Computer Systems (SS 2009)
Exercise 3: April 28, 2009

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

March 27, 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 3: Hash Tables

Implement a class `HashTable` whose objects represent hash tables of a certain size. The class shall have the following public members:

```
static const int SIZE = ... ; // default table size

// constructs hash table with n slots
HashTable(int n = SIZE);

// puts pointer to item into this table
// (the item's key must not yet occur in the table)
void put(HashItem* item);

// return from table item with denoted key
// (NULL, if there is no such item)
HashItem* get(HashKey& key);
```

where `HashKey` and `HashItem` are two abstract classes

```
class HashKey {
public:
  virtual ~HashKey() {}

  // the hash value of the key
  virtual unsigned int hashValue() = 0;

  // true if this key equals the denoted key
  virtual bool operator==(HashKey& key) = 0;
};

class HashItem {
public:
  virtual ~HashKey() {}

  // get item's key
  virtual HashKey &getKey() = 0;

  // get hash value of item
  unsigned int hashValue() { return getKey().hashValue(); }

  // decide whether item has denoted key
  bool hasKey(HashKey &key) { return getKey() == key; }
};
```

The hash table is represented by an array of size n where each array entry denotes a linked list of nodes; each node contains an item (a pointer to an `HashItem` object) and the number of occurrences of this item. When a new item is put into the table, the hash value h of its key is determined and the item is inserted into the list at position $h \bmod n$ (if an item with this key already exists in the list, the number of its occurrences is increased by one); likewise, if a key is looked up, its hash value h is determined and an item with this key is searched in the list at position $h \bmod n$.

The item objects are allocated by the caller of `put` for the sole use of the hash table; the hash table may thus delete these objects in its destructor.

Use these classes to reimplement the program `TextStatistics` N of Exercise 2 as follows:

- Derive from `HashKey` a concrete class `WordKey` which is represented by a word (i.e. character string) with a corresponding member function for string hashing.

  Note that in the definition of function `WordKey::operator==` the parameter `key` must be explicitly converted from type `HashKey&` to type `WordKey&`; if this conversion is not possible (i.e. if the actual argument is not of this type), the result must be `false`. The corresponding code is

  ```
  #include <typeinfo>
  bool WordKey::operator==(HashKey& key) {
    if(typeid(key) != typeid(WordKey&)) return false;
    WordKey& k = dynamic_cast<WordKey&>(key);
    return ...; // compare contents of this key and k
  }
  ```

- Derive from `HashItem` a concrete class `WordItem` which holds a key of type `WordKey` (representing the word).

- Derive from `HashTable` a class `Text` as

  ```
  class Text: protected HashTable { ... }
  ```

  which provides (on top of the inherited functionality of the hash table which remains protected) the same public functionality as the class of Exercise 2.

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