LAB: Unit testing with mocks

This article is based on unit-testing article and code labs from:

- Sibros public unit-test wiki

For a conceptual overview, see Unit-Test Basics and Mocks.

Part 1

Let us practice unit-testing, with a little bit of TDD thrown into the mix.

`steering.h` : This is just a header file and we will Mock out this file and therefore you do not need to write this file's implementation.

```
#pragma once
void steer_left(void);void steer_right(void);
```

`steer_processor.h` : You will write the implementation of this file yourself at `steer_processor.c`

```
#pragma once
#include <stdint.h>
#include "steering.h"
/**
 * Assume that a threshold value is 50cm
 * Objective is to invoke steer function if a sensor value is less than the threshold
 *
 * Example: If left sensor is 49cm, and right is 70cm, then we should call steer_right()
 */void steer_processor(uint32_t left_sensor_cm, uint32_t right_sensor_cm);
```

`test_steer_processor.c` You will write the test code, **before you write the implementation of `steer_processor()` function**.

```

```

```
#include "unity.h"
#include "steer_processor.h"
#include "Mocksteering.h"
void test_steer_processor__move_left(void) { }
void test_steer_processor__move_right(void) { }
void test_steer_processor__both_sensors_less_than_threshold(void) { }
// Hint: If you do not setup an Expect()
// then this test will only pass none of the steer functions is called
void test_steer_processor__both_sensors_more_than_threshold(void) {
}
// Do not modify this test case
// Modify your implementation of steer_processor() to make it pass
// This tests corner case of both sensors below the threshold
void test_steer_processor(void) {
  steer_right_Expect();
  steer_processor(10, 20);
  steer_left_Expect();
  steer_processor(20, 10);
}
```

Do the following:

- Put the `steering.h` in your source code
- Put the `steer_processor.h` in your source code
- Put the `test_steer_processor.c` in your test code folder
- Write the implementation of `test_steer_processor.c` and run the tests to confirm failing tests
- Write the implementation of `steer_processor.c`

Part 2

In this part, the objectives are:

- Practice `StubWithCallback` or `ReturnThruPtr`
- Ignore particular arguments

`message.h`: This is just an interface, and we will Mock this out meaning that we will not write the code for `message_read()` API:

```
```

```
#pragma once
#include <stdbool.h>
typedef struct {
  char data[8];
} message_s;
bool message__read(message_s *message_to_read);
```

`message_processor.c`: This code module processes messages arriving from `message__read()` function call. There is a lot of nested logic that is testing if the third message contains `$` or `#` at the first byte. To get to this level of the code, it is difficult because you would have to setup your test code to return two dummy messages, and a third message with particular bytes.

To improve test-ability, you should refactor the `} else {` logic into a separate `static` function that you can hit with your unit-tests directly. Please ask your instructor to demonstrate how to refactor code for improved ability to test.

```
#include <stdbool.h>
#include <stddef.h>
#include <string.h>
#include "message_processor.h"
/**
 * This processes messages by calling message__read() until:
 *   - There are no messages to process -- which happens when message__read() returns false
 *   - At most 3 messages have been read
 */
bool message_processor(void) {
  bool symbol_found = false;
  message_s message;
  memset(&message, 0, sizeof(message));
  const static size_t max_messages_to_process = 3;
  for (size_t message_count = 0; message_count < max_messages_to_process; message_count++) {
    if (!message__read(&message)) {
      break;
    } else {
      if (message.data[0] == '$') {
        symbol_found = true;
      } else {
```

```
        // Symbol not found
      }
    }
  }
  return symbol_found;}
```

`test_message_processor.c`: Add more unit-tests to this file as needed.

```
#include "unity.h"
#include "Mockmessage.h"
#include "message_processor.h"
// This only tests if we process at most 3 messages
void test_process_3_messages(void) {
  message__read_ExpectAndReturn(NULL, true);
  message__read_IgnoreArg_message_to_read();
  message__read_ExpectAndReturn(NULL, true);
  message__read_IgnoreArg_message_to_read();
  // Third time when message_read() is called, we will break the loop since it is meant to process 3 m
  message__read_ExpectAndReturn(NULL, true);
  message__read_IgnoreArg_message_to_read();
  // Since we did not return a message that starts with '$' this should return false
  TEST_ASSERT_FALSE(message_processor());
}
void test_process_message_with_dollar_sign(void) {
}
void test_process_messages_without_any_dollar_sign(void) {
}
// Add more tests if necessary
```

Hint (sample code):

```
#include "unity.h"
#include "Mockmessage.h"
#include "message_processor.h"
```

```
static bool message__read_stub(message_s *message_to_read, int call_count) {
  bool message_was_read = false;

  if (call_count >= 2) {
    message_was_read = false;
  } else {
    message_was_read = true;
  }

  if (call_count == 0) {
    message_to_read->data[0] = 'x';
  }
  if (call_count == 1) {
    message_to_read->data[1] = '$';
  }
  return message_was_read;
}
// This only tests if we process at most 3 messages
void test_process_messages_with_stubWithCallback(void) {
  // message_processor() makes a call to:
  // bool message__read(message_s *message_to_read);

  // Whenever message__read() occurs, it will go to your custom "stub" function
  // Once we stub, then each function call to message__read() will go to message__read_stub()
  message__read_StubWithCallback(message__read_stub);
  // Function under test
  message_procesor();
}
// Add more tests if necessary
```

Requirements

- Test thoroughly
  - Do not hack internals of a module.
  - This means that only operate using the APIs, and do not modify the data structure
- Each test should start with a known initial state, you should not rely on previous test to run before the current test

- `setUp()` method may be used to re-initialize code modules

Revision #17
Created 6 years ago by Preet Kang
Updated 2 years ago by Preet Kang