Computer Systems (SS 2014)
Exercise 5: June 2, 2014

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

May 8, 2014

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 Multivariate Polynomials

The goal of this exercise is to implement a class `RPoly` whose objects represent multivariate polynomials with rational coefficients, as in Exercise 4. However, in contrast to Exercise 4, the implementation shall be in this exercise based on a class template `TPoly`; 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> TPoly` such that the objects of the resulting class represent multivariate polynomials over the domain `Ring::type`. The class *Ring* is assumed to hold by a declaration of form

   ```
   typedef ... type;
   ```

   a type `Ring::type` that represents the carrier set of the ring.

   The class provides the same operations as those in Exercise 4 *except* that all ring operations are (static) class members and functions and that as function arguments respectively results `Ring::type` values (not pointers) are passed respectively returned. For instance, a class function `Ring::str(r)` is assumed to return the string representation of a `Ring::type` value *r* and a class constant `Ring::zero` of type `Ring::type` shall represent the neutral element of the ring. Furthermore, the inverse operation has one argument of type `Ring::type` while addition, multiplication and comparison have two such arguments. By this setup `Ring::type` can be any (also an atomic) type.

   The representation and functionality of class template `TPoly` is analogous to that of class `GPoly` of Exercise 4 *except* that the internal list stores `Ring::type` values (not pointers) and that the first arguments of the binary function `add` shall be of type `Ring::type`. Since the class resulting from the instantiation of `TPoly` is is not designed for inheritance, the operations need not be `virtual`.

2. Then implement a class `DoubleRing` (where `DoubleRing::type` denotes `double`) that may be substituted for *Ring*; test with this class the functionality of class `TPoly<DoubleRing>` which represents multivariate polynomials with floating point coefficients; the operations of `DoubleRing` must thus be implemented by the primitive operations on `double`.

3. Next implement a class `RatRing` that may be substituted for *Ring*; declare in this class

   ```
   typedef Rational* type;
   ```

   where `Rational` is a class that encapsulates a rational number; you may reuse here the class of Exercise 4. Please note that the various class operations of `RatRing` must thus take and return pointers to `Rational` objects; these operations internally call the corresponding operations in `Rational`.

4. Finally, use `TPoly` and `RatRing` to derive a class `RPoly` that implements polynomials with rational coefficients:

   ```
   class RPoly: public TPoly<RatRing> { ... };
   ```

   This class shall have the same functionality as the corresponding class of Exercise 4.

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