Exercises:

```
int i = -1;
const int ic = i;
const int* pic = &ic;
int* const cpi = &ic;
const int* const pcic = &ic;
```

1) Which of the following statements is invalid? Why?

2) General questions:
   a) How can we pass an array by value?
   b) How can we pass an int by reference?
   c) How can we retrieve the length of an array?
   d) How can we append new items to an array?
3) Get acquainted with the usage of arrays. What happens if you create an array of size ten and:
   a) access the element at index 10?
   b) access the element at index -1?
   Document the results with screenshots if required.
4) Write a void-function that returns an int-value via a parameter.
5) Create a function that accepts two arrays as arguments (maybe more parameters must be added to implement this exercise). The function should copy the content of one array into the other array.
   a) Write a test program to test your function.
   b) Which error cases do you need to handle in this function?
6) What happens if you dereference a 0-pointer? Document the result with screenshots if required.
7) Re-implement the program that calculates to sum of squares with an array of five items.
8) Write a function, that inverts a "2-dimensional" array of 0s and 1s and print the result to the console.
9) Write your own variant of *std::strlen()*, *std::strcmp()* and *std::strcpy()* without using the original functions.
10) Write a recursive function, which writes each char of a cstring to the console separated by newlines.
11) Learn how to get and process command line arguments of a C++ program. Also learn how you can pass these arguments from within your IDE in order to debug the program. You need this information in the next exercise.
12) In another program the user should input a string via command line arguments. The program checks, whether the passed string is a palindrome. A palindrome is a "symmetric" word, i.e. such a word that can be read from left to right and from right to left having the same wording, e.g. "otto", "anna", "lagerregal". After the check following output should be generated: "anna is a palindrome". The case of the word's letters should be regarded, so "Otto" is no palindrome. The check for a word to be a palindrome should be implemented in a single function with following signature:

```
bool isPalindrome(const char* word);
```

A function, which tests an argument against a criterion is sometimes called predicate.

13) Create a program, which accepts a text as command line argument. This text should be mirrored and printed to the command line.
14) Create a new program, which accepts a text and some options as command line arguments. The program should count all vowels of the passed string. Another optional command line argument, -i, should tell the program to ignore the case, while counting the vowels. Additionally, the program should print a help text to the console, when the option -h is used.
15) Develop a function that awaits a source string and a substring. The result of this function is the count of occurrences of the substring in the source string. This function will have following signature:

```
int countSubString(const char* source, const char* subString);
```

The functionality of the function should be proved with a test program.

16) Create deep copy of a w-cstring using *std::memcpy()*.

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!**
  - **You are not allowed to use C++ references instead of pointers.**
- 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.