Programming in C

Exercises Week3: Functions & Structures

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

Introduction

The archive *startUpExerciseWeek3.zip* provides a Visual Studio Code project. The Visual Studio code project consists of the initial version of a C console application to handle and facilitate the administration of students of schools and universities. The end-users of the console application can be e.g. administration employees and teachers. Keep in mind that the application is kept purposefully (very) simple in the context of this exercise, with the intention of practicing functions and structures. All initial source code is provided in file *main.c*. You will have to extend the code in file *main.c* and you are not requested to create any other files.

In its initial version the application is able to store the personal details of two student, namely Lisa and Eric. The personal details stored by the application consists of the name, the number (similar to PCN number of Fontys), the age and the grades of each student. The application is also initially capable of calculating the average grade of the two students and printing the average grade to the terminal.

Exercise 1 - Functions:

You are tasked to refactor the provided code, i.e. to improve the overall maintainability of the code without changing its end-user functionality. More specifically, you are asked to refactor the code calculating the average grade of a student by means of using a function. In exercise1 you do not have to change the code declaring and initializing the personal details of the two students.

Tips about the way of working: **First Think, then Implement, then Solve all compiler warnings, then Test, and lastly Reflect**

- First think of your strategy by means of design, then implement your design by means of writing C code. To design a function you are strongly recommended to document, e.g. as comment in your code, the prototype (a.k.a. signature, declaration) of your function. To design the prototype of your function you can take inspiration from the following example in Table 1. The following template has the advantage of properly documenting your function prototypes. Feel free to use your own (additional) way of designing functions (e.g. tables, block diagram, flow chart, etc).

```
/*
 * Function finds and returns the sum of the two inputs variables
 * Input(s):
 *      - a: first integer input
 *      - b: second integer input
 * Output(s):
 *      - sum of the two integer inputs
int addition(int x, int y);
```

Table 1: prototype of function addition

- When implementing your function, you are advised to first declare, then define, and lastly call your functions. All three steps are required when using functions in C. Where to place the code to print the result to the terminal? Inside or outside the function you created? Tip: think about separation of concerns.

- Do not forget to check and solve all compiler warnings! Why do think it is important to have an eye on compiler warnings?

- After implementing your functions do not forget to test your code. It is the responsibility of the developer to produce functional code. How are you going to test that your changes did not alter the functionality of the application?

- Lastly, reflect on the produced code. What are the advantages of using functions instead of the provided initial code? To your opinion was the refactoring effort eventually worthwhile?

Exercise 2 - Structures:

With exercise 2 you will continue improving the overall maintainability of the initial application for the administration of students. You are now challenged to refactor the code declaring and initializing the personal details of students by means of structures. You are strongly recommended to follow the generic way of working introduced with the previous exercise: Think, Implement, Solve all compiler warnings, Test, and Reflect.

Implementation tip(s):

- Do not forget to give a meaningful name to your structure, and more generically to all your variables and function names. Remember to choose a consistent naming convention. In C programming you can choose either *underscore_separation* or *capitalizedSeparation* for naming variables and functions. Examples:

```
int my_first_variable;     // underscore_separation
int mySecondVariable;  // capitalisedSeparation
```

- The application will eventually host the personal details of more than two students. How can you take this into account during refactoring? Would the use of an array of structure be an option to consider?

- Will you need to update the prototype and/or the implementation of the function you created in exercise1? Whichever your choice, can you substantiate?

Reflection tips:

- What are the advantages of using structures instead of the provided initial code? To your opinion was the refactoring effort eventually worthwhile?

Exercise 3 – Further improvements:

3.a) Advanced refactoring :

Note that the initial code hard-codes the size of arrays. Why do you think it is not a good practice? How can you refactor the code in order to avoid hard-coding the size of arrays?

Tips: To avoid hard-coding sizes of array in your code, you can think of (at least) one of the two options. What are the pros and cons of the two options below?

- Option1: Use a macro defined by a pre-compiler directive. The macro should be define on top of your *main.c* file before the main function. We will further study macro's in week3. Example:

```
#define MY_SIZE 5
…
…
int my_array[MY_SIZE];
```

- Option2: Automatically calculate the size of your array in your code using the *sizeof* operator of the C programming language. What does the *sizeof* operator do? Why is it needed to divide by *sizeof(my_array[0])*. Example:

```
int my_array[] = {1, 2, 3};
int size_my_array = sizeof(my_array) / sizeof(my_array[0]);
```

3.b) Extended functionality :

The refactored application is so far capable of calculating the average grade of two students and printing the average grade to the terminal. The end-users of the applicable appreciate your effort so far and would greatly appreciate if you would be able to read the personal data of students in a neat way. Can you update the application to satisfy the request from the end-users? You will probably need to use the *printf* function of the *cstdio* standard library. Which file do you need to include in order to use the *printf* function? Do not forget to follow the generic way of working introduced in previous exercises : Think, Implement, Solve all compiler warnings, Test, and Reflect.

Tip: you could think of printing the data to the terminal in a way similar to the example below.

```
=============================================================

Nr  Name          Number          Age      ….       ….
=============================================================

 1  Anna           214365          19       ….       ….

…

…

=============================================================
```