Exercises:

1. Program a recursive variant of the function *factorial()*
   a) Test and document the function. – Also try very large arguments and try to break *factorial()*!
2. Get some information about the Fibonacci series. Implement a method that prints the Fibonacci series up to a number, which is a parameter of the method.
   a) Test and document the method. – Also try very large arguments and try to break the method!
3. Create a function, to which a double is passed. This double should be rounded to an int and this int will be returned. How would you test this function?
4. A program should ask the user for five integers. The program then outputs the sum of squares. – The program resulting from the last exercise is the basis for this exercise.
   a) The individual actions in the program should be divided into meaningful functions (e.g. *acceptUserInput()*, *output()*, *showMenu()* …). Readability and avoidance of redundancy (i.e. obeying the DRY-principle) should be your primary targets in the implementation. All functions should be defined in the namespace *ExercisesProcedural* and need to be separated into h- and cpp-files. The function declarations should be adorned with comments.
   b) Show and document clearly, how you separately compile and link the resulting files on the console, e.g. using gcc/g++.

Remarks:

- If exercises ask to document something, a Word document with explanatory text, maybe incl. snippets and screenshots is awaited as companion artifact in the repository or sent as attachment to the solution of the exercise!
- When writing functions, apply separated compilation, i.e. separate h-files from cpp-files!
- The functions must have documentation comments, e.g. following the HeaderDoc convention.
- Everything that was left unspecified can be solved as you prefer.
- In order to solve the exercises, only use known constructs, esp. the stuff you have learned in the lectures!
- **Please obey these rules for the time being:**
  - **When using g++ (e.g. via Xcode), your code must successfully compile with the -pedantic compiler-flag, which deactivates any non-standard C++-extensions.**
  - **The usage of goto, C++11 extensions, as well as #pragmas is not allowed.**
  - **The usage of global variables is not allowed.**
  - **You mustn't use the STL, esp. *std::string*, because we did not yet understood how it works!**
  - **But *std::cout*, *std::cin* and belonging to manipulators can be used.**
  - **You mustn't use new and delete!**
- Avoid magic numbers and use constants where possible.
- The results of the programming exercises need to be runnable applications! All programs have to be implemented as console programs.
- The programs need to be robust, i.e. they should cope with erroneous input from the user.
- You should be able to describe your programs after implementation. Comments are mandatory.
- In documentations as well as in comments, strings or user interfaces make correct use of language (spelling and grammar)!

- Don't send binary files (e.g. the contents of debug/release folders) with your solutions! Do only send source and project files.
- Don't panic: In programming multiple solutions are possible.
- If you have problems use the Visual Studio help (F1) or the Xcode help, books and the internet primarily.
- Of course you can also ask colleagues; but it is of course always better, if you find a solution yourself.