Self-Verification Test

This computer-based test requires the following equipment:

- C compiler
- Matlab

The test has a duration of three hours. It consists of three parts: Understanding and writing C code, as well as Matlab programming. The maximum score you can get is 100 points; while this self-verification test is ungraded, we ask you to upload your score at the end of the 3 hours in an anonymous survey on the Moodle server.

The test is open book, i.e. all printed/written material is allowed (e.g., books, lecture notes, exercises, solutions, personal notes) but, for sake of fairness across all students, you must use a computer from the computer room running a Linux operating system (Ubuntu). Note that such conditions will be enforced for all the graded tests of the course, so this will be also a good "dry run" from this perspective. **In parts II and III only** you can use internet to look for necessary information, but **do not communicate with other students. Additionally, as this is a self-assessment test, refrain from using chat-GPT and similar tools to answer the questions.**

Getting Started

To start with this test, you will need to download the material available on Moodle. Download `self-test.zip` and extract it in your home directory. The uncompressed folder has the following contents:

- *part2/rnd_print, part2/fibonacci, part2/struct_copy* and *part2/count:* files needed for Part II.
- *part3/:* Matlab files needed to answer Part III.

Part I: Understanding C Programming (32 points)

Do not use internet for this part.

1. (8pt): What is the resulting output of the following code?

```
1  #include <stdio.h>
2
3  int c = 5;
4  int d = 10;
5
6  int func1(int a, int b)
7  {
8      int c = a + b;
9      c *= 2;
10     return c;
11 }
12 int func2(int a, int b)
13 {
14     return a % b-- ;
15 }
16 void main(void)
17 {
18     int a = c + d;
19     int b = 8;
20     c = func1(a, b);
21     d = func2(a,b);
22     printf("%d %d %d %d \n", a, b, c, d);
23 }
```

2. (2pts): Why did we need to include the `stdio` library in the code of Question 1?
3. (4pts): Explain what is wrong with the following code snippet:

```
1  #include <stdio.h>
2
3  int* function(void)
4  {
5          int b = 5;
6          return &b;
7  }
8
9  void main (void)
10 {
11         int * ap = function();
12         *ap = 10;
13         printf("*ap = %d \n", *ap);
14 }
```

4. (8pt): The following function takes as arguments an integer array and the length of that array, and then modifies the array.

```
1  void mod_array(int* array, int length){
1     // iteration variables declaration
2     int i, j, tmp;
3     for(i=0; i<length; i++){
4           for(j=i+1; j<length; j++){
5                 if(array[i] > array[j]) {  // Array element comparison
6                       tmp = array[i];
7                       array[i] = array[j];
8                       array[j] = tmp;
9                 }
10          }
11    }
12 }
```

What is this function doing? Assume the array [3, 9, 2, 6] (length = 4) is passed to the function. What will be the modified array?

5. (4pts): Give two possibilities to return 4 values from a function.
6. (6pts): Building and running your C code:
    a. (2pts) What are the two main steps involved in building an executable?
    b. (2pts) What is the commonly used command to build an executable?
    c. (Bonus) Can the command above be simplified?
    d. (2pts) Assume you have successfully built a program using this command. This generated the file "hello_world". How do you run this executable on Ubuntu?

Part II: C Programming (38 points)

7. (10pts): Open the file rnd_print.c located in folder part2/rnd_print. Complete the main function to generate and print a random number between 1 and 100. Compile using make and test your function. Important: The number generated should be different every time you call the program. (Hint: useful functions are *rand()* and *srand()*)
8. (5pts): Open the file *main.c* located in folder *part2/fibonacci*. This file contains the main function of a program intended to print the n (user input) first Fibonacci numbers. In the *fibonacci.c* file, implement a function fibonacci(), which calculates and prints the fibonacci numbers. You'll also have to add the function declaration to the .h file. Compile using make and test your function. (*Hint: Fibonacci number N is defined as: F(N) = F(N-1) + F(N-2).)*

9. (8pts): Open the file *struct_copy.c* located in folder *part2/struct_copy*. This program contains two types of structures, A and B, where B is a simplified version of A (without the "address" field). Structure A already has an initialized instance.
   Write a function void struct_cpy(struct B* b, struct A a) that copies the common fields of A to B. Use the function struct_print(struct B b) to print the copied fields of structure B. Compile using make and test your function. (*Hint: you can use a function of the string.h library to copy strings.)*

10. (15pt): Open the file *count.c* located in folder *part2/count*. Write a program that takes an array of length L as user input and counts how many times numbers from 0 to 9 appear in it.
    Input: The first line of the input should be the length L of the array. The following lines should be the elements of the array, which are decimal numbers from 0 to 9.
    Output: The output of the program should print one line per number from 0 to 9. For each line, print the number and how many times it appears in the input array
    Example: In the following example, the user provides an array of letters of length 6 and its content: 2 5 7 2 2 5. The numbers that appear in this array are 2, 5 and 7 and they appear 3, 2 and 1 times, respectively. The program outputs one line for each number from 0 to 9 that prints the number and how many times it appears in the array.

Input	Output
➢ 6 // array length ➢ 2 // array content ➢ 5 ➢ 7 ➢ 2 ➢ 2 ➢ 5	➢ 0 0 ➢ 1 0 ➢ 2 3 ➢ 3 0 ➢ 4 0 ➢ 5 2 ➢ 6 0 ➢ 7 1 ➢ 8 0 ➢ 9 0

Hint: In the file count.c you fill find the declaration of the array count_array *and a function called* count. *Use* count_array *and the function* count *to store how many times a number appears. Understanding what is happening inside the function will help you to solve this problem.*

Part III: Matlab Programming (30 points)

11. (14pts) Open the file matrix_main.m located in folder part3/.
    a. (4pts) Create a function generate_random_matrix which returns a 5x5 matrix filled with random integers between 1 and 10.
    b. (2pts) In the main script, call the function created above to obtain matrix A and print it to the console line.
    c. (2pts) Set A(1,2) and A(3, 4) to zero
    d. (2pts) Calculate and print $B = A - A^T$, with A^T being the transpose of matrix A.
    e. (4pts) Calculate and print $C = A * A^T$, as well as $D = A \cdot A^T$. With * the standard matrix multiplication and · being element-wise multiplication.
12. (16pts) Plotting, open the file plotting.m located in folder part3/:
    f. (3pts) Generate a vector x that ranges from -2π to 2π with a step size of 0.1.
    g. (2pts) Calculate a corresponding vector "y" given by: $y = sin(x) + cos(2x)$.
    h. (3pts) Plot y as a function of x using a blue solid line.
    i. (2pts) Indicate the point (0,0) with a red O on the same plot.
    j. (2pts) Add a title to the plot, label the x and y-axes appropriately.
    k. (4pts) Find and print the maximum value of y and the corresponding value of x.