Software design (C++) course work: String class & test runner

Code has been developed using mingw 4.8.1 on Windows 8.1. Tested to be working on Visual Studio 2013. Linux or OS X compatibility has not been tested. NDEBUG is defined; remove #define NDEBUG from string.cpp to enable assertions.

Notable design decisions

Test suite

Test methods are registered into test runner by calling register_test method and providing the name of the test and the test function. Test function must not have any parameters and must have void return value. Helper macro REGISTER_TEST(runner, function) exists that registers the test using function name as test name.

Test suite provides various macros for testing (defined in test.h): ASSERT_EQUALS, ASSERT_TRUE, ASSERT_FALSE and ASSERT_THROWS. Mostly these are just helper macros that wrap over actual test functions that expect line number and file name as well (e.g. ASSERT_EQUALS(expected, actual) wraps assert_equality(expected, actual, line, file)).

ASSERT_THROWS(operation, expected_exception) is slightly more interesting, as the operation must be called inside try/catch-block so that the exception can actually be caught and verified. This means that the ASSERT_THROWS cannot wrap a function, and instead must do a large text insertion that introduces try/catch expected/catch unexpected-block to call site. The replacement itself is wrapped inside a block to prevent any helper variable declarations from causing issues in the test method.

Memory allocation tracking

I decided to replace the standard new\delete functions with my own versions, as defining those in any translation unit overrides the standard new\delete-functions. My custom versions use malloc\free internally, and use a global singleton class to store the (de)allocation information.

The storage class tracks allocations by saving and removing addresses in std::map (address\allocation size pairs). I chose map over unordered_map to guarantee the order of addresses when printing errors. The class had some issues with instrumented new\delete functions (see interesting issues section below), which forced me to provide a custom malloc based allocator for the maps.

Test runner uses the singleton to determine if there are either memory leaks or double deletes after test method has been run.

Interesting issues during development

Uninitialized variable causing random crashes

Fairly bog-standard whoops-I-forgot-to-initialize-a-variable –issue, but it was somewhat annoying as it still managed to work most of the time.

Copy constructor did not initialize the class variables at all as those would be overwritten immediately. However, the copy constructor calls method copy_string that uses those variables and assumes that they are initialized. The copy operation still succeeded most of the times, but sometimes the invalid values would lead into crash when copy operation assumed that the class already had enough buffer space for copy.

Memory allocation instrumentation causing stack overflow

Memory allocation tracking ultimately uses std::map to store the allocated pointers. This led to rather annoying bug, since map is implemented as a tree, and it internally uses new when creating new nodes. This meant that any calls to new in the program would lead into call to new -> save pointer into map -> allocate space for map node -> call to new -> save pointer into map -> … loop that would eventually cause a stack overflow. Thankfully C++ containers allow you to provide a custom allocator for them, so I was able to solve this by providing a barebones allocator that uses malloc\free instead of new\delete.

Known issues

No known bugs or issues in production version (NDEBUG defined). When NDEBUG is not defined, exception guarantees or class invariants are not respected if assert fails as assert throws. It is assumed that this does not matter as NDEBUG would only be turned off in testing environment where memory leaks do not matter that much (and even then, memory leaks would only happen due to severe logic error in string class).