Chain Hash API For C Code

Generated by Doxygen 1.8.12

Contents

1 Chained-Hashtable **1**

2 Class Index **3**

2.1 Class List 3

3 File Index **5**

3.1 File List 5

4 Class Documentation **7**

4.1 _CHash_ Struct Reference 7

4.1.1 Detailed Description 7

5 File Documentation **9**

5.1 include/chain-hash.h File Reference 9

5.1.1 Detailed Description 10

5.1.2 Typedef Documentation 10

5.1.2.1 CHash 10

5.1.3 Function Documentation 10

5.1.3.1 chash_destroy() 10

5.1.3.2 chash_init() 11

5.1.3.3 chash_insert() 11

5.1.3.4 chash_lookup() 11

5.1.3.5 chash_remove() 12

5.1.3.6 chash_traverse() 12

Index **13**

Chapter 1

Chained-Hashtable

This repository contains an implementation of a *Chained Hash Table* in C. A hash table is a data structure that maps data to elements in an indexable, single dimensional array, by performing a transformation to the data in order to obtain a *key*. Some hash tables, like in the case of the chained hash table, provide a mechanism for handling *collisions*, which are edge cases that result from two or more datum being mapped to the same key. In a chained hash table, elements are placed into buckets at each index of the array, which are containers that allow for multiple datum to be placed. In this case, these containers are implemented as a singly linked list.

The goal for this repository is to not only contain the code for implementing a chained hash table, but also a few common hash functions to use with the API. These are not currently included, but will be in a future commit.

Building: To build this repository on any system, one follows the traditional Autotools procedure.

```
./configure

make

make install
```

This repository allows for two different build configurations. The default builds the API assuming statically allocated structures, as in they cannot be resized post-allocation. The repository also supports dynamically-allocated structures, meaning that a size is not specified upon calling chash_init(), and buckets are allocated as needed. This configuration will result in slightly slower operation of the structure, but is useful when the API is employed in environments where efficiency is not key.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CHash
The CHash struct definition 7

Chapter 3

File Index

3.1 File List

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

include/chain-hash.h
Details public interface of the CHash API 9

Chapter 4

Class Documentation

4.1 _CHash_ Struct Reference

The CHash struct definition.

`#include <chain-hash.h>`

Public Attributes

- unsigned int **size**
- unsigned int **buckets**
- int(∗ **hash**)(const void ∗)
- int(∗ **match**)(const void ∗, const void ∗)
- void(∗ **destroy**)(void ∗)
- List ∗∗ **table**

4.1.1 Detailed Description

The CHash struct definition.

The CHash structure is the main type provided in this API. This structure represents the Chained Hash table and all data held within it.

Warning

The user should interface directly with the struct elements as sparingly as possible.

The documentation for this struct was generated from the following file:

- include/chain-hash.h

Chapter 5

File Documentation

5.1 include/chain-hash.h File Reference

Details public interface of the CHash API.

```
#include "linkedlist.h"
```

Classes

- struct _CHash_

  The CHash struct definition.

Macros

- #define chash_size(Table) ((Table)->size)

  Returns the size of the Hash.
- #define chash_isempty(Table) ((Table)->size == 0 ? 1 : 0)

  Returns true if the hash is empty, false if it is not.

Typedefs

- typedef struct _CHash_ CHash

  The CHash struct definition.

Functions

- CHash * chash_init (int size, int(*hash)(const void *), int(*match)(const void *, const void *), void(*destroy)(void *))

  The initialization funtion.
- void chash_destroy (CHash *table)

  Function to free all data associated with the hash.
- int chash_insert (CHash *table, const void *data)

  Inserts the value specified by `data` into the hash.
- int chash_remove (CHash *table, void **data)

  Removes the element specified by data from the table.
- void chash_traverse (CHash *table, void(*callback)(void *))

  Invokes `callback` on every element in the hash.
- int chash_lookup (CHash *table, void **data)

  Queries the hash for a specific data point.

5.1.1 Detailed Description

Details public interface of the CHash API.

Author

Ethan D. Twardy

Date

07/15/2017 This file contains everything a user needs to know to use the CHash API. Functions and their usage, macros, types included, etc.

5.1.2 Typedef Documentation

5.1.2.1 CHash

```
typedef struct _CHash_ CHash
```

The CHash struct definition.

The CHash structure is the main type provided in this API. This structure represents the Chained Hash table and all data held within it.

Warning

The user should interface directly with the struct elements as sparingly as possible.

5.1.3 Function Documentation

5.1.3.1 chash_destroy()

```
void chash_destroy (
            CHash * table )
```

Function to free all data associated with the hash.

Parameters

table	The table to destroy.

Returns

void

Warning

If ->destroy is set to NULL, the data will not be freed, and it is the responsibility of the programmer to manage this mem.

5.1.3.2 chash_init()

```
CHash* chash_init (
            int size,
            int(*)(const void *) hash,
            int(*)(const void *, const void *) match,
            void(*)(void *) destroy )
```

The initialization funtion.

Parameters

size	The number of containers to create in the hash.
hash	The user-defined hash function.
match	The user-defined function for comparing two data points.
destroy	The user-defined function for freeing data points.

Returns

CHash* Pointer to a created and initialized CHash struct.

5.1.3.3 chash_insert()

```
int chash_insert (
            CHash * table,
            const void * data )
```

Inserts the value specified by `data` into the hash.

Parameters

table	The table to insert the value into.
data	The data to insert into the table.

Returns

int `0` on success, `1` if the data already exists within the table, and `-1` if there was an error.

5.1.3.4 chash_lookup()

```
int chash_lookup (
            CHash * table,
            void ** data )
```

Queries the hash for a specific data point.

Parameters

table	The table to search
data	The data to search for

Returns

int `1` if the data was found, `0` if it was not. If the data was found, data now contains the address of that data.

5.1.3.5 chash_remove()

```
int chash_remove (
            CHash * table,
            void ** data )
```

Removes the element specified by data from the table.

Parameters

table	The table to operate on
data	Pointer to the data to remove. If set to `NULL`, removes the first element in the hash.

Returns

int `0` on success, `-1` if there was an error.

5.1.3.6 chash_traverse()

```
void chash_traverse (
            CHash * table,
            void(*)(void *) callback )
```

Invokes `callback` on every element in the hash.

Parameters

table	The hash table to traverse
callback	The callback function to invoke on every element.

Returns

void

Index

CHash, 7

CHash
    chain-hash.h, 10
chain-hash.h
    CHash, 10
    chash_destroy, 10
    chash_init, 10
    chash_insert, 11
    chash_lookup, 11
    chash_remove, 12
    chash_traverse, 12
chash_destroy
    chain-hash.h, 10
chash_init
    chain-hash.h, 10
chash_insert
    chain-hash.h, 11
chash_lookup
    chain-hash.h, 11
chash_remove
    chain-hash.h, 12
chash_traverse
    chain-hash.h, 12

include/chain-hash.h, 9