jgromero@inf.uc3m.es

Lesson 6. Functions
Exercises

Exercise 1. Write a function to calculate the *n*-th power of a number *x* –being *n* an integer value and *x* a double value.

Exercise 2. Write a function with two integer parameters that returns the largest of them.

Exercise 3. Write a program that presents the menu below and perform the selected operation. Use a function to print the menu (it must return the selected option) and three functions for each respective task (they must return the corresponding area).

```
Calculate the area
------------------------
1. Calculate the area of a triangle (base, height)
2. Calculate the area of a trapezoid (edge a, edge b, height)
3. Calculate the area of a rectangle (base, height)
```

Exercise 4. Write a function to swap the values of two integer variables.

Exercise 5. The final mark of the Programming lecture is accounted as follows: first partial exam 20%; second partial exam 20%; June exam 60%. Nevertheless, if the mark of the June exam is larger than the mark obtained with the previous calculation, this is the mark.

Develop a program to read three marks and calculate the final mark of a student.
Next, the program must ask the user whether a new mark should be calculated or the program should end. Write two functions: (i) a function to calculate the mark with the continuous evaluation; (ii) a function to obtain the larger of two values.

Exercise 6. Develop a function named *factorial* to calculate the factorial of an integer number.

Exercise 7. Based on the functions created in 1 and 6, develop a function named *approximation_e* to calculate an approximation to the number e^x with precision *n* according to the following formula:

$$e^x \approx \sum_{k=0}^{n} \frac{x^k}{k!}$$

The function must receive two parameters: *n* (integer value >= 0) and *x* (real). The function must return a real value with the approximation to e^x (real).

For instance:

- x = 1, n = 5

$$e^1 = e \approx \sum_{k=0}^{5} \frac{1^k}{k!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} \rightarrow 2{,}7166$$

- x = 2, n = 4

$$e^2 \approx \sum_{k=0}^{4} \frac{2^k}{k!} = \frac{1}{0!} + \frac{2}{1!} + \frac{2^2}{2!} + \frac{2^3}{3!} + \frac{2^4}{4!} \rightarrow 6$$

Exercise 8. Develop a program to calculate various approximations to e^x by using the method *approximation_e* for different values of *n*.

The program must read two values from the keyboard: *x* (real) and *max* (positive integer):

- *x* is the exponent of the approximation e^x
- *max* determines the maximum value of *n* that will be used to calculate the approximation; i.e., the method *approximation_e* will be executed for *n* = {1, 2, …, *max*}

The program must print on the screen the value of n, the value of the approximation for this *n*, and the value of e^x as obtained with the *exp* function of the math library (*math.h*).

For example, for x=2 and max=10, the output must be:

```
 n  approx   exp
------------------
 1 3.00000 7.38906
 2 5.00000 7.38906
 3 6.33333 7.38906
 4 7.00000 7.38906
 5 7.26667 7.38906
 6 7.35556 7.38906
 7 7.38095 7.38906
 8 7.38730 7.38906
 9 7.38871 7.38906
10 7.38899 7.38906
```

(Note that, due to range and precision of data types, the previous values may be different. The results for large values of *x* and *max* may result in values out of the range of the types and be incorrect.)

Exercise 9. Write a program to calculate the largest and the smallest values of a one-dimension array of integers. Use two functions to obtain each one of respective values.

Exercise 10. Write a program to calculate the largest and the smallest values of a one-dimension array. Use only one function to obtain both values.

Exercise 11. Develop functions to perform the following operations involving matrices – represented as two-dimension arrays.

(**NOTE**: Matrices size (number of rows and columns) must be defined as a constant with the directive `#define`).

a. Read values
   Parameters: `float m[ROWS][COLUMNS]`
   Returns: Nothing
   Tasks: Reads matrix values

b. Print matrix
   Parameters: Matrix `float m[ROWS][COLUMNS]`
   Returns: Nothing
   Task: Prints `m` on the screen

c. Find largest value of the matrix
   Parameters: Matrix `float m[ROWS][COLUMNS]`
   Returns: `double max`
   Task: Obtains the largest value of the matrix

d. Find smallest value of the matrix
   Parameters: Matrix `float m[ROWS][COLUMNS]`
   Returns: `double min`
   Task: Obtains the smallest value of the matrix

e. Add matrices
   Parameters: Matrices `double m1[ROWS][COLUMNS]`, `double m2[ROWS][COLUMNS]`, Matrix `double r[ROWS][COLUMNS]`
   Returns: Nothing
   Task: Calculates `r` = `m1` + `m2`

f. Subtract matrics
   Parameters: Matrices `double m1[ROWS][COLUMNS]`, `double m2[ROWS][COLUMNS]`, Matrix `double r[ROWS][COLUMNS]`
   Returns: Nothing
   Task: Calculates `r` = `m1` - `m2`

g. Scalar multiplication
   Parameters: Matrices `double m[ROWS][COLUMNS]`, `double x`, Matrix `double r[ROWS][COLUMNS]`
   Returns: Nothing
   Task: Calculates `r` = `x` x `m`

h. Matrix multiplication
   Parameters: Matrices `double m1[ROWS][COLUMNS]`, `double m2[COLUMNS][ROWS]`, Matrix `double r[ROWS][ROWS]`
   Returns: Nothing
   Task: Calculates `r` = `m1` * `m2`

i. Matrix transpose
   Parameters: Matrix `double m[ROWS][COLUMNS]`, Matrix `double r[COLUMNS][ROWS]`

Returns: Nothing
Task: Calculates `r = transpose(m)`

Write a main function to test the functioning of the implemented functions.

Exercise 12. Write a program implementing functions to operate with fractions. Fractions must be represented as 2-component structures. Operations are: addition, multiplication, division, opposite, and reciprocal –a function must be implemented for each operation.

Write a main function presenting a menu to the user to select the operation to perform – then, the program must read the corresponding data from the keyboard. Repeat until the user select 'Exit' option.

Exercise 13. Write a program implementing the functioning of a vending machine. The program will ask the user for the price of a product and the money introduced in the machine. The program will calculate the number of coins of each type that should be returned to the user. Let us assume that the machine has enough coins to return the change. The values of the coins are 1, 2, 5, 10, 20, 50 cents; 1, 2 Euros. Use functions to make the main function more readable.

Exercise 14. Extend the previous program to calculate the change, but assuming that the number of available coins is limited. (The number of available coins of each type can be declared as a constant.)

Exercise 15. Write a C program to test if a Sudoku solution is correct, according to the following rules:

a) The Sudoku board has 81 cells (9x9 board). The board is divided in 9 3x3 non-overlapping sub-boards.

b) A Sudoku board is correct when:
   a. The cells have values in {1, …, 9}
   b. A value cannot be repeated in a row
   c. A value cannot be repeated in a column
   d. A column cannot be repeated in a sub-board

3	8	1	9	7	6	5	4	2
2	4	7	5	3	8	1	9	6
5	6	9	2	1	4	8	7	3
6	7	4	8	5	2	3	1	9
1	3	5	7	4	9	6	2	8
9	2	8	1	6	3	7	5	4
4	1	2	6	8	5	9	3	7
7	9	6	3	2	1	4	8	5
8	5	3	4	9	7	2	6	1

Example of a correct solution