Exercise 9.1 — Association Lists

Produce a data type of association lists from strings to integers. Such association lists can be used for example as price lists, associating a price (in cents) with each element of a finite set of product names.

Obviously, each list node then needs to contain a string key and an integer value besides the successor pointer.

These association lists should be ordered by key (similar to CharList), and all keys in a single list need to be different, i.e., no duplicate keys (unlike CharList)!

Provide functions for the following tasks, and carefully document their interfaces:

(a) Counting the number of pairs in a list.
(b) Returning the largest key contained in a list.
(c) Calculating the average of the values contained in a list.
(d) Insertion of a non-NULL key with a new value into an association list — in case that key already was in the list, the new value overrides the old value.
(e) Lookup of a key in an association list, returning the associated value.
(f) Deletion of a key-value-pair, identified by the key.
(g) Printing the list as a sequence of lines to stdout.
(h) Given two such lists, find all keys with which the two lists associate different values (prices), and for each such case print an informative line to stdout.
(i) Given a border value b, splitting a list into two such that one contains all the pairs with values less than b, and the other contains all the other pairs.

Note: This is an implementation of partial functions of type string -> int.

What I call “partial function” you may have encountered under the names “univalent relation” and “deterministic relation” in discrete mathematics.

A more precise name would be “potentially non-total function”, but the name “partial function” is widely used, and frequently supported by special notation; I follow the Z specification notation (ISO/IEC 13568:2002, see also http://vl.fmnet.info/z/ and http://en.wikipedia.org/wiki/Z_notation) in writing “A -> B” for the set of all partial functions from the set A to the set B.