Computer Systems (SS 2009)
Exercise 6: June 9, 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 6: Polynomials

Take the interface `Polynomial` of Exercise 5 and implement two template classes for dense and sparse polynomials as

```
template<typename C, template<typename> class S>
  class DensePoly: Polynomial<C> {...S<C>...}
template<typename C, template<typename> class S>
  class SparsePoly: Polynomial<C> {...S<C>...}
```

where `S` is assumed to be a class template that provides those operations that are common to all sequence containers of the C++ standard library (please note that `operator[]` is *not* among these operations). Both polynomial classes shall use objects of type `S<C>` for their internal representation and use *iterators* to process these objects.

Define template classes that use the C++ standard containers `vector` and `list` for their internal representation:

```
template<typename C> class DensePolynomialVector:
  public DensePoly<C,vector> {...}
template<typename C> class DensePolynomialList:
  public DensePoly<C,list> {...}
template<typename C> class SparsePolynomialVector:
  public SparsePoly<C,vector> {...}
template<typename C> class SparsePolynomialList:
  public SparsePoly<C,list> {...}
```

and define as in Exercise 5

```
// 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") {...}
```

Test the operations of above classes as in Exercise 5.