--gtest_repeat=1000` repeat test 1000 times (useful for 'random' errors)
`--gtest_repeat=-1` repeat forever
`--gtest_break_on_failure` stop on first failure with a breakpoint (useful in combination with `--gtest_repeat`)
`--gtest_shuffle` to check if tests are really independant (used pseudo-random seed will be displayed)
`--gtest_random_seed=SEED` to repeat a failed shuffled test
`--gtest_catch_exceptions=0` disable unexpected exceptions catching, very useful in debug

Good debug flags : (when trying to find and correct errors)
`--gtest_catch_exceptions=0` `--gtest_break_on_failure`
`--gtest_also_run_disabled_tests` `--gtest_shuffle`

Assertions : *Fatal* assertions will stop the current unit test while *nonfatal* allow it to continue and catch multiple failures at once
⇨ Use fatal only when subsequent expectations depends on this one

Nonfatal (use them first)	**Fatal** (only when needed)	**Verifies :**
Basic assertions		
`EXPECT_TRUE(condition);`	`ASSERT_TRUE`	condition is true
`EXPECT_FALSE(condition);`	`ASSERT_FALSE`	condition is false
Binary comparison		
`EXPECT_EQ(expected, actual);`	`ASSERT_EQ`	expected == actual ✋ *please note the correct order of args*
`EXPECT_NE(val1, val2);`	`ASSERT_NE`	val1 ≠ val2
`EXPECT_LT(val1, val2);`	`ASSERT_LT`	val1 < val2
`EXPECT_LE(val1, val2);`	`ASSERT_LE`	val1 ≤ val2
`EXPECT_GT(val1, val2);`	`ASSERT_GT`	val1 > val2
`EXPECT_GE(val1, val2);`	`ASSERT_GE`	val1 ≥ val2
C string comparison		
`EXPECT_STREQ(expected_str, actual_str);`	`ASSERT_STREQ`	the two C strings have the same content
`EXPECT_STRNE(str1, str2);`	`ASSERT_STRNE`	the two C strings have different content
`EXPECT_STRCASEEQ(expected_str, actual_str);`	`ASSERT_STRCASEEQ`	the two C strings have the same content, ignoring case
`EXPECT_STRCASENE(str1, str2);`	`ASSERT_STRCASENE`	the two C strings have different content, ignoring case
Exception assertions		
`EXPECT_THROW(statement, exception_type);`	`ASSERT_THROW`	statement throws an exception of the given type
`EXPECT_ANY_THROW(statement);`	`ASSERT_ANY_THROW`	statement throws an exception of any type
`EXPECT_NO_THROW(statement);`	`ASSERT_NO_THROW`	statement doesn't throw any exception
Predicate assertions (for better error messages)		
`EXPECT_PRED1(pred1, val1);`	`ASSERT_PRED1`	`pred1(val1)` returns true
`EXPECT_PRED2(pred2, val1, val2);`	`ASSERT_PRED2`	`pred2(val1, val2)` returns true
`EXPECT_PRED_FORMAT1(pred_format1, val1);`	`ASSERT_PRED_FORMAT1`	`pred_format1(val1)` is successful
`EXPECT_PRED_FORMAT2(pred_format2, val1, val2);`	`ASSERT_PRED_FORMAT2`	`pred_format2(val1, val2)` is successful
Floating-point comparison		
`EXPECT_FLOAT_EQ(expected, actual);`	`ASSERT_FLOAT_EQ`	the two float values are almost equal
`EXPECT_DOUBLE_EQ(expected, actual);`	`ASSERT_DOUBLE_EQ`	the two double values are almost equal
`EXPECT_NEAR(val1, val2, abs_error);`	`ASSERT_NEAR`	the difference between values doesn't exceed the given absolute error
Windows HRESULT assertions		
`EXPECT_HRESULT_SUCCEEDED(expression);`	`ASSERT_HRESULT_SUCCEEDED`	expression is a success HRESULT
`EXPECT_HRESULT_FAILED(expression);`	`ASSERT_HRESULT_FAILED`	expression is a failure HRESULT
Type assertions		
`::testing::StaticAssertTypeEq<T1, T2>();`		
Death tests		
``EXPECT_DEATH(statement, regex`);``	`ASSERT_DEATH`	statement crashes with the given error
``EXPECT_DEATH_IF_SUPPORTED(statement, regex`);``	`ASSERT_DEATH_IF_SUPPORTED`	if death tests are supported, verifies that statement crashes with the given error; otherwise verifies nothing
``EXPECT_EXIT(statement, predicate, regex`);``	`ASSERT_EXIT`	statement exits with the given error and its exit code matches predicate

Custom failure messages :
```cpp
ASSERT_EQ(x.size(), y.size()) << "Vectors x and y are of unequal length";

for (int i = 0; i < x.size(); ++i) {
  EXPECT_EQ(x[i], y[i]) << "Vectors x and y differ at index " << i;
}
```

TODO : fixtures
TODO : Predicate Assertions for Better Error Messages
TODO : Teaching Google Test How to Print Your Values
TODO : Value-Parameterized Tests

TODO : Type-Parameterized Tests

Google Test http://code.google.com/p/googletest/issues/detail?id=38

Printing instructions :

1. Don't print this page and subsequent pages
2. Print in **color** if possible (color is information, color increases information density)
3. Ask your print manager to **NOT RESIZE** or FIT the content (especially adobe reader which tend to do that)
4. print in **recto/verso** "**across short size**"

After printing :

1. Fold along the lines
2. Share !
3. Send me a token of appreciation for my hard work (facebook like, paypal, etc.)
4. Contribute

1 http://code.google.com/p/googletest/wiki/FAQ#Why_should_not_test_case_names_and_test_names_contain_underscore