GP 2	Generative Programming	ST 17, Exercise 1

Deadline: 31.03.2017, 13:50

☐ **Gr. 1,** J. Karder, MSc.
☐ **Gr. 2,** P. Fleck, MSc.

Name _____ **Effort in h** _____

Points _____ **Lecturer** _____

1. ... like Bunnies (2 + 2 Points)

Since your first programming exercises (in the long gone days of your bachelor studies) you know the famous Fibonacci sequence which is defined for all positive integers *n*:

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n) = fib(n-1) + fib(n-2)$$

Perhaps you also remember that the Fibonacci sequence is not only a well-known example of recursive functions, but is also of unbelievable practical use: Using the Fibonacci sequence you can compute the size of an (ideal) bunny population at time *t*.

Due to this immense importance of the Fibonacci sequence, we would like to focus on it in this first task. Anyhow, the computation of the sequence should now not be done at runtime but during compilation.

a) Write a C++ template metafunction *Fibonacci* which computes the *n*-th Fibonacci number at compile time. Demonstrate how the function can be used by computing *Fibonacci<40>*. For comparison, also implement a recursive function *fibonacci* which computes the Fibonacci numbers in the conventional way at runtime.

b) Extend your program so that it lists the first 40 Fibonacci numbers. Thereby the numbers should be computed on the one hand at compile time using template metafunctions (e.g., *DO* or *WHILE*) and on the other hand conventionally at runtime.

2. 1, 2, 3, 4, 5, 6, 7, … (6 + 6 Points)

a) Implement a class *Counter* which provides the methods *reset*, *increment* and *value* to access and manipulate its counter value. The data type and the initial value of the counter value should be configurable. Use a *traits class* to pass all required configuration information to the *Counter* (data type and initial value).

b) Furthermore, two additional optional configurations should be available for counters. On the one hand it should be possible to define an upper limit for the counter value, which must not be exceeded (*BoundedCounter*). On the other hand the step size of the counter should be configurable (*VarIncrementCounter*). Use the concept of *inheritance-based wrapper classes* to realize specializations of *Counter* for these two configurations.

   Test different configurations (e.g., *IntCounter*, *DoubleCounter*, *BoundedIntCounter*, *VarIncrementIntCounter*, *BoundedVarIncrementIntCounter*, …)

3. Counters off the Shelf (8 Points)

Similarly to the *LIST_GENERATOR* example which has been discussed in the lecture, implement your own generator *COUNTER_GENERATOR* to create suitable configurations for your *Counter* class automatically.

Thoroughly think about, how you can pass all the different values to the generator which are required in the configuration (initial value, upper bound, step size).

Test your generator extensively by creating and using different counter variants.