Computer Systems (SS 2009)
Exercise 5: May 26, 2009

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

May 7, 2009

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 to 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: Polynomials

Take the following interface for an univariate polynomial over a coefficient ring C with operations `+` and `*`:

```
template<typename C> class Polynomial
{
  public:
    typedef C Coefficient;
    typedef unsigned int Exponent;
    virtual ~Polynomial() { } ;

    // the highest exponent
    virtual Exponent getDegree() = 0;

    // the coefficient with exponent i
    virtual Coefficient operator[](Exponent i) = 0;

    // evaluation of polynomial on element x
    virtual Coefficient evaluate(Coefficient x) = 0;

    // addition of two polynomials
    virtual Polynomial operator+(Polynomial& p) = 0;
};
```

Derive from `Polynomial` a (non-abstract) template class `DensePolynomial<C>` with constructor

```
// create a polynomial of denoted degree with denoted coefficients
// coeffs is an array of polynomials of length degree
// where coeffs[i] represents the coefficient of exponent i
DensePolynomial(Exponent degree, Coefficient* coeffs)
```

that implements a polynomial by an array of all coefficients for all exponents up to the degree of the polynomial.

Also derive from `Polynomial` a (non-abstract) template class `SparsePolynomial<C>` with constructor

```
// create a polynomial from a sequence of monomials
// exps and coeffs are arrays of length number that represent the monomials
// exps is in strict increasing order and coeffs does not contain zeros
// monomial i is represented by exponent exps[i] and coefficient coeffs[i]
SparsePolynomial(int number, Exponent *exps, Coefficient* coeffs)
```

that implements a polynomial by an two arrays of exponents and coefficients representing all monomials with non-zero coefficients.

Both classes shall have exclusive access to their internal representation (i.e. the constructor arguments must be duplicated) and shall take care of appropriate memory cleanup on destruction.

Also implement a generic function

```
// print polynomial p on stream out using var as the name of the variable
template<typename C>
void print(ostream &out, Polynomial<C> &p, const string var = "x") { ... }
```

and use this function to test above classes *in an extensive way* also considering special cases (the polynomial "0" and other polynomials of degree 0); in particular, test the operation `+` by

- adding to a dense polynomial a dense polynomial (the result is a dense polynomial),
- adding to a sparse polynomial a sparse polynomial (the result is a sparse polynomial),
- adding to a dense polynomial a sparse polynomials (the result is a dense polynomial),
- adding to a sparse polynomial a dense polynomials (the result is a sparse polynomial).

The code must be purely generic (no type query operations like `dynamic_cast` or `typeid` are allowed).