FACULTY OF COMPUTER SCIENCE
Programming II

Assessment guidelines

- Practicals must be done in teams of TWO STUDENTS throughout the whole semester.
- Practicals will be assessed with one of the following marks: `PASS` (which will have associated a numerical mark between 0.9 and 2 points) or `FAIL` (impossibility of passing the course). In order to have the opportunity to obtain a `PASS`, the submitted solution must first meet a series of minimum requirements that we will explain below.
- If deemed appropriate, lab teachers may call a meeting with the students to defend their work.
- In the section Evaluation of the Teaching Guide, other topics can be consulted, such as the consequences in case of plagiarism or practical delivery for the JULY opportunity.

Minimum requirements

The submitted solution must meet the following **minimum requirements** in order to be assessed. In the event that these were not met, the practical would be automatically assessed as `FAIL`:

- The source code of your submitted practicals must be compilable in *gcc* on the reference servers (IP `10.11.28.50`).
- In the course website, along with the instructions of each practical, we will make available a *script* and a set of text files to be used to test the proper functioning of your system. This script must work perfectly **without modifying any of its lines**.
- No global variables can be found.
- The code must be structured in procedures and/or functions, with at least one function or procedure for each operation of the main program.
- The code is properly annotated.

Other assessment criteria

When assessing your work, the following aspects will be taken into account:

- *Effectiveness*: that specifications are met, properly implementing all the required functionalities and also taking into account the *effectiveness* of the solution.
- *Efficiency*: the execution of unnecessary or clearly inefficient operations or processes must be avoided.
- *Error handling*: intensive control of all runtime errors.
- *Clarity*: the source code must be easily understandable, be properly indented and annotated, use of meaningful identifiers, etc.
- *Modularity*: modules must be interchangeable and reusable. The correct structuring of the main program into subprograms will be valued.