Computer Systems (SS 2009)
Exercise 4: May 12, 2009

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

March 31, 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 4: Generic Hash Tables

Implement a template class `HashTable<`*`Value`*`,`*`Info`*`>` whose objects represent hash tables of a certain size holding values of type *Value* and where *Info* is a class with public functions

```
// get hash value of o
static unsigned int hashValue(const Value& o);

// true if o1 and o2 are considered equal
static bool equals(const Value& o1, const Value& o2);
```

The hash table is implemented (analogous to the hash table of Exercise 3) as an array of linked lists of nodes where every node holds a pointer to a *Value* object (for this purpose, auxiliary template classes have to be introduced); the functionality of the hash table is implemented with the help of the functions provided by *Info*.

Use this template to derive a text statistic class (similar to Exercise 3) as

```
class Text: protected HashTable<string, StringInfo> { ... }
```

where `StringInfo` is a class of the kind discussed above with operations representing the hashing and comparison of strings.

Test class `Text` in the same way as in Exercise 2.