1 Purpose of this Document

In this lecture the submission of exercise solutions will take place electronically using the platform GitLab (`https://about.gitlab.com/`), which is a web-based Git repository manager similar to GitHub and Bitbucket. Handing in your solutions digitally on such a platform is a good way of learning and training collaborative coding, which is commonplace in Scientific Computing research groups.

Please note that due to the large number of participants some rules for the submission of exercise solutions are necessary. There are predefined structures for the repositories to facilitate a speedy correction of submitted solutions, and there are guidelines you should follow when coding. Both are listed below.

Not following the rules for repositories may result in your solution not being graded. You will then have to reformat it and submit again. Not following the style guidelines may result in point deduction, with the amount of points depending on the impact your style could have on actual collaborative projects.

2 Your Repository on GitLab

2.1 Account

Please create an account on GitLab, using your name as it appears in MUESLI for the account details. This ensures that you will get credit for your submissions.

2.2 Project Name

Please form groups of three to four members and exchange contact details so you can find each other on GitLab. One of you should create a private (!) project and invite the rest of the group. The name of your project is `OOPFSC2017-Name1Name2Name3`, where `Name1` is the last name of the first student, `Name2` that of the second student, and so on. Further names should be added in the same way, and names should be sorted alphabetically. If your lineup changes for any reason during the course of the semester, please create a new project that reflects this and inform your tutors.

2.3 Directory Structure

Your repository should consist of two levels of directories, one for the exercise sheets and one for the individual exercises. This results in a tree structure like this:

1. `sheet1`
    a) `exercise1`
        i. `main.cc`
        ii. `solution.txt`
        iii. `<header and source files>`

    b) `exercise2`
        i. `<files as above>`
    c) `...`
2. `sheet2`
    a) `exercise1`
    b) `exercise2`
    c) `...`
3. `...`
4. `scratch`

You are free to name and structure your C++ files in any way you like, but there should always be a file named `main.cc` that is a sample application of your implemented classes. Often the content is given in the exercise, if it isn't you are free to choose sensible test cases yourselves.

The file `solution.txt` is meant for the output of your `main.cc` and for answering questions that appear in the exercises. If an exercise is of theoretical nature and doesn't include actual coding, this is the only file you have to submit.

The directory `scratch` is meant as an area where you can put temporary files, solution attempts and other things that aren't actual submissions. Any file put there will be ignored by the tutors, and can neither increase nor decrease the number of points you receive.

2.4 Access to the Repository

Please give your tutor write access to your project so that your submissions can be graded. You also need to give read access to the repository to all tutors if there is more than one, regardless of the group you are in. This allows for flexible handling of unforeseen situations.

3 Style Guidelines

All programs you submit should follow basic programming rules like the following:

- Formatting
  - Put each instruction on a separate line (two lines if it is very large)
  - Use spaces to separate variable names, functions and operators
  - Indent your lines to visually mark different lines belonging to different scopes
- Variable names
  - The name should reflect the purpose of the variable

  - Variable names start with a lowercase letter, types with an uppercase letter
  - The rest of the name format for identifiers is up to you
  - Simple counting `int`s and similar variables are exempt

- Comments
  - Comments allow others to understand your intentions
  - Tutors can only give you points if they understand what you were trying to do
  - Guideline: one comment per class, per function, per algorithm subtask, per tricky or "exotic" source line
  - Don't comment to much either, this may visually drown the actual code or diverge from what is actually coded (!)
  - Leave out trivial comments ("This is the constructor")

- Language constructs
  - You may use any construct you want, even if it has not yet been introduced in the lecture
  - Avoid constructs that have been superseded by better alternatives in the lecture
  - Declare variables and references as `const` where it is possible
  - Separate interface and implementation by correctly using `public` and `private`
  - Use exceptions instead of asserts / aborts and smart pointers instead of raw pointers once the lecture has introduced them