LAB 01 Introduction to Object-Oriented Programming with C++

- Do not rely solely on the lecture's content! Please consider browsing http://en.cppreference.com/w/cpp , http://www.cplusplus.com/reference , as well as any other document you may find useful.
- We use cmake to generate the `Makefile`. Generation rules are contained in `CMakeLists.txt`. How to build:
  - Create a build directory in `LAB01`, e.g. named `build`
  - Go in the build directory and execute `cmake ..` to generate a `Makefile`. The cmake parameter, here `..`, is the location where cmake will search the `CMakeLists.txt` file.

1. "Hello World"

Open `01_hello/hello.cpp`.

a. Compile and run.
b. Declare a new variable (for instance, `int i;`) but do not use it. Compile again. Uncomment the line beginning with `#set(CMAKE_CXX_FLAGS` in `CMakeLists.txt`. Compile again. Can you explain? Explore the effect of the W flags.
c. Execute the command `make clean` and then the command `VERBOSE=1 make.` What changes compared to the execution of `make` alone ?
d. Add some spurious whitespaces, some spurious empty lines… The goal is to keep the file correct (it must compile) while making it ill-formatted. Now execute the `beautify.bash` script. Check the source file. Convenient, isn't it?

2. First steps with streams

Open `02-streams/streams.cpp`.

a. Write a program that declares an int variable, assigns a value to it, then prints it to the standard output.
b. Modify the program to print the same value but without using any variable.
c. Modify the program so that it reads a numeric value (float) from the standard input, stores it in a variable, and finally prints it.
d. Modify the program to read, store and print two distinct numeric values.

Have a look at http://www.cplusplus.com/doc/tutorial/basic_io .

e. Write a program that reads a word (as text) from the standard input and prints it.
f. Modify the program to read an entire sentence (including spaces!) into a single variable and print it.
g. Modify the program so that it reads a line into a string, then extracts the first word as text and the second as a number, puts them into two distinct variables and prints them.

3. Deal with compile errors

The goal is to "get used" to compile-time errors, to understand them and to fix them.

Open `03-mess/mess.cpp`.

Fix all the errors so that the program compiles with the flags `-std=c++11 -Wall -Werror`. *Hint*: formatting the file may help, making it more readable.

NOTA BENE: removing lines of code is not the solution! However, you may need to reorder and modify them…

The expected output of the program is:

```
2
3
MyType{ myIntValue=5 }
MyType2{ myDoubleValue=5 }
```

4. Play with strings, files and control structures

The goal is to get more familiar with string manipulation, control structures (if/else, loops…) and file access. They are not object-oriented concepts, but they are handy basic bricks. Have a look to https://cplusplus.com/reference/string/string/

Open `04-strings/strings.cpp`.

Strings:

a. Write a program that declares a string variable initialized with the value "Hello World". Count the number of occurrences of a given character (for instance, 'o') in the string with a for loop. Also, print the length of the string.
b. Modify the program to append the value " and Hi to the Universe!" to the existing string.
c. Replace all the occurrences of the letter 'o' by '0' and all the occurrences of 'l' by '1' with a for loop.
d. Do the same as **c.** but using an iterator. See https://cplusplus.com/reference/string/string/ Member functions > Iterator > begin
e. Find the first occurrence of "Uni" in the string.
f. Is the string obtained after exercise **c.** equal to the string obtained after **b.**? Is it greater than the second? How do you asses that?

Files:

g. Write a program that opens the file `wdeeping.txt`, reads its content line by line and prints each line to the standard output. See https://cplusplus.com/reference/fstream/ifstream

h. Modify the program to count the occurrences of the word "the".
i. Modify the program to write a copy of the file where the line number is prepended to each line of text.

5. Functions

The goal is to get more familiar with function signatures and with the concept of "function overloading".

Open `05-functions/functions`.

a. Write and test a function `addAndPrint(int, int)` that sums two integer parameters and prints the result to the standard output
b. Add more functions with the same name and invoke each of them at least once in the same program:
   I. `addAndPrint(string, string)` (concatenation)
   II. `addAndPrint(double, double)`
   III. `addAndPrint(float, float)`
c. Invoke the function `addAndPrint(double, double)` and invoke it with values `(1.1, 2.2)` then with values `(1, 2.2)`.

6. First object

Open 06-circle/main.cpp

Write a class `Circle` that contains a public field/attribute `radius`.

a. Write a program that asks the user for a radius value, then
   I. Creates a Circle instance
   II. "Asks" the instance to compute its circumference, then prints it
   III. "Asks" the instance to compute its area, then prints it

You will need to add appropriate methods to the class, the circumference and area are computed when the related methods are invoked.

b. Modify the program so that it runs an "endless" loop: each iteration does the same as **a.** above. The loop stops when a negative value is entered.
c. *Bonus*: modify the loop of **b.** so that it stops when EOF (End Of File, ctrl+d) is obtained.
d. Move and split the Circle class description in two files: `Circle.hpp` which contains the class interface and `Circle.cpp` which contains the class implementation. The test program remains in `main.cpp`.
e. Change the `radius` access control to radius to private, add what is missing to change `radius` value.
f. Modify your code to compute the circumference and area only when the radius is changed.
g. Add a constructor to the class `Circle` which takes a radius as a parameter.