A hash table is a data structure that associates values with identifiers of some kind called "keys." The idea is to store the value in the structure in a way that allows it to be recovered efficiently without knowing its position in the structure, just by supplying the key for that value. Keys are most often Scheme symbols, strings, or integers, but can in principle belong to almost any data type. Values can be of any type.

I have provided an implementation of hash tables in the form of a library, (cl hash-tables). On MathLAN, it is available in the file

/home/reseda/courses/computational-linguistics/code/cl/hash-tables.ss

which you may use and copy. It is adapted from an earlier Scheme implementation by Panu Kalliokoski, published in 2005 as Scheme Request for Implementation 69.

In this implementation, the keys and values are stored as pairs. The data structure is a vector of lists containing these pairs. When a key–value association is stored in the hash table, a procedure called a “hash function” is applied to the key. Its result is an index into the vector. The pair containing the key and the value is prepended to the list of pairs at that position in the vector. To recover the value associated with a given key, the hash function is once more applied to the key to get the same index into the vector. The list at the position in the vector is then searched for a pair whose car matches the key. When such a pair is found, its cdr is returned.

1. The make-hash-table procedure creates an empty hash table. In successive optional arguments, the caller can supply a specialized function for determining whether two keys are to be treated as equivalent, a customized hash function, and the length of the (initial) underlying vector of lists.

Create a hash table named tallies in which the keys are characters, to be compared with the char=? procedure and hashed with the hash procedure, which is also in the (cl hash-tables) library.

2. The hash-table-set! procedure takes a hash table, a key, and a value and associates the key with the value in the table. This procedure is done only for its side effect and does not return any useful value.

Use hash-table-set! to associate the character #\T with the integer 1 in the tallies table.

3. The hash-table-ref procedure takes a hash table and a key and returns the value associated with that key in the table.

Use hash-table-ref to determine the value associated with the character #\T in the tallies table.

4. The hash-table-ref procedure signals an error if you give it a key that is not present in the hash table at all. To avoid this, programmers can check in advance for the presence of a key with the hash-table-exists? predicate, which takes a hash table and a putative key and returns #t if the table associates a value with that key and #f if it does not.

Determine whether #\t is a key in the tallies table.

5. Alternatively, the hash-table-ref/default procedure allows you to supply an extra argument, the value of which will be returned (“by default”) if the key is not found in the table.
Use \texttt{hash-table-ref/default} to determine the value associated with the character \texttt{#\c} in the \texttt{tallies} table, returning 0 if there is no association for that character.

6. The procedure \texttt{hash-table-update!/default} takes four arguments: a hash table, a key, an update function, and a default value. It searches for the key in the hash table. If it finds an association for that key, it applies the update function to the corresponding value and updates the association so that the key is now associated with the value returned by the update function. If it does not find an association for the key, it associates the key instead with the result of applying the update function to the default value.

Use the \texttt{hash-table-update!/default} procedure to create a hash table with the characters that occur in the string "The cat sat on the mat." as keys and the number of occurrences of those characters as the corresponding values. (For example, the character \texttt{#\a} should be associated with the integer value 3 in the hash table, because there are three occurrences of \texttt{#\a} in the given string.)

Check your results by using \texttt{hash-table->alist} to convert your hash table to an association list (a list of the key–value pairs in the table) and examining that list.

7. A data structure that keeps a tally of the number of occurrences of each character in a text is called a character spectrum for that text.

Design, implement, and test a Scheme procedure \texttt{character-spectrum} that takes as its argument an input port (assumed already to have been opened), reads in and tallies characters from that port one by one, and returns a character spectrum of the text it reads, in the form of an association list in which the car of each pair is a character occurring in the text and the cdr is the number of occurrences of that character. Ideally, the keys should appear in ascending order of their Unicode codepoints. Use a hash table so that you can process each character efficiently.

8. Use your \texttt{character-spectrum} procedure to construct a character spectrum for Louisa May Alcott’s novel \textit{Eight Cousins}, the text of which is available on MathLAN (courtesy of Project Gutenberg) in the file

\texttt{/home/reseda/computational-linguistics/resources/eight-cousins.txt}