Programming in C
Exercises: Unit Tests

Version: April 2023
Author: Brice Guayrin (b.guayrin@fontys.nl)

Introduction

The archive startUp_UnitTests.rar 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 three modules:

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

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

3) Module "main_test" provides software developers with a console application (main_test.exe) to check whether the functions in module "statistics" are functional. The use of *setUp()* and *tearDown()* in main_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?

```
/*
 *  Function finds and 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

- Before writing your test cases, 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 *main_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 2Table 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?

```
/*
 * 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):
 *     - Boolean indicating the successful execution of the function
 *       (e.g. return false if at least one of the input pointers is NULL)
 */
bool 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 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.

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

Exercise 3:

You are tasked to implement unit tests to verify that the function *sort_array* (see in Table 4) 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.

```
/*
 *  Function sorts the 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):
 *      - Boolean indicating the successful execution of the function
 *       (e.g. return false if at least one of the input pointers is NULL)
 */
bool sort_array(int* array, int size);
```

Table 4: prototype of function sort_array

- Define the function *sort_array* in statistics.c. Which sorting algorithm will you use? Think of using 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.

Exercise 4:

In case you are already done with the 3 exercises above, design and implement test cases for the exercise of Week8 (*startUp_studentManagment_Part3.rar*). Think of using the template table (see Table 2) when designing your test cases.