Computer Systems (SS 2010)
Exercise 3: May 4, 2010

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.uni-linz.ac.at

April 9, 2010

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).

Exercise 3: Generic Polynomials

Implement a class `GPoly` whose objects represent univariate polynomials over a generic coefficient domain. The functionality and representation is essentially the same as in Exercise 1 with the exception that the coefficient type is an abstract class `GPoly::Coeff` that provides the required coefficient operations as virtual functions:

```
class GPoly::Coeff {
public:
  // destructor and assignment
  virtual ~Coeff() {}
  virtual Coeff& operator=(const Coeff &c) = 0;

  // prints coefficient on standard output stream
  virtual void print() const = 0;

  // returns pointer to copy of this coefficient
  virtual Coeff* copy() const = 0;

  // addition, multiplication, comparison, check for =0 and >0
  virtual Coeff* operator+(const Coeff& c) const = 0;
  virtual Coeff* operator*(const Coeff& c) const = 0;
  virtual bool operator==(const Coeff& c) const = 0;
  virtual bool isZero() const = 0;
  virtual bool isPositive() const = 0;
};
```

Derive from `GPoly` a concrete class `Poly` which provides (on top of the functionality inherited from `GPoly`) the same functionality as the class of Exercise 1 (with double precision floating point numbers as coefficients).

For this purpose, derive from `GPoly::Coeff` a concrete class `Poly::Coeff`; every object of this class encapsulates a double precision floating point number. Use for coefficient comparison the approximative equality of Exercise 1.

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

Test class `Poly` in the same way as in Exercise 1.