Lab: Unit testing with mocks

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

- Sibros public unit-test wiki

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

Write the unit-tests first, and then the implementation for the following header file:

```
#pragma once

#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

/* In this part, the queue memory is statically defined
 * and fixed at compile time for 100 uint8s
 */
typedef struct {
  uint8_t queue_memory[100];

  // TODO: Add more members as needed
} queue_s;
```

```
15
16 // This should initialize all members of queue_s
17 void queue__init(queue_s *queue);
18
19 /// @returns false if the queue is full
20 bool queue__push(queue_s *queue, uint8_t push_value);
21
22 /// @returns false if the queue was empty
23 bool queue__pop(queue_s *queue, uint8_t *pop_value);
24
25 size_t queue__get_item_count(const queue_s *queue);
```

Part 3

Write the unit-tests first, and then the implementation for the following header file. This is a slight variation of Part 2 and it provides you with the static memory based programming pattern popular in Embedded Systems where we deliberately avoid allocating memory on the heap.

```
1  #pragma once
2
3  #include <stdbool.h>
4  #include <stddef.h>
5  #include <stdint.h>
6
7  /* In this part, the queue memory is statically defined
8   * by the user and provided to you upon queue__init()
9   */
10 typedef struct {
11   uint8_t *static_memory_for_queue;
12   size_t static_memory_size_in_bytes;
13
14   // TODO: Add more members as needed
15 } queue_s;
16
17 /* Initialize the queue with user provided static memory
18  * @param static_memory_for_queue This memory pointer should not go out of scope
19  *
20  * @code
21  *   static uint8_t memory[128];
22  *   queue_s queue;
23  *   queue__init(&queue, memory, sizeof(memory));
24  * @endcode
25  */
26 void queue__init(queue_s *queue, void *static_memory_for_queue, size_t static_memory_size_i
27
28 /// @returns false if the queue is full
29 bool queue__push(queue_s *queue, uint8_t push_value);
```

```
30
31 /// @returns false if the queue was empty
32 /// Write the popped value to the user provided pointer pop_value_ptr
33 bool queue__pop(queue_s *queue, uint8_t *pop_value_ptr);
34
35 size_t queue__get_item_count(const queue_s *queue);
```

Part 4

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.

```
1 #pragma once
2
3 #include <stdbool.h>
4
5 typedef struct {
6   char data[8];
7 } message_s;
8
9 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.

```
1 #include <stdbool.h>
2 #include <stddef.h>
3 #include <string.h>
4
5 #include "message_processor.h"
6
7 /**
8  * This processes messages by calling message__read() until:
9  *   - There are no messages to process -- which happens when message__read() returns false
10 *   - At most 3 messages have been read
11 */
12 bool message_processor(void) {
13   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();

  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
```