Programming 2

Ayman Makki (ayman.makki@uni.lu)
Semen Yurkov (semen.yurkov@uni.lu)

Lab 2 – Functions, Control Flow & Standard I/O

Exercise 5 – Factorial

#c :PrimitiveType #c :printf #c :PreprocessorInclude #c :FunctionDefinition #c :ForLoop #c :If #c :scanf

Write a C console application calculating the factorial of a non-negative number, defined as :

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

1° As the factorial function grows quite fast, which data type would you consider appropriate ? What format specifier will you need in order to display the result ?

2° Think about the constraints of the factorial function : what should you check (absolutely) before computing the factorial ? Do you have any idea how to do so ? (Hint : there is a good way and a "less good" way to do so).

3° In a separate header and source file `iterative.h/iterative.c`, write a function `factorial_iter` that calculates the factorial of a non-negative number, passed as argument to the function, in an *iterative* way.

4° In yet a different header and source file `recursive.h/recursive.c`, write a function `factorial_rec` which calculates the factorial in a recursive way.

5° Test your functions in a main program `main.c` :

- Read a value n from standard input using `scanf()`. Is there anything you should check from the input before computing the factorial ?
- Include both header files and call each function with the value of n to calculate $n!$, which shall be printed to the console.
- What would happen if you gave both functions the same name, e.g. `factorial` ? First think about the implications for the linker, then try it out and verify your (hopefully educated) guess.

Exercise 6 – Call by value

#programming :CallByValue

The C programming language employs *call by value* when passing arguments to functions. Write a console application to verify this statement. An example can be found under the above hashtag.

Exercise 7 – Calculator

#c :Operator #c :If #c :Switch #c :scanf #c :WhileLoop #c :printf

Write a calculator as a C console application containing the following elements :

- basic arithmetic operations : addition, subtraction, multiplication and division
- factorial of a (non-negative !) integer (you may take your solution from exercise 5)
- handling of the special cases of division by zero and the factorial of a negative number
- separate header and source files for the function implementations
- a `switch` statement for the selection of arithmetic operations as well as for the factorial
- user input of values and possibility to choose arithmetic operations
- possibility to continue with another calculation without the need to restart the program

Only use integer and float data types for input and output. You can assume that the user only enters integer or float values as instructed by the program. Inform the user about an invalid input with respect to the above mentioned special cases. Feel free to add any other functionality you deem relevant, such as trigonometric functions, defined in the `math.h` header file.

An example output could look like this :

```
Choose an arithmetic operation with '+', '-', '*', '/', '!' or press 'q' to quit: +
Enter two values to be added, separated by a space: 3.14 2.718
3.14 + 2.72 = 5.86
Choose an arithmetic operation with '+', '-', '*', '/', '!' or press 'q' to quit: !
Enter a value to calculate its factorial: 0
0! = 1
Choose an arithmetic operation with '+', '-', '*', '/', '!' or press 'q' to quit: q
```

☞ **Flushing the input buffer**

You might notice a "strange" behavior when expecting some input after another one, i.e. `scanf` consumes a line without you entering anything and hitting return.

This is due to the fact that the buffer still contains the `\n` character from the previous return. If your current input expectation is a numerical value, this character will not be consumed. However, if you are expecting, e.g., a `char`, then it will be !

A possible way to make sure that the input buffer is properly flushed is the following idiom :

```c
char buffer;
while((buffer = getchar()) != EOF && buffer != '\n');
```

✎ **Linking Libraries**

Depending on the version of gcc, the inclusion of the `math.h` library might require an additional linker flag `-lm`. If you are working from the command line, you may proceed as follows :

```
gcc -o main main.c -lm
```

Of course, this can be included in the Makefile, and we usually set up a variable named "CFLAGS" with all the compilation options required by our C programs.

☞ C99 Mode Error

gcc is not (necessarily) set to the C99 standard of the C programming language by default. This becomes clear, e.g., when declaring a variable inside a `for`-loop, as the following error message will be displayed :

```
CDT Build Console [C99_Error]
../src/C99_Error.c: In function 'main':
../src/C99_Error.c:16:2: error: 'for' loop initial declarations are only allowed in C99 mode
  for (int i = 0; i <= 10; i++){
  ^
../src/C99_Error.c:16:2: note: use option -std=c99 or -std=gnu99 to compile your code
make: *** [src/C99_Error.o] Error 1
```

Two possible solutions exist :

— declare the variable outside the `for`-loop
— add `-std=c99` to the gcc build options (and in the Makefile)