Computer Systems (SS 2014)
Exercise 4: May 19, 2014

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

April 23, 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 4: Generic Multivariate Polynomials

The goal of this exercise is to implement a class `RPoly` whose objects represent multivariate polynomials with rational coefficients. The implementation shall be based on a generic polynomial class `GPoly` which works for arbitrary coefficient types that support the usual ring operations.

In more detail, the implementation shall work as follows:

1. Take the following abstract class `Ring`:

```
class Ring {
public:
  // destructor
  virtual ~Ring() {}

  // string representation of this element
  virtual string str() const = 0;

  // the constant of the type of this element and the inverse of this element
  virtual const Ring* zero() const = 0;
  virtual const Ring* operator-() const = 0;

  // sum and product of this element and c
  virtual const Ring* operator+(const Ring* c) const = 0;
  virtual const Ring* operator*(const Ring* c) const = 0;

  // comparison function
  virtual bool operator==(const Ring* c) const = 0;
};
```

2. Implement the generic polynomial class `GPoly` which provides the same operations as the class `Polynomial` of Exercise 2 (and is internally implemented in an analogous way) except that its operation `add` has interface

```
GPoly& add(Ring* coeff, int* exps);
```

   i.e., we can add to the polynomial a monomial whose coefficient has as its type a (pointer to a) concrete class that is derived from the abstract class `Ring`. The internal representation thus has to operate with and store `Ring*` values (class `GPoly` does *not* know and use the class `Rational` described below).

3. Derive from `Ring` a concrete class `Rational`; every object of this class encapsulates a rational number represented by the numerator and denominator (a pair of `int` values that are relatively prime such that the nominator is greater than zero; use the Euclidean algorithm to divide common factors). The derived class provides implementations for all abstract operations of the base class (and possibly some extra operations).

Note that in the definition of the arithmetic and comparison functions the parameter `c` must be explicitly converted from type `const Ring*` to type `const Rational*`. Use `dynamic_cast<const Rational*>(c)` to receive a pointer to the corresponding `Rational` object (respectively `0`, if the conversion is not possible; the program may then be aborted with an error message).

4. Derive from `GPoly` the concrete class `RPoly` whose components denote rational numbers. The interface of this class supports (by inheritance) the same operations as those of `GPoly` but also provides an additional operation

   ```
   RPoly& add(int n, int d, int* exps);
   ```

   which allows to add a monomial with coefficient n/d to the polynomial. It is primarily this operation by which the user builds polynomials with rational coefficients (the method internally constructs a `Rational` object and calls the method `add` inherited from `GPoly`).

Test classes `Rational` and `RPoly` in a *comprehensive* way (several calls of each function) in a similar way as in Exercise 2 (print the function results and show the output in the deliverable).