Computer Systems (SS 2012)
Exercise 5: June 4, 2012

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

May 15, 2012

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

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

Here *Coeff* is assumed to be (in analogy to the class of Exercise 4) a class with public functions

```
// prints coefficient on standard output stream
virtual void print() const;

// pointer to sum, and product
virtual const Coeff* operator+(const Coeff* c) const = 0;
virtual const Coeff* operator*(const Coeff* c) const = 0;
```

The representation and functionality of class `TPoly` is analogous to that of class `GPoly` of Exercise 4.

Second, implement a class `Double` that may be substituted for *Coeff*; an object of type `Double` encapsulates a double precision floating point number as a coefficient (in analogy to the class of Exercise 4).

Third, use `TPoly` and `Double` to derive a class `Poly` that implements polynomials with double precision floating point coefficients and has the same functionality as the class of Exercise 4:

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

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