Computer Systems (SS 2010)
Exercise 2: April 20, 2010

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

March 19, 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).

Please obey the coding style recommendations posted on the course site.

Exercise 2: Multivariate Polynomials

A k-variate polynomial $\sum_{i=1}^{n} m_i$ is essentially a sequence of k-variate monomials $m_1, \ldots, m_n$ where each monomial $m_i = c_i \cdot x_1^{e_1} \cdots x_k^{e_k}$ consists of a coefficient c_i and k exponents $e_1, \ldots, e_k$.

Using this idea, write a class `MPoly` whose objects represent k-variate polynomials with double precision floating point coefficients. The class has an inner class `MPoly::Mono` whose objects represent k-variate monomials; the exponent vector of a monomial is represented by a heap-allocated array of unsigned integers. An `MPoly` object is essentially a linked list of `Mono` objects, use for list nodes an inner class `MPoly::Node`.

With these classes it shall be possible to execute the following commands:

```
// construction of monomials from coefficient and exponents
unsigned int e1[] = { 3, 1, 2 };
MPoly::Mono m1(1.5, 3, e1);       // 1.5*x^3*y*z^2

unsigned int e2[] = { 1, 0, 1 };
MPoly::Mono m2(-2.1, 3, e2);      // -2.1*x*z

unsigned int e3[] = { 0, 2, 0 }
MPoly::Mono m3(-1, 3, e3);        // y^2

// set absolute and relative accuracy of coefficient comparisons
MPoly::setAccuracy(0.000001, 0.001);

// P = (1.5*x^3*y*z^2) + (-2.1*x*z) + (y^2)
MPoly::Mono ms[] = { m1, m2, m3 };
MPoly p(3, 3, ms);  // p has 3 variables, ms has 3 monomials

// number of variables
int m = p.vars();  // 3

// evaluation of b for x=1.0, y=1.0, z=0.0
double xs[] = {1.0, 1.0, 0.0};
double value = p.eval(xs); // 3.5

// initialization of q by p
MPoly q = p;

// assignment of p to q
q = p;
```

```
// addition of p and q giving r; p and q are not modified
MPoly r = p+q;

// variables to be used in printing
char* vars[] = { "x", "y", "z" };
p.print(vars);
```

For adding a monomial m to a polynomial p, you must determine whether p already contains a monomial with the same exponents as m:

- If no, in p a new monomial identical to m has to be created.
- If yes, the coefficient of m is added to the coefficient of the corresponding monomial; if the resulting coefficient is zero, the monomial vanishes from the polynomial.

Consequently, make sure that the resulting polynomial has not multiple monomials with identical exponents and no monomials with coefficient 0. Choose a natural representation of the zero polynomial.

The same considerations apply in the initial construction of p from an array a of monomials (for every monomial m in a) and when adding a polynomial q to a polynomial p (for every monomial m of q). Please note that the same form of "approximative equality" as in Exercise 1 is to be used.

Also like in Exercise 1, object representations are never shared, i.e. exponent vectors have to be copied when creating a new monomial and monomials have to be copied when they are inserted into a polynomial. Also no memory leaks may arise, i.e. whenever a class allocates an object/array, it is also responsible for deallocating it in its destructor. Also avoid any code duplication but make extensive use of auxiliary functions (that shall become private member functions of `MPoly`).

Try to make efficient use of the internal data representation; e.g., when implementing $p + q$, run over the linked list of p to determine its individual monomials rather than applying repeatedly an indexing operation $p[i]$ to determine the i-th monomial (if p has n monomials, this would increase the time complexity of polynomial addition by a factor n).

Organize your code into files `MPoly.h`, `MPoly.cpp`, and `MPolyTest.cpp` in analogy to Exercise 1. Test each operation with at least three test cases that also include special cases (such as addition by 0).