Hash Table

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 HTable Struct Reference

```
#include <HashTableAPI.h>
```

Data Fields

- size_t size

  number that represents the size of the hash table
- Node ∗∗ table

  array that contains all of the table nodes
- void(∗ destroyData)(void ∗data)

  function pointer to a function to delete a single piece of data from the hash table
- int(∗ hashFunction)(size_t tableSize, int key)

  function pointer to a function to hash the data
- void(∗ printNode)(void ∗toBePrinted)

  function pointer to a function that prints out a data element of the table

3.1.1 Detailed Description

Hash table structure

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

- include/HashTableAPI.h

3.2 Node Struct Reference

```
#include <HashTableAPI.h>
```

Data Fields

- int key

    integer that represents a piece of data in the table (eg 35->"hello")
- void ∗ data

    pointer to generic data that is to be stored in the hash table
- struct Node ∗ next

    pointer to the next Node if a collision is detected

3.2.1 Detailed Description

Node of the hash table.

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

- include/HashTableAPI.h

Chapter 4

File Documentation

4.1 include/HashTableAPI.h File Reference

File containing the function definitions of a hash table.

```
#include <stdio.h>
#include <stdlib.h>
```

Data Structures

- struct Node
- struct HTable

Typedefs

- typedef struct Node Node
- typedef struct HTable HTable

Functions

- HTable ∗ createTable (size_t size, int(∗hashFunction)(size_t tableSize, int key), void(∗destroyData)(void ∗data), void(∗printNode)(void ∗toBePrinted))
- Node ∗ createNode (int key, void ∗data)
- void destroyTable (HTable ∗hashTable)
- void insertData (HTable ∗hashTable, int key, void ∗data)
- void removeData (HTable ∗hashTable, int key)
- void ∗ lookupData (HTable ∗hashTable, int key)
- int **hashNode** (size_t tableSize, int key)
- void **destroyNodeData** (void ∗data)
- void **printNodeData** (void ∗toBePrinted)

4.1.1 Detailed Description

File containing the function definitions of a hash table.

Author

Michael Ellis

Date

February 2017

4.1.2 Typedef Documentation

4.1.2.1 HTable

```
typedef struct HTable HTable
```

Hash table structure

4.1.2.2 Node

```
typedef struct Node Node
```

Node of the hash table.

4.1.3 Function Documentation

4.1.3.1 createNode()

```
Node* createNode (
            int key,
            void * data )
```

Function for creating a node for the hash table.

Precondition

Node must be cast to void pointer before being added.

Postcondition

Node is valid and able to be added to the hash table

Parameters

key	integer that represents the data (eg 35->"hello")
data	is a generic pointer to any data type.

Returns

returns a node for the hash table

4.1.3.2 createTable()

```
HTable* createTable (
            size_t size,
            int(*)(size_t tableSize, int key) hashFunction,
            void(*)(void *data) destroyData,
            void(*)(void *toBePrinted) printNode )
```

Function to point the hash table to the appropriate functions. Allocates memory to the struct and table based on the size given.

Returns

pointer to the hash table

Parameters

size	size of the hash table
hashFunction	function pointer to a function to hash the data
destroyData	function pointer to a function to delete a single piece of data from the hash table
printNode	function pointer to a function that prints out a data element of the table

4.1.3.3 destroyTable()

```
void destroyTable (
            HTable * hashTable )
```

Deletes the entire hash table and frees memory of every element.

Precondition

Hash Table must exist.

Parameters

hashTable	pointer to hash table containing elements of data

4.1.3.4 insertData()

```
void insertData (
            HTable * hashTable,
            int key,
            void * data )
```

Inserts a Node in the hash table.

Precondition

hashTable type must exist and have data allocated to it

Parameters

hashTable	pointer to the hash table
key	integer that represents the data (eg 35->"hello")
data	pointer to generic data that is to be inserted into the list

4.1.3.5 lookupData()

```
void* lookupData (
            HTable * hashTable,
            int key )
```

Function to return the data from the key given.

Precondition

The hash table exists and has memory allocated to it

Parameters

hashTable	pointer to the hash table containing data nodes
key	integer that represents a piece of data in the table (eg 35->"hello")

Returns

returns a pointer to the data in the hash table. Returns NULL if no match is found.

4.1.3.6 removeData()

```
void removeData (
            HTable * hashTable,
            int key )
```

Function to remove a node from the hash table

Precondition

Hash table must exist and have memory allocated to it

Postcondition

Node at key will be removed from the hash table if it exists.

Parameters

hashTable	pointer to the hash table struct
key	integer that represents a piece of data in the table (eg 35->"hello")