Computer Systems (SS 2012)
Exercise 6: June 18, 2012

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

May 22, 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 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 6: Polynomials by Sequence Containers

Take the following interface for a univariate polynomial over a coefficient ring `Coeff` with constants `0` and `1` and operations `+` and `*`; we also assume that a coefficient can be printed using the operator `<<` (think of `int` as an example of such a domain):

```
template<typename Coeff> class Polynomial {
public:
  // destructor
  virtual ~Polynomial() {}

  // degree of polynomial
  virtual int degree() const = 0;

  // coefficient of monomial with power i (<= degree)
  virtual const Coeff operator[](int i) const = 0;

  // evaluate polynomial on given value
  virtual const Coeff eval(Coeff c) const = 0;

  // print polynomial using  the given string for the variable
  virtual void print(string& var) const = 0;
};
```

Derive from `Polynomial` a non-abstract template class

```
  template<typename Coeff>
  class DensePolynomial: public Polynomial<Coeff> {...}
```

with constructor

```
  // create polynomial of denoted degree with denoted coefficients
  // 'coeffs' is a coefficient array of length 'degree+1'
  // where 'coeffs[i]' represents the coefficient of exponent 'i'
  DensePolynomial(int degree, Coeff* coeffs)
```

that implements a polynomial by the sequence of all coefficients. This sequence is to be implemented with the help of the standard library as an object of type `vector<Coeff>`; the implementation shall as far as possible make use of the operations that are already available on this type.

Also derive from `Polynomial` a non-abstract template class

```
  template<typename Coeff>
  class SparsePolynomial: public Polynomial<Coeff> {...}
```

with constructor

```
// create a polynomial from a sequence of 'number' monomials
// each represented by an exponent 'exps[i]' and coefficient 'coeffs[i]'
// 'exps' is in strict increasing order, 'coeffs' does not contain zeros
SparsePolynomial(int number, int* exps, Coeff* coeffs)
```

that implements a polynomial by the sequence of all monomials with non-zero coefficients. The sequence is to be implemented with the help of the standard library as an object of type `list<Mono>` (where `Mono` is a user-defined monomial type); the implementation shall as far as possible make use of the operations that are already available on this type.

Please note that by the use of the standard library classes, it is not necessary to explicitly allocate heap memory with the operator `new`; thus there is also no need to re-define the default copy constructors, copy assignment operators, and destructors.

Finally implement the generic function

```
// draws polynomial in interval [from, to] where
// x,y denotes the position of the coordinate center and
// h,v denote horizontal/vertical scaling factors
// we assume that "static_cast<double>(c)" is possible on a Coeff value c
template<typename Coeff>
void draw(const Polynomial<Coeff> &p,
          int from, int to, int x, int y, double h, double v);
```

Use this function and the object function `print()` to test above classes *in an extensive way* on polynomials of type `DensePolynomial<double>` and `SparsePolynomial<int>`.