Raphael Braun
Ayesha Feroz
Kyowon Jeong
Matteo Pilz
Lukas Ruppert
Arslan Siraj
Faezeh S. Zakeri

Eberhard Karls
Universität
Tübingen

Programming in C++

Sheet 8

Submission date: 13.10.2022

8.1 Transformation and Generation (60P = 20P + 10P + 20P + 10P)

C++

In the template exercise you implemented your own `reduce` and `map` functions. In this task you will use `std::transform` from `<algorithm>` in combination with lambda functions.

a) Implement the Mean Squared Error (`MSE`) function in `submission/exercise_81.cpp` using variant (3) of `std::transform` for the squared difference and variant (1) from `std::accumulate` for the sum. The MSE is defined as:

$$MSE(\text{gt}, \text{pred}) = \frac{1}{N}\sum_{i=1}^{N}(\text{gt}_i - \text{pred}_i)^2 \tag{1}$$

Where N is the size of both gt and pred, gt_i is the i-th element in gt and pred_i is the i-th element of pred.

Take a look at the example in `main.cpp` to see a working example for the usage of a lambda function with `std::transform`.

b) Implement the Mean Absolute Error (`MAE`) function in `submission/exercise_81.cpp` using the same tools as for the MSE. The MAE is defined as:

$$MAE(\text{gt}, \text{pred}) = \frac{1}{N}\sum_{i=1}^{N}|\text{gt}_i - \text{pred}_i| \tag{2}$$

c) Implement a template function `zip` in `submission/exercise_81.h` using `std::transform`. Given two vectors of same length but arbitrary element types `T` and `U` `zip` should combine them into one vector with element type `std::pair<T, U>`. Make sure the call given in `main.cpp` works.

d) Implement the `range(int start, int end)` function in `submission/exercise_81.cpp`. The function returns a vector of integers with all numbers between `start` and `end`, including `start`, excluding `end`.

Please use `std::generate` in with a lambda function as generator in your implementation. You will need to capture `start` and make it mutable. This allows you to return `start++` in the body.

8.2 Sort and Median (40P = 15P + 15P + 10P)

C++

In this task you will use `std::sort` to sort and eventually extract a median point from a list of 2D-points. A point is represented by a `Point` object - whose struct definition you find in `submission/point.h`.

a) Implement the function `sort_x` in `submission/exercise_82.cpp`. Use `std::sort` with a lambda function that defines the order such that the points are first sorted by the x, then by the y-coordinate.

b) Implement the function `sort_y` in `submission/exercise_82.cpp`. Use `std::sort` with a lambda function that defines the order such that the points are first sorted by the y, then by the x-coordinate.

c) Implement the function `median` in `submission/exercise_82.cpp`. The returned point should contain the median coordinates of the x- and y-direction. Use your `sort_x` and `sort_y` functions. We use the median definition from Wikipedia.