The Hash Library

0.2.1

Generated by Doxygen 1.7.6.1

Tue Apr 23 2013 22:31:52

Contents

Chapter 1

C Data Structure Library: Hash Library

Version

0.2.1

Author

Jun Woong (woong.jun at gmail.com)

Date

last modified on 2013-04-23

1.1 Introduction

This document specifies the Hash Library which belongs to the C Data Structure Library. The basic structure is from David Hanson's book, "C Interfaces and Implementations." I modified the original implementation to make it more appropriate for my other projects, to speed up operations, to add missing but useulf facilities and to enhance its readibility; for example a prefix is used more strictly in order to avoid the user namespace pollution.

Since the book explains its design and implementation in a very comprehensive way, not to mention the copyright issues, it is nothing but waste to repeat it here, so I finish this document by giving introduction to the library; how to use the facilities is deeply explained in files that define them.

The Hash Library reserves identifiers starting with `hash_` and `HASH_`, and imports the Assertion Library (which requires the Exception Handling Library) and the Memory Management Library.

1.2 How to Use The Library

The Hash Library implements a hash table and is one of the most frequently used libraries; it is essential to get a hash key for datum before putting it into tables by the

Table Library or sets by the Set Library. The storage used to maintain the hash table is managed by the library and no function in the library demands memory allocation done by user code.

The Hash Library provides one global hash table, so that there is no function that creates a table or destroy it. A user can start to use the hash table without its creation just by putting data to it using an appropriate function: hash_string() for C strings, hash_int() for signed integers and hash_new() for other arbitrary forms of data. Of course, since the library internally allocates storage to manage hash keys and values, functions to remove a certain key from the table and to completely clean up the table are offered: hash_free() and hash_reset(). In addition, since strings are very often used to generate hash keys for them, hash_vload() and hash_aload() are provided and useful especially when preloading several strings onto the table.

1.2.1 Some Caveats

A common mistake made when using the Hash Library is to pass data to functions that expect a hash key without making one. For example, table_put() in the Table Library requires its second argument be a hash key but it is likely to careless write this code to put to a table referred to as `mytable` a string key and its relevant value:

```
char *key, *val;
...
table_put(mytable, key, val);
```

This code, however, does not work because the second argument to table_put() should be a hash key not a raw string. Thus, the correct one should be:

```
table_put(mytable, hash_string(key), val);
```

One more thing to note is that hash_string() and similar functions to generate a hash key is an actual function. If a hash key obtained from your data is frequently used in code, it is better for efficiency to have it in a variable rather than to call hash_string() several times.

If your compiler rejects to compile the library with a diagnostic including "scatter[] assumes UCHAR_MAX $<$ 256!" which says `CHAR_BIT` (the number of bits in a byte) in your environment is larger than 8, you have to add elements to the array `scatter` to make its total number match the number of characters in your implementation. `scatter` is used to map a character to a random number. For how to generate the array, see the explanation given for the array in code.

1.3 Boilerplate Code

No boilerplate code is provided for this library.

1.4 Future Directions

No future change on this library planned yet.

1.5 Contact Me

Visit http://code.woong.org to get the lastest version of this library. Only a small portion of my homepage (http://www.woong.org) is maintained in English, thus one who is not good at Korean would have difficulty when navigating most of other pages served in Korean. If you think the information you are looking for is on pages written in Korean, do not hesitate to send me an email to ask for help.

Any comments about the library are welcomed. If you have a proposal or question on the library just email me, and I will reply as soon as possible.

1.6 Copyright

IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY - DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 hash.c File Reference

Source for Hash Library (CDSL)

`#include <stddef.h> #include <stdio.h> #include <string.h> #include <limits.h> #include <stdarg.h> #include "cbl/assert.h" #include "cbl/memory.h" #include "hash.h"` Include dependency graph for hash.c:

Functions

- const char *() hash_string (const char *str)

  returns a hash string for a string.
- const char *() hash_int (long n)

  returns a hash string for a signed integer.
- const char *() hash_new (const char *byte, size_t len)

  returns a hash string for a byte sequence.
- size_t() hash_length (const char *byte)

  returns the length of a hash string.
- void() hash_free (const char *byte)

  deallocates storage for a hash string.
- void() hash_reset (void)

  resets the hash table by deallocating all hash strings in it.
- void hash_vload (const char *str,...)

  puts a sequence of strings to the hash table.
- void hash_aload (const char *strs[])

  puts given strings to the hash table.

3.1.1 Detailed Description

Source for Hash Library (CDSL)

3.1.2 Function Documentation

3.1.2.1 void hash_aload (const char * *strs[]*)

puts given strings to the hash table.

hash_aload() takes strings from an array of strings (precisely, an array of pointers to `char`) and puts them into the hash table. Since the function does not take the size of the string array there should be a way to mark end of the array, which a null pointer is for. hash_aload() is useful when a program needs to preload some strings to the hash table for later use. A variadic version is also provided; see hash_vload().

Possible exceptions: mem_exceptfail

Unchecked errors: none

Parameters

`in`	*strs*	array of null-terminated strings

Returns

nothing

Here is the call graph for this function:

3.1.2.2 void() hash_free (const char * *byte*)

deallocates storage for a hash string.

hash_free() deallocates storage for a hash string, which effectively eliminates a hash string from the hash table. This facility is not used so frequently by user code that the original implementation did not provide it.

Possible exceptions: assert_exceptfail

Unchecked errors: hash string modified by user given for `byte`

Parameters

`in`	*byte*	hash string to remove

Returns

nothing

3.1.2.3 const char*() hash_int (long *n*)

returns a hash string for a signed integer.

hash_int() returns a hash string for a given, possibly signed, integer whose type is `long`.

Possible exceptions: mem_exceptfail

Unchecked errors: none

Parameters

in	*n*	integer for which hash string returned

Returns

hash string for integer

Here is the call graph for this function:

3.1.2.4 size_t() hash_length (const char * *byte*)

returns the length of a hash string.

Given a hash string, hash_length() returns its length.

Possible exceptions: assert_exceptfail

Unchecked errors: hash string modified by user given for `byte`, foreign string given for `byte` (only for faster version)

Parameters

in	*byte*	byte sequence whose length returned

Returns

length of byte sequence

3.1.2.5 const char*() hash_new (const char * *byte*, size_t *len*)

returns a hash string for a byte sequence.

hash_new() returns a hash string for a given byte sequence, which may not end with a null character or may have a null character embedded in it. Even if it has "new" in its name, hash_new() just returns the existing hash string if there is already one created for the same byte sequence, which means there is only one instance of each byte sequence in the hash table; that is what hashing is for.

An empty byte sequence which contains nothing and whose length is 0 is also valid. But remember that `byte` has a valid pointer value by pointing to a valid object even when `len` is zero, since C allows no zero-sized object.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: hash string modified by user given for `byte`

Warning

It leads to an unpredictable result to modify a hash string returned by the library.

Parameters

in	*byte*	byte sequence for which hash string returned
in	*len*	length of byte sequence

Returns

hash string for byte sequence

Here is the caller graph for this function:

3.1.2.6 void() hash_reset (void)

resets the hash table by deallocating all hash strings in it.

hash_reset() deallocates all hash strings in the hash table and thus resets it.

Possible exceptions: none

Unchecked errors: none

Returns

nothing

3.1.2.7 const char*() hash_string (const char * *str*)

returns a hash string for a string.

hash_string() returns a hash string for a given null-terminated string. It is equivalent to call hash_new() with the string and its length counted by strlen(). Note that the trailing null character is not counted and it is appended when storing into the hash table. An empty string which is consisted only of a null character is also a valid argument for hash_string(); see hash_new() for details.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters

in	*str*	null-terminated string for which hash string returned

Returns

hash string for string

Here is the call graph for this function:

Here is the caller graph for this function:

3.1.2.8 void hash_vload (const char * *str*, ...)

puts a sequence of strings to the hash table.

hash_vload() takes a possibly empty sequence of null-terminated strings and puts them into the hash table. There should be a way to mark the end of the argument list, which a null pointer is for. hash_vload() is useful when a program needs to preload some strings to the hash table for later use. An array-version of hash_vload() is also provided; see hash_aload().

Possible exceptions: mem_exceptfail

Unchecked errors: none

Parameters

in	*str*	null-terminated string to put to hash table
in	...	other such strings to put to hash table

Returns

nothing

Here is the call graph for this function:

3.2 hash.h File Reference

Documentation for Hash Library (CDSL)

`#include <stddef.h>` Include dependency graph for hash.h: This graph shows which files directly or indirectly include this file:

Functions

hash string creating functions:

- const char * hash_string (const char *)
  returns a hash string for a string.
- const char * hash_int (long)
  returns a hash string for a signed integer.
- const char * hash_new (const char *, size_t)
  returns a hash string for a byte sequence.

- void hash_vload (const char *,...)
  puts a sequence of strings to the hash table.
- void hash_aload (const char *[])
  puts given strings to the hash table.

hash destroying functions:

- void hash_free (const char *)
  deallocates storage for a hash string.
- void hash_reset (void)
  resets the hash table by deallocating all hash strings in it.

misc. functions:

- size_t hash_length (const char *)
  returns the length of a hash string.

3.2.1 Detailed Description

Documentation for Hash Library (CDSL) Header for Hash Library (CDSL)

3.2.2 Function Documentation

3.2.2.1 void hash_aload (const char * *strs[]*)

puts given strings to the hash table.

hash_aload() takes strings from an array of strings (precisely, an array of pointers to `char`) and puts them into the hash table. Since the function does not take the size of the string array there should be a way to mark end of the array, which a null pointer is for. hash_aload() is useful when a program needs to preload some strings to the hash table for later use. A variadic version is also provided; see hash_vload().

Possible exceptions: mem_exceptfail

Unchecked errors: none

Parameters

`in`	*strs*	array of null-terminated strings

Returns

nothing

Here is the call graph for this function:

3.2.2.2 void hash_free (const char * *byte*)

deallocates storage for a hash string.

hash_free() deallocates storage for a hash string, which effectively eliminates a hash string from the hash table. This facility is not used so frequently by user code that the original implementation did not provide it.

Possible exceptions: assert_exceptfail

Unchecked errors: hash string modified by user given for `byte`

Parameters

`in`	*byte*	hash string to remove

Returns

nothing

3.2.2.3 const char* hash_int (long *n*)

returns a hash string for a signed integer.

hash_int() returns a hash string for a given, possibly signed, integer whose type is `long`.

Possible exceptions: mem_exceptfail

Unchecked errors: none

Parameters

`in`	*n*	integer for which hash string returned

Returns

hash string for integer

Here is the call graph for this function:

3.2.2.4 size_t hash_length (const char * *byte*)

returns the length of a hash string.

Given a hash string, hash_length() returns its length.

Possible exceptions: assert_exceptfail

Unchecked errors: hash string modified by user given for `byte`, foreign string given for `byte` (only for faster version)

Parameters

in	*byte*	byte sequence whose length returned

Returns

length of byte sequence

3.2.2.5 const char* hash_new (const char * *byte*, size_t *len*)

returns a hash string for a byte sequence.

hash_new() returns a hash string for a given byte sequence, which may not end with a null character or may have a null character embedded in it. Even if it has "new" in its name, hash_new() just returns the existing hash string if there is already one created for the same byte sequence, which means there is only one instance of each byte sequence in the hash table; that is what hashing is for.

An empty byte sequence which contains nothing and whose length is 0 is also valid. But remember that `byte` has a valid pointer value by pointing to a valid object even when `len` is zero, since C allows no zero-sized object.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: hash string modified by user given for `byte`

Warning

It leads to an unpredictable result to modify a hash string returned by the library.

Parameters

in	*byte*	byte sequence for which hash string returned
in	*len*	length of byte sequence

Returns

hash string for byte sequence

Here is the caller graph for this function:

3.2.2.6 void hash_reset (void)

resets the hash table by deallocating all hash strings in it.

hash_reset() deallocates all hash strings in the hash table and thus resets it.

Possible exceptions: none

Unchecked errors: none

Returns

nothing

3.2.2.7 const char* hash_string (const char * *str*)

returns a hash string for a string.

hash_string() returns a hash string for a given null-terminated string. It is equivalent to call hash_new() with the string and its length counted by strlen(). Note that the trailing null character is not counted and it is appended when storing into the hash table. An empty string which is consisted only of a null character is also a valid argument for hash_string(); see hash_new() for details.

Possible exceptions: assert_exceptfail, mem_exceptfail

Unchecked errors: none

Parameters

in	*str*	null-terminated string for which hash string returned

Returns

hash string for string

Here is the call graph for this function:

Here is the caller graph for this function:

3.2.2.8 void hash_vload (const char * *str*, ...)

puts a sequence of strings to the hash table.

hash_vload() takes a possibly empty sequence of null-terminated strings and puts them into the hash table. There should be a way to mark the end of the argument list, which a null pointer is for. hash_vload() is useful when a program needs to preload some strings to the hash table for later use. An array-version of hash_vload() is also provided; see hash_aload().

Possible exceptions: mem_exceptfail

Unchecked errors: none

Parameters

in	*str*	null-terminated string to put to hash table
in	...	other such strings to put to hash table

Returns

nothing

Here is the call graph for this function: