CPPOCL Unit Test Framework

This unit test framework is designed to be used with C++ and only requires Test.hpp to be included, and TestClass.cpp to be added to your makefile or project file.

To make it easier and faster to write the unit tests, macros have been used for defining the tests for functions, member functions or general purpose.

Tests can be written to verify correct functionality or performance.

Unit tests have been written to test this framework.

Overview of using the macros

The test macros allow unit testing of a function, member function or a general test to be written.

It's also possible to override the names used in the macros so that the output accurately describes the function name or arguments.

Within the test macros, checker macros can be used to test various conditions for your unit tests.

These macros can be found in TestMacros.hpp.

Overview of helper functions

The framework also provides so helper functions for doing some basic but common manipulation of basic data types, i.e. char*.

If for any reason you don't want these helper functions to be enabled, then define
OCL_TEST_HELPERS_DISABLED pre-processor in your build.

These helper functions also provide support for char* and wchar_t* string pointers.

These helpers functions can be found in TestClassHelperFunctions.inl.

Macros for testing functionality or performance

TEST

TEST_FUNCTION

TEST_FUNCTION_TIME

TEST_MEMBER_FUNCTION

TEST_MEMBER_FUNCTION_TIME

TEST_CONST_MEMBER_FUNCTION

TEST_CONST_MEMBER_FUNCTION_TIME

Macros for checking conditions

CHECK_TRUE

CHECK_FALSE

CHECK_EQUAL

CHECK_NOT_EQUAL

CHECK_GREATER

CHECK_GREATER_EQUAL

CHECK_LESS

CHECK_LESS_EQUAL

CHECK_NULL

CHECK_NOT_NULL

CHECK_ZERO

CHECK_NOT_ZERO

CHECK_COMPARE

CHECK_NOT_COMPARE

CHECK_TIME

CHECK_EXCEPTION

CHECK_ALL_EXCEPTIONS

Macros for customizing test

TEST_FAILURE_INDENT	/* Set number of spaces to indent error information */
TEST_OVERRIDE_FUNCTION_NAME	/* Set the function name for output, e.g. “operator=” */
TEST_OVERRIDE_ARGS	/* Set the argument names for output, e.g. “int, int” */
TEST_OVERRIDE_FUNCTION_NAME_ARGS	/* Set the function name and arguments for output */

Helper functions

StrLen	/* Return the length of a char* or wchar_t* variable as a size_t */
StrEnd	/* Return a pointer to the end of the string, i.e. position of '\0' */
SetStr	/* Set a const char* pointer and length from a source string */
StrCpy	/* Same as strcpy (without Microsoft warnings) */
StrCmp	/* Same as strcmp (without Microsoft warnings) */
CharCount	/* Count all characters matching character(s) to find in string */
MemCmp	/* Same as memcmp */
IsDigit	/* Return true when character is in '0'..'9' */
ToInt	/* Convert character or string to integer type */

Examples for unit tests

```
// Only need to call this once for all unit tests, outside of any test.
TEST_FAILURE_INDENT(4);

// Perform some checks on some functions.
TEST(TestSomeFunctions)
{
   std::string str;
   CHECK_TRUE(str.empty());
   CHECK_ZERO(str.length());
}

// Test function std::min(int, int)
TEST_FUNCTION(min, ints)
{
   TEST_OVERRIDE_FUNCTION_NAME_ARGS(“std::min”, “int, int”);
   CHECK_EQUAL(min(1, 2), 1);
}

// Test member function std::string::insert(size_t, std::string)
// NOTE: NA is pre-defined to signify that function doesn't have arguments.
TEST_MEMBER_FUNCTION(string, insert, size_t_insert, NA)
{
   TEST_OVERRIDE_ARGS(“size_t, string const&”);
   std::string str("ello");
   CHECK_EQUAL(::strcmp(str.insert(0, "h").c_str(), "hello"), 0);
}

TEST_CONST_MEMBER_FUNCTION(string, length, NA)
{
   std::string str;
   CHECK_ZERO(str.length ());
}
```

Examples for performance tests

```
// Test min for a sample time period of 1.5 seconds.
TEST_FUNCTION_TIME(min, ints, 1, 500)
{
   TEST_OVERRIDE_FUNCTION_NAME_ARGS(“std::min”, “int, int”);
   CHECK_TIME(min(1, 2));
}

// Test string::length for a sample time period of 1 second.
TEST_CONST_MEMBER_FUNCTION_TIME(string, length, 1, 0)
{
   std::string str = "Hello World!";
   CHECK_TIME(str.length());
}
```

Examples of helper functions

```
char const* str = “Hello”;
CHECK_EQUAL(StrLen(str), 5U);

char const* end = StrEnd(str);
CHECK_EQUAL(str, end + 5);

char* str_cpy = new char[StrLen(str)+1];
CHECK_NOT_NULL(str_cpy);
StrCpy(str_cpy, str);

CHECK_EQUAL(StrCmp(str_cpy, str), 0);

CHECK_EQUAL(CharCount(str, 'e'), 1U);

CHECK_ZERO(MemCmp(str_cpy, str));

CHECK_TRUE(IsDigit('1'));

StrCpy(str_cpy, “12”);
int value = 0;
CHECK_TRUE(ToInt(str_cpy, value));
CHECK_EQUAL(value, 12);
```