Programming in C
Exercises: Unit Tests

Version: October 2024
Author: Brice Guayrin (b.guayrin@fontys.nl)

Introduction

The archive startUp_UnitTests.zip provides a Visual Studio Code project. The project is intended to be used as an initial project for installing the unity framework and practicing with designing and implementing unit tests in C.

The Visual Studio code project consists of the following modules:

1) Module "shared/statistics" consists of several functions performing statistical operations on integers (e.g. finding a maximum value)

2) Module "product/main" provides an end-user with a console application to perform the operations of module “statistics”

3) Modules "test/main" and "test/statistics_test" provides software developers with a console application to check whether the functions in module "shared/statistics" are functional. The use of *setUp()* and *tearDown() in statistics_test.c* is optional.

Exercise 1:

You are tasked to implement unit tests to verify that the function *find_maximum* (see in Table 1) operates as expected. The function is already declared in *statistics.h* and defined in *statistics.c*. Note that this function voluntarily includes a bug. Will you be able to find the bug using unit tests?

```
// pre  : the PREcondition statement. Indicates what must be true before the function is called.
// post : the POSTcondition statement indicates what will be true when the function finishes its work.

/*
 *  pre: N.A. (Not Applicable)
 *  post: function returns the maximum value of the two inputs variables
 *  input(s):
 *      - a: first integer input
 *      - b: second integer input
 *  output(s):
 *      - maximum of the two integer inputs
 */
int find_maximum(int a, int b);
```

Table 1: prototype of function find maximum

- First think about the test cases to implement in order to ensure that the function operates as expected. Use the following table (or similar) to design your test cases (see in Table 2). Note that the following table is a template (it does NOT mean that you should strictly use 3 test cases). To help you defining your test cases, think of the following:
    - What kind of test cases should be defined?
    - How many test cases to write per function?
    - When do we know that a function is fully tested?

Test ID	Description	Test inputs	Expected result

Table 2: Test cases specification

- Once you have designed your test cases, write your unit tests in file *statistics_test.c* using the unity framework. What assertions are you going to use?

- Have you eventually found the bug in the function *find_maximum* using unit tests? If so, do not forget to fix the bug in the function *find_maximum*.

Exercise 2:

You are tasked to implement unit tests to verify that the function *find_maximum_array* (see in Table 3) operates as expected. The function is already declared in *statistics.h* but NOT defined in *statistics.c*. Will you be able to implement the function *find_maximum_array* and the corresponding unit tests?

```
/*
 *  pre: N.A.
 *  post: function finds the maximum value of an array of integers
 *  input(s):
 *      - array: pointer to the first element of an array of integers
 *      - size: number of elements in the array of integers
 *      - maximum: pointer to the maximum value in the array of integers
 *  output(s):
 *      - integer indicating the successful execution of the function
 *       (i.e. 0 on success or -1 if an error occurs)
 */
int find_maximum_array(int* array, int size, int* maximum);
```

Table 3: prototype of function find maximum_array

- First define the function *find_maximum_array* in *statistics.c*. Will you use optimised pointer arithmetic to implement the function?

- You can then design your test cases using a similar template table as with exercise 1. What edge cases will you test? Also, do not forget to verify that the function does not manipulate NULL pointers. To give you inspiration, see below in Table 4 an example of a meaningful test case for testing the function *find_maximum_array*. Keep in mind that the example is not sufficient to thoroughly verify the functionality of the function. You therefore need to design additional test cases by yourself.

- You are now ready to implement your unit tests in C using the Unity Framework.

ID	Description	Inputs	Expected results
1	Test that function *find_maximum_array* actually finds the maximum value	- array: {1, 5, 3, 12, 9} - size: 5 - maximum points to value 0	- maximum points to value 12 - returned integer is 0
2	Test function *find_maximum_array* with an invalid pointer	- array: NULL - size: 5 - maximum points to value 0	- returned integer is -1

Table 4: examples of two meaningful test cases for function find_maximum_array

Exercise 3:

You are tasked to implement unit tests to verify that the function *sort_array* (see in Table 5) operates as expected. The function is already declared in *statistics.h* but NOT defined in *statistics.c*. You are challenged to implement the function *sort_array* and the corresponding unit tests.

```
/*
 *  pre: N.A.
 *  post: function sorts an array of integers in ascending order
 *  input(s):
 *      - array: pointer to the first element of an array of integers
 *      - size: number of elements in the array of integers
 *  output(s):
 *      - integer indicating the successful execution of the function
 *       (i.e. 0 on success or -1 if an error occurs)
 */
int sort_array(int* array, int size);
```

Table 5: prototype of function sort_array

- Define the function *sort_array* in *statistics.c*. Which sorting algorithm will you use? When applicable, consider using optimised pointer arithmetic to implement your algorithm.

- You can then design your test cases using a similar template table as with exercise 1. How to ensure that all possible execution path are tested?

- You are now ready to implement your unit tests in C using the Unity Framework. Remember that assertions ending with the postfix "_ARRAY" are used to check that all elements in two arrays are identical.