Computer Systems (SS 2013)
Exercise 5: June 3, 2013

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

May 14, 2013

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.

Exercise 5: Generic Polynomials

1. Implement a template class `template<typename` *`Ring`*`> TPoly` whose objects represent univariate polynomials over the coefficient domain *`Ring`*.

   Here the parameter *`Ring`* is assumed to represent a class that supports the same operations as those in Exercise 4 *except* that as arguments `Ring` values (not pointers) are passed and as results `Ring` values (not pointers) are returned. Furthermore, a class function `Ring::str(r)` is assumed to return the string representation of r and a class constant `Ring::zero` shall represent the neutral element of the ring (rather than the corresponding object functions of Exercise 4).

   The representation and functionality of class `TPoly` is analogous to that of class `GPoly` of Exercise 4 *except* that the internal array stores `Ring` values (not pointers); arguments and results of the various operations shall be of type `const Ring&` and `Ring` (not `Ring*`) respectively. Since the class is not designed for inheritance, the operations need not be `virtual`.

2. Implement a class `Rational` that may be appropriately substituted for *`Ring`*; an object of type `Rational` encapsulates a rational number as a coefficient (in analogy to the class of Exercise 4). The arguments the various operations are to be of type `const Rational&` (rather than `const Ring*`) and results are to be of type `Rational` (rather than `Ring*`).

3. Use `TPoly` and `Rational` to derive a class `Poly` that implements polynomials with rational coefficients:

   ```
   class Poly: public TPoly<Rational> { ... };
   ```

   This class shall have the same functionality as the corresponding class of Exercise 4 except that the result of function `evaluate` shall be of type `Rational` rather than `Rational*`.

Test class `Poly` in the same way as in Exercise 4.