Object-Oriented Programming in C++ (SS 2022) Exercise 5: June 9, 2022

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at

February 2, 2022

The exercise is to be submitted by the denoted deadline via the submission interface of the Moodle course as a single file in zip (`.zip`) or tarred gzip (`.tgz`) format which contains the following files:

- A PDF file `Exercise`*`Number-MatNr`*`.pdf` (where *Number* is the number of the exercise and *MatNr* is your "Matrikelnummer") which consists of the following parts:
  1. A decent cover page with the title of the course, the number of the exercise, and the author of the solution (identified by name, Matrikelnummer and email address).
  2. For every source file, a listing in a *fixed width font*, e.g. `Courier`, (such that indentations are appropriately preserved) and an appropriate *font size* such that source code lines do not break.
  3. A description of all tests performed (copies of program inputs and program outputs) explicitly highlighting, if some test produces an unexpected result.
  4. Any additional explanation you would like to give. In particular, if your solution has unwanted problems or bugs, please document these explicitly (you will get more credit for such solutions).
- Each source file of your solution (no object files or executables).

Please obey the coding style recommendations posted on the course site.

no inheritance and no virtuality required

Exercise 5: Recursive Polynomials by Templates

The goal of this exercise is to implement a class whose objects represent polynomials with integer coefficients, as in Exercise 4. However, in contrast to Exercise 4, the implementation shall now be based on a class template; thus genericity is achieved by parametric polymorphism rather than by inheritance.

In detail, your tasks are as follows:

1. First implement a class template `template<class Ring> class RecPoly` such that the objects of the resulting class represent univariate polynomials over the parameter domain *`Ring`*.

   The parameter *`Ring`* is assumed to denote any class that provides the same operations as the class `Ring` of Exercise 4 (please note that *`Ring`* denotes here a formal parameter, the class `Ring` of Exercise 4 is *not* needed in this exercise). The representation and functionality of class template `RecPoly` is analogous to that of the class of Exercise 4. However, since the class resulting from the instantiation of this template is is not designed for inheritance with overriding, the operations need not be `virtual`.

2. Next, use class template `RecPoly` and class `Integer` (you may use the same class as in the previous exercise) to create by the definition

   ```
   typedef RecPoly<Integer> UniPoly;
   ```

   a type `UniPoly` that implement univariate polynomials with integer coefficients. This class shall have the same functionality as the corresponding class of Exercise 4.

3. Finally, create by the definition

   ```
   typedef RecPoly<UniPoly> BiPoly;
   ```

   a type `BiPoly` that implements bivariate polynomials with integer coefficients. This class shall have the same functionality as the corresponding class of Exercise 4.

Test both types in the same way as in Exercise 4.