ECOTE Summer 2025 – regulations of laboratory classes

Supervisor

Agnieszka Malanowska (MSc)

Office hours: Thursday, 15.15-17.00

Room: 313A

Email: agnieszka.malanowska@pw.edu.pl

Organization of laboratories

1. Laboratory classes consist of preparation and presentation of subsequent phases of the project (**introductory documentation** – 5 points, **deadline: 23/24.04.2025**; **program** – 12 points, **deadline: 28/29.05.2025**; **final documentation** – 3 points, **deadline: 4/5.06.2025**). It is possible to obtain **20 points** for the project. Deadlines for each laboratory group are provided in the course regulations.

2. All phases have to be **presented** to the supervisor **in the classroom** at the given deadline or earlier. It is **not enough** to upload solutions on MS Teams.

3. The supervisor will wait for students in the laboratory for the first **15 minutes** of the classes. If no students appear in the first 15 minutes in the classroom, the supervisor will be available in her room during the rest of the laboratory time.

4. Projects can be consulted either during laboratory classes or during supervisor's office hours, but **consultations** at other terms may be impossible.

5. Introductory documentation **has to be accepted** by the supervisor before the program development. If it is not accepted, further phases will **not be assessed**.

6. Points for the given phase **cannot be changed after its deadline**. However, correction of solutions according to the suggestions of the supervisor is required.

7. Topics of projects are assigned via MS Teams before the first classes. Solutions also have to be sent via Assignments on MS Teams. The whole **solution should consist of** introductory and final documentations and source code of the program. Please, do not place executable code in your solutions.

8. **Final marks** are given on the last classes.

Projects

9. Both introductory and final documentation needs to be prepared according to a **template** available on the Studia server. Documentation needs to contain description of:

    1. all assumptions (e.g. selected subset of supported statements of the input language, selected notation in the case of many equivalent possibilities, chosen programming language used for implementation),
    2. requirements,

3. implementation details (including architecture of the program, necessary data structures, descriptions of modules and algorithms, specification of input and output – e.g. number and format of input files and the instructions to run the program), and
4. test cases (**for both correct and incorrect input**; test cases should include not only tested input, but also **expected output**).

The documentation has to contain **grammar of the input language**, even if it is some standard file format (e.g. JSON) or some subset of the existing programming language (in that case the grammar probably will have to be adjusted to the considered subset). If the project is dedicated to transformation of one file format to another, the **transformation rules** also have to be specified in the documentation.

10. Program can be written in **one of the following programming languages: C++, Java, C#, Python**, but ready implementations cannot be used for those parts of code which are related to the skills learnt in this course. For example, usage of built-in regular expressions or libraries/functions which parse input (e.g. json.loads in Python) is **forbidden**, the whole text processing should be written manually instead – solutions using built-in language mechanisms will **not be accepted**! In general, it is advisable to consult usage of external libraries and high-level functions.

11. In every topic, there are **various possibilities of correct contents** of input and output files. Contents presented in the topic descriptions are only the examples and can be different, as long as they are correct in a given file format / programming language and are compliant with the documentation.

12. Other implementation guidelines:

    1. Program parameters should be read from command line arguments, the user should not be asked for them interactively. Paths to test files cannot be hardcoded!
    2. Lazy evaluation should be used – the program has to read only one character from the input at a time, build only one token at a time, etc. Reading the whole file to the program memory or reading whole lines of input files is a bad solution!
    3. Programs have to be modularized (e.g. source reader, scanner, parser, transforming module)! Programs consisting only of 'parser' responsible for the whole processing will **not be accepted**.

13. **Error handling** has to be implemented for every topic. It includes, but is not limited to, handling errors arising from incorrect format of input.

13. The program needs to be tested for both correct and incorrect input. It is **not enough** to test the program only on the exemplary input provided in the topic description. Programs tested only for the exemplary input from the topic description will be evaluated to **0 points**! Unit tests testing all possibilities should be implemented.