Lecture Notes for Lecture 13 of CS 5001 (Foundations of CS) for the Fall, 2018 session at the Northeastern University Silicon Valley Campus.

Unit Testing

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 12 Review

- Separating code into multiple source files is a must for many reasons, including code size, partitioning functionality by data type, team collaboration, and testing.
- The key concept is separating the declarations of functions and variables from their definitions.
- The C preprocessor allows putting function and variable declarations into header (“.h”) files that can be included in any file where the declaration is needed.
- Function and variable definitions go into source (”.c”) files that #include the declarations in the header files
- Examples in class showed how to modularize Issn, Publisher, and Journal data types into their own “.c” and “.h” files, and call them from main program and testing modules.

Creating Unit Tests

- The previous lecture stated that one advantage of separating code into separate modules by functionality is that it simplifies creating separate unit test modules for them.
- We could continue to simply call functions with sample values on our own, but using a unit test framework has a number of advantages, including automating unit tests for a number of modules, and reporting results.
- In this lecture, we will look at how to create unit tests modules for the Issn, Publisher, and Journal data types from previous lectures, and how to use the Cunit unit test framework for C and C++ to define and run them.

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Ensure that the CUnit library and includes are configured for each project as described under “Eclipse IDE” in module 0.
- For this first exercise, put all the unit tests in the main module, “main.c”. We will learn more advanced ways to manage tests later.
- Add the following #include declarations to “main.c”

```
#include "CUnit/CUnit.h"
#include "CUnit/Basic.h”
#include “journal.h”
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Create static functions that test Issn, Publisher, and Journal
  We use static because tests are used only within “main.c”.

```
/**
 * Test Issn functions.
 */
static void test_issn(void) { … }

/**
 * Test Publisher functions.
 */
static void test_publisher(void) { … }

/**
 * Test journal functions.
 */
static void test_journal(void) { … }
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Create static function, test_all_local() to run all tests in one suite.

```
/**
  * Run all tests.
  * @return test error code
  */
static CU_ErrorCode test_all_local(void) {
   // initialize the CUnit test registry – only once per application
   CU_initialize_registry();

   // add a single suite to the registry with no init or cleanup
   CU_pSuite pSuite = CU_add_suite(”test all", NULL, NULL);


   // add the tests to the suite
   CU_add_test( pSuite, ”Issn tests", test_issn);   // test Issn functions
   CU_add_test( pSuite, ”publisher tests", test_publisher);  // test publisher functions
   CU_add_test( pSuite, ”journal tests", test_journal);  // test journal functions
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Create static function, test_all_local() to run all tests in one suite.

```
    // run all suites using the basic interface that echoes to the console in this example
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();

    // display information on failures that occurred
    CU_basic_show_failures(CU_get_failure_list());

    // Clean up registry and return status CU_cleanup_registry();
    return CU_get_error();
}
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Use main() method to run unit tests.

```
/**
 * Main program to invoke test functions
 * @return the exit status of the unit tests
 */
int main(void) {
	// test all the functions
	CU_ErrorCode code = test_all_local();
	return (code == CUE_SUCCESS) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Test output.

```
     CUnit - A unit testing framework for C - Version 2.1-3
    http://cunit.sourceforge.net/

Suite: test all
  Test: issn tests ...passed
  Test: publisher tests ...passed
  Test: journal tests ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      3      3      3      0        0
             asserts     10     10     10      0      n/a


Elapsed time =    0.000 seconds
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Framework supports multiple suites of tests, with multiple test functions per suite.
- Each test consists of calls to unit test functions that compare actual to expected results.
- Test assertion functions for data of various types, including basic types, strings, and pointers.
- No tests for aggregate types such as arrays and structs; these must be built up from existing assertions.

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

CU_ASSERT(int expression) **CU_ASSERT_FATAL**(int expression) **CU_TEST**(int expression) **CU_TEST_FATAL**(int expression)	Assert that *expression* is TRUE (non-zero)
CU_ASSERT_TRUE(value) **CU_ASSERT_TRUE_FATAL**(value)	Assert that *value* is TRUE (non-zero)
CU_ASSERT_FALSE(value) **CU_ASSERT_FALSE_FATAL**(value)	Assert that *value* is FALSE (zero)
CU_PASS(message)	Register a passing assertion with the specified message. No logical test is performed.
CU_FAIL(message) **CU_FAIL_FATAL**(message)	

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

CU_ASSERT_EQUAL(actual, expected) **CU_ASSERT_EQUAL_FATAL**(actual, expected)	Assert that *actual* = = *expected*
CU_ASSERT_NOT_EQUAL(actual, expected)) **CU_ASSERT_NOT_EQUAL_FATAL**(actual , expected)	Assert that *actual* != *expected*

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

CU_ASSERT_PTR_EQUAL(actual, expected) **CU_ASSERT_PTR_EQUAL_FATAL**(actual, expected)	Assert that pointers *actual* = = *expected*
CU_ASSERT_PTR_NOT_EQUAL(actual, expected) **CU_ASSERT_PTR_NOT_EQUAL_FATAL**(actual, expected)	Assert that pointers *actual* != *expected*
CU_ASSERT_PTR_NULL(value) **CU_ASSERT_PTR_NULL_FATAL**(value)	Assert that pointer *value* == NULL
CU_ASSERT_PTR_NOT_NULL(value) **CU_ASSERT_PTR_NOT_NULL_FATAL**(value)	Assert that pointer *value* != NULL

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

CU_ASSERT_STRING_EQUAL(actual, expected) **CU_ASSERT_STRING_EQUAL_FATAL**(actual, expected)	Assert that strings *actual* and *expected* are equivalent
CU_ASSERT_STRING_NOT_EQUAL(actual, expected) **CU_ASSERT_STRING_NOT_EQUAL_FATAL**(actual, expected)	Assert that strings *actual* and *expected* differ
CU_ASSERT_NSTRING_EQUAL(actual, expected, count) **CU_ASSERT_NSTRING_EQUAL_FATAL**(actual, expected, count)	Assert that 1st count chars of *actual* and *expected* are the same
CU_ASSERT_NSTRING_NOT_EQUAL(actual, expected, count) **CU_ASSERT_NSTRING_NOT_EQUAL_FATAL**(actual, expected, count)	Assert that 1st count chars of *actual* and *expected* differ

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

CU_ASSERT_DOUBLE_EQUAL(actual, expected, granularity) **CU_ASSERT_DOUBLE_EQUAL_FATAL**(actual, expected, granularity)	Assert that |*actual* - *expected*| <= |*granularity*| *Math library must be linked in for this assertion.*
CU_ASSERT_DOUBLE_NOT_EQUAL(actual, expected, granularity) **CU_ASSERT_DOUBLE_NOT_EQUAL_FATAL**(actual, expected, granularity)	Assert that |*actual* - *expected*| > |*granularity*| *Math library must be linked in for this assertion.*

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Example: test_issn() unit test function

```
/**
 * Test Issn functions.
 */
static void test_issn(void) {
        char str[20];  // char array converting Issn to string
        char *s;   // conversion result pointer

        // convert ISSN to string
        Issn issn0 = 0x12345678;
        s = issnToString(issn0, str);
        CU_ASSERT_STRING_EQUAL(s, "1234-5678");

        // parse string to Issn
        Issn issn_0 = parseIssn(s);
        CU_ASSERT_EQUAL(issn0, issn_0);
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Example: test_issn() unit test function

```
        // convert  ISSN with X checksum
        Issn issn1 = 0x0034567a;
        s = issnToString(issn1, str);
        CU_ASSERT_STRING_EQUAL(s, "0034-567X");

        // parse string to Issn
        Issn issn_1 = parseIssn(s);
        CU_ASSERT_EQUAL(issn1, issn_1);

        // parse invalid ISSN
        Issn issn_2 = parseIssn("012a-5468");
        CU_ASSERT_EQUAL(issn_2, ISSN_UNKNOWN);
        // parse invalid ISSN

        Issn issn_3 = parseIssn("");
        CU_ASSERT_EQUAL(issn_3, ISSN_UNKNOWN);
}
```

Creating Unit Tests

Using Separate Suites for Each Module

- Another way to organize tests is to have separate test files for each module.
- Each test file has a public function that sets up a named suite of tests for that module, with multiple static test functions for the suite.
- A separate file “test_all.c” contains the test driver function test_all(). It calls the public test function for each test file, which registers a suite and its test functions.
- The main() function calls test_all(), which runs the unit test framework function on all the suites.
- This also allows running individual suites.

Creating Unit Tests

Using Separate Suites for Each Module

- Example: issn_test.c unit test file

```
#include <stdlib.h>
#include ”issn.h"
#include "CUnit/CUnit.h"
#include "CUnit/Basic.h”

/**
 * Test Issn functions.
 */
static void test_issn(void) { … }
```

Creating Unit Tests

Using Separate Suites for Each Module

- Example: issn_test.c unit test file

```
/**
 * Add suite for Issn unit tests
 */
void issn_test(void) {
        // add a suite to the registry with no init or cleanup
        CU_pSuite pSuite = CU_add_suite("issn_test", NULL, NULL);

        // add the tests to the suite
        CU_add_test(pSuite, "issn tests", test_issn);
}
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Modified test_all() calls functions to register test suites.

```
extern void issn_test(void);
extern void jouranl_test(void);
extern void publisher_test(void);

/**
 * Run all tests.
 * @return test error code
 */
static CU_ErrorCode test_all(void) {
   // initialize the CUnit test registry – only once per application
   CU_initialize_registry();

   // call functions to register test suites
   issn_test();            // register issn test suite
   pubisher_test();    // register publisher test suite
   journal_test();       // register journal test suite
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Modified test_all() calls functions to register test suites.

```
    // run all suites using the basic interface that echoes to the console in this example
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();

    // display information on failures that occurred
    CU_basic_show_failures(CU_get_failure_list());

    // Clean up registry and return status CU_cleanup_registry();
    return CU_get_error();
}
```

Creating Unit Tests

Using the CUnit Test Framework in Eclipse

- Revised main() method run unit tests in main.c.

```
extern CU_ErrorCode test_all(void);
/**
 * Main program to invoke test functions
 * @return the exit status of the unit tests
 */
int main(void) {
        // test all the functions
        CU_ErrorCode code = test_all ();
        return (code == CUE_SUCCESS) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```