LISP - HASH TABLE

The hash table data structure represents a collection of **key-and-value** pairs that are organized based on the hash code of the key. It uses the key to access the elements in the collection.

A hash table is used when you need to access elements by using a key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Creating Hash Table in LISP

In Common LISP, hash table is a general-purpose collection. You can use arbitrary objects as a key or indexes.

When you store a value in a hash table, you make a key-value pair, and store it under that key. Later you can retrieve the value from the hash table using the same key. Each key maps to a single value, although you can store a new value in a key.

Hash tables, in LISP, could be categorised into three types, based on the way the keys could be compared - eq, eql or equal. If the hash table is hashed on LISP objects then the keys are compared with eq or eql. If the hash table hash on tree structure, then it would be compared using equal.

The **make-hash-table** function is used for creating a hash table. Syntax for this function is:

```
make-hash-table &key :test :size :rehash-size :rehash-threshold
```

Where:

- The **key** argument provides the key.
- The **:test** argument determines how keys are compared - it should have one of three values #'eq, #'eql, or #'equal, or one of the three symbols eq, eql, or equal. If not specified, eql is assumed.
- The **:size** argument sets the initial size of the hash table. This should be an integer greater than zero.
- The **:rehash-size** argument specifies how much to increase the size of the hash table when it becomes full. This can be an integer greater than zero, which is the number of entries to add, or it can be a floating-point number greater than 1, which is the ratio of the new size to the old size. The default value for this argument is implementation-dependent.
- The **:rehash-threshold** argument specifies how full the hash table can get before it must grow. This can be an integer greater than zero and less than the :rehash-size *inwhichcaseitwillbescaledwheneverthetableisgrown*, or it can be a floating-point number between zero and 1. The default value for this argument is implementation-dependent.

You can also call the make-hash-table function with no arguments.

Retrieving Items from and Adding Items into the Hash Table

The **gethash** function retrieves an item from the hash table by searching for its key. If it does not find the key, then it returns nil.

It has the following syntax:

```
gethash key hash-table &optional default
```

where:

- key: is the associated key

- hash-table: is the hash-table to be searched
- default: is the value to be returned, if the entry is not found, which is nil, if not specified.

The **gethash** function actually returns two values, the second being a predicate value that is true if an entry was found, and false if no entry was found.

For adding an item to the hash table, you can use the **setf** function along with the **gethash** function.

Example

Create a new source code file named main.lisp and type the following code in it.

```
(setq empList (make-hash-table))
(setf (gethash '001 empList) '(Charlie Brown))
(setf (gethash '002 empList) '(Freddie Seal))
(write (gethash '001 empList))
(terpri)
(write (gethash '002 empList))
```

When you execute the code, it returns the following result:

```
(CHARLIE BROWN)
(FREDDIE SEAL)
```

Removing an Entry

The **remhash** function removes any entry for a specific key in hash-table. This is a predicate that is true if there was an entry or false if there was not.

The syntax for this function is:

```
remhash key hash-table
```

Example

Create a new source code file named main.lisp and type the following code in it.

```
(setq empList (make-hash-table))
(setf (gethash '001 empList) '(Charlie Brown))
(setf (gethash '002 empList) '(Freddie Seal))
(setf (gethash '003 empList) '(Mark Mongoose))

(write (gethash '001 empList))
(terpri)
(write (gethash '002 empList))
(terpri)
(write (gethash '003 empList))
(remhash '003 empList)
(terpri)
(write (gethash '003 empList))
```

When you execute the code, it returns the following result:

```
(CHARLIE BROWN)
(FREDDIE SEAL)
(MARK MONGOOSE)
NIL
```

The maphash Function

The **maphash** function allows you to apply a specified function on each key-value pair on a hash table.

It takes two arguments - the function and a hash table and invokes the function once for each key/value pair in the hash table.

Example

Create a new source code file named main.lisp and type the following code in it.

```
(setq empList (make-hash-table))
(setf (gethash '001 empList) '(Charlie Brown))
(setf (gethash '002 empList) '(Freddie Seal))
(setf (gethash '003 empList) '(Mark Mongoose))

(maphash #'(lambda (k v) (format t "~a => ~a~%" k v)) empList)
```

When you execute the code, it returns the following result:

```
3 => (MARK MONGOOSE)
2 => (FREDDIE SEAL)
1 => (CHARLIE BROWN)
```