0x1A. C - Hash tables

CAlgorithmData structure

- By: Julien Barbier
- Weight: 1
- Project over - took place from Jun 30, 2022 6:00 AM to Jul 2, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 46.0/47 mandatory & 21.0/21 optional
- **Altogether: 195.74%**
  - Mandatory: 97.87%
  - Optional: 100.0%
  - Calculation: 97.87% + (97.87% * 100.0%) == **195.74%**

Resources

Read or watch:

- What is a HashTable Data Structure - Introduction to Hash Tables , Part 0
- Hash function
- Hash table

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone, **without the help of Google**:

General

- What is a hash function
- What makes a good hash function
- What is a hash table, how do they work and how to use them
- What is a collision and what are the main ways of dealing with collisions in the context of a hash table
- What are the advantages and drawbacks of using hash tables
- What are the most common use cases of hash tables

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using gcc, using the options -Wall -Werror -Wextra -pedantic -std=gnu89
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl and betty-doc.pl
- You are not allowed to use global variables
- No more than 5 functions per file
- You are allowed to use the C standard library
- The prototypes of all your functions should be included in your header file called `hash_tables.h`
- Don't forget to push your header file
- All your header files should be include guarded

More Info

Data Structures

Please use these data structures for this project:

```
/**
 * struct hash_node_s - Node of a hash table
 *
 * @key: The key, string
 * The key is unique in the HashTable
 * @value: The value corresponding to a key
 * @next: A pointer to the next node of the List
```

```
 */
typedef struct hash_node_s
{
     char *key;
     char *value;
     struct hash_node_s *next;
} hash_node_t;


/**
 * struct hash_table_s - Hash table data structure
 *
 * @size: The size of the array
 * @array: An array of size @size
 * Each cell of this array is a pointer to the first node of a linked list,
 * because we want our HashTable to use a Chaining collision handling
 */
typedef struct hash_table_s
{
     unsigned long int size;
     hash_node_t **array;
} hash_table_t;
```

Tests

We strongly encourage you to work all together on a set of tests

Python Dictionaries

Python dictionaries are implemented using hash tables. When you will be done with this project, you will be able to better understand the power and simplicity of Python dictionaries. So much is actually happening when you type `d = {'a': 1, 'b': 2}`, but everything looks so simple for the user. Python doesn't use the exact same implementation than the one you will work on today though. If you are curious on how it works under the hood, here is a good blog post about how dictionaries are implemented in Python 2.7 (not mandatory).

Note that all dictionaries are not implemented using hash tables and there is a difference between a dictionary and a hash table. Read more here (not mandatory).

Tasks

0. >>> ht = {}
mandatory

Score: 80.0% (*Checks completed: 80.0%*)

Write a function that creates a hash table.

- Prototype: `hash_table_t *hash_table_create(unsigned long int size);`
  - where `size` is the size of the array
- Returns a pointer to the newly created hash table
- If something went wrong, your function should return `NULL`

```
julien@ubuntu:~/0x1A. Hash tables$ cat 0-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"

/**
 * main - check the code for
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    hash_table_t *ht;

    ht = hash_table_create(1024);
    printf("%p\n", (void *)ht);
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 0-main.c 0-hash_table_create.c -o a
julien@ubuntu:~/0x1A. Hash tables$ ./a
0x238a010
```

```
julien@ubuntu:~/0x1A. Hash tables$ valgrind ./a
==7602== Memcheck, a memory error detector
==7602== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==7602== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==7602== Command: ./a
==7602==
0x51fc040
==7602==
==7602== HEAP SUMMARY:
==7602==     in use at exit: 8,208 bytes in 2 blocks
==7602==   total heap usage: 2 allocs, 0 frees, 8,208 bytes allocated
==7602==
==7602== LEAK SUMMARY:
==7602==    definitely lost: 16 bytes in 1 blocks
==7602==    indirectly lost: 8,192 bytes in 1 blocks
==7602==      possibly lost: 0 bytes in 0 blocks
==7602==    still reachable: 0 bytes in 0 blocks
==7602==         suppressed: 0 bytes in 0 blocks
==7602== Rerun with --leak-check=full to see details of leaked memory
==7602==
==7602== For counts of detected and suppressed errors, rerun with: -v
==7602== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x1A. Hash tables$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `0-hash_table_create.c`

Done? Help Check your code Ask for a new correction Get a sandbox QA Review

1. djb2

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a hash function implementing the djb2 algorithm.

- Prototype: `unsigned long int hash_djb2(const unsigned char *str);`
- You are allowed to copy and paste the function from this page

```
julien@ubuntu:~/0x1A. Hash tables$ cat 1-djb2.c
unsigned long int hash_djb2(const unsigned char *str)
{
    unsigned long int hash;
    int c;

    hash = 5381;
    while ((c = *str++))
    {
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
    }
    return (hash);
}
julien@ubuntu:~/0x1A. Hash tables$
julien@ubuntu:~/0x1A. Hash tables$ cat 1-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"

/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    char *s;

    s = "cisfun";
    printf("%lu\n", hash_djb2((unsigned char *)s));
```

```
    s = "Don't forget to tweet today";
    printf("%lu\n", hash_djb2((unsigned char *)s));
    s = "98";
    printf("%lu\n", hash_djb2((unsigned char *)s));
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 1-main.c 1-djb2.c -o b
julien@ubuntu:~/0x1A. Hash tables$ ./b
6953392314605
3749890792216096085
5861846
julien@ubuntu:~/0x1A. Hash tables$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `1-djb2.c`

Done! Help Check your code Get a sandbox QA Review

2. key -> index

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that gives you the index of a key.

- Prototype: `unsigned long int key_index(const unsigned char *key, unsigned long int size);`
    - where `key` is the key
    - and `size` is the size of the array of the hash table
- This function should use the `hash_djb2` function that you wrote earlier
- Returns the index at which the key/value pair should be stored in the array of the hash table
- You will have to use this hash function for all the next tasks

```
julien@ubuntu:~/0x1A. Hash tables$ cat 2-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

```
#include "hash_tables.h"


/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    char *s;
    unsigned long int hash_table_array_size;


    hash_table_array_size = 1024;
    s = "cisfun";
    printf("%lu\n", hash_djb2((unsigned char *)s));
    printf("%lu\n", key_index((unsigned char *)s, hash_table_array_size));
    s = "Don't forget to tweet today";
    printf("%lu\n", hash_djb2((unsigned char *)s));
    printf("%lu\n", key_index((unsigned char *)s, hash_table_array_size));
    s = "98";
    printf("%lu\n", hash_djb2((unsigned char *)s));
    printf("%lu\n", key_index((unsigned char *)s, hash_table_array_size));
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 2-m
ain.c 1-djb2.c 2-key_index.c -o c
julien@ubuntu:~/0x1A. Hash tables$ ./c
6953392314605
237
3749890792216096085
341
5861846
470
```

```
julien@ubuntu:~/0x1A. Hash tables$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `2-key_index.c`

Done! Help Check your code Get a sandbox QA Review

3. >>> ht['betty'] = 'cool'

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that adds an element to the hash table.

- Prototype: `int hash_table_set(hash_table_t *ht, const char *key, const char *value);`
  - Where `ht` is the hash table you want to add or update the key/value to
  - `key` is the key. `key` can not be an empty string
  - and `value` is the value associated with the key. `value` must be duplicated. `value` can be an empty string
- Returns: `1` if it succeeded, `0` otherwise
- In case of collision, add the new node at the beginning of the list

```
julien@ubuntu:~/0x1A. Hash tables$ cat 3-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"

/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    hash_table_t *ht;

    ht = hash_table_create(1024);
```

```
    hash_table_set(ht, "betty", "cool");
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 3-m
ain.c 0-hash_table_create.c 1-djb2.c 2-key_index.c 3-hash_table_set.c -o d
julien@ubuntu:~/0x1A. Hash tables$
```

If you want to test for collisions, here are some strings that collide using the djb2 algorithm:

- **hetairas** collides with **mentioner**
- **heliotropes** collides with **neurospora**
- **depravement** collides with **serafins**
- **stylist** collides with **subgenera**
- **joyful** collides with **synaphea**
- **redescribed** collides with **urites**
- **dram** collides with **vivency**

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `3-hash_table_set.c`

Done! Help Check your code Get a sandbox QA Review

4. >>> ht['betty']

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that retrieves a value associated with a key.

- Prototype: `char *hash_table_get(const hash_table_t *ht, const char *key);`
    - where `ht` is the hash table you want to look into
    - and `key` is the key you are looking for
- Returns the value associated with the element, or `NULL` if `key` couldn't be found

```
julien@ubuntu:~/0x1A. Hash tables$ cat 4-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"
```

```
/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    hash_table_t *ht;
    char *value;

    ht = hash_table_create(1024);
    hash_table_set(ht, "c", "fun");
    hash_table_set(ht, "python", "awesome");
    hash_table_set(ht, "Bob", "and Kris love asm");
    hash_table_set(ht, "N", "queens");
    hash_table_set(ht, "Asterix", "Obelix");
    hash_table_set(ht, "Betty", "Cool");
    hash_table_set(ht, "98", "Battery Street");
    hash_table_set(ht, "c", "isfun");

    value = hash_table_get(ht, "python");
    printf("%s:%s\n", "python", value);
    value = hash_table_get(ht, "Bob");
    printf("%s:%s\n", "Bob", value);
    value = hash_table_get(ht, "N");
    printf("%s:%s\n", "N", value);
    value = hash_table_get(ht, "Asterix");
    printf("%s:%s\n", "Asterix", value);
    value = hash_table_get(ht, "Betty");
    printf("%s:%s\n", "Betty", value);
    value = hash_table_get(ht, "98");
    printf("%s:%s\n", "98", value);
    value = hash_table_get(ht, "c");
```

```
    printf("%s:%s\n", "c", value);
    value = hash_table_get(ht, "javascript");
    printf("%s:%s\n", "javascript", value);
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 4-main.c 0-hash_table_create.c 1-djb2.c 2-key_index.c 3-hash_table_set.c 4-hash_table_get.c -o e
julien@ubuntu:~/0x1A. Hash tables$ ./e
python:awesome
Bob:and Kris love asm
N:queens
Asterix:Obelix
Betty:Cool
98:Battery Street
c:isfun
javascript:(null)
julien@ubuntu:~/0x1A. Hash tables$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `4-hash_table_get.c`

Done! Help Check your code Get a sandbox QA Review

5. >>> print(ht)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that prints a hash table.

- Prototype: `void hash_table_print(const hash_table_t *ht);`
  - where `ht` is the hash table
- You should print the key/value in the order that they appear in the array of hash table
  - Order: array, list
- Format: see example
- If `ht` is NULL, don’t print anything

```
julien@ubuntu:~/0x1A. Hash tables$ cat 5-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"

/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    hash_table_t *ht;

    ht = hash_table_create(1024);
    hash_table_print(ht);
    hash_table_set(ht, "c", "fun");
    hash_table_set(ht, "python", "awesome");
    hash_table_set(ht, "Bob", "and Kris love asm");
    hash_table_set(ht, "N", "queens");
    hash_table_set(ht, "Asterix", "Obelix");
    hash_table_set(ht, "Betty", "Cool");
    hash_table_set(ht, "98", "Battery Street");
    hash_table_print(ht);
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 5-main.c 0-hash_table_create.c 1-djb2.c 2-key_index.c 3-hash_table_set.c 4-hash_table_get.c 5-hash_table_print.c -o f
julien@ubuntu:~/0x1A. Hash tables$ ./f
{}
{'Betty': 'Cool', 'python': 'awesome', 'Bob': 'and Kris love asm', '98': 'Battery Street', 'N': 'queens', 'c': 'fun', 'Asterix': 'Obelix'}
```

```
julien@ubuntu:~/0x1A. Hash tables$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `5-hash_table_print.c`

Done! Help Check your code Get a sandbox QA Review

6. >>> del ht

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that deletes a hash table.

- Prototype: `void hash_table_delete(hash_table_t *ht);`
    - where `ht` is the hash table

```
julien@ubuntu:~/0x1A. Hash tables$ cat 6-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"

/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    hash_table_t *ht;
    char *key;
    char *value;

    ht = hash_table_create(1024);
    hash_table_set(ht, "c", "fun");
```

```
    hash_table_set(ht, "python", "awesome");
    hash_table_set(ht, "Bob", "and Kris love asm");
    hash_table_set(ht, "N", "queens");
    hash_table_set(ht, "Asterix", "Obelix");
    hash_table_set(ht, "Betty", "Cool");
    hash_table_set(ht, "98", "Battery Streetz");
    key = strdup("Tim");
    value = strdup("Britton");
    hash_table_set(ht, key, value);
    key[0] = '\0';
    value[0] = '\0';
    free(key);
    free(value);
    hash_table_set(ht, "98", "Battery Street");
    hash_table_set(ht, "hetairas", "Bob");
    hash_table_set(ht, "hetairas", "Bob Z");
    hash_table_set(ht, "mentioner", "Bob");
    hash_table_set(ht, "hetairas", "Bob Z Chu");
    hash_table_print(ht);
    hash_table_delete(ht);
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra 6-main.c 0-has
h_table_create.c 1-djb2.c 2-key_index.c 3-hash_table_set.c 4-hash_table_get.c 5-hash_
table_print.c 6-hash_table_delete.c -o g
julien@ubuntu:~/0x1A. Hash tables$ valgrind ./g
==6621== Memcheck, a memory error detector
==6621== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==6621== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==6621== Command: ./g
==6621==
{'Betty': 'Cool', 'mentioner': 'Bob', 'hetairas': 'Bob Z Chu', 'python': 'awesome', '
Bob': 'and Kris love asm', '98': 'Battery Street', 'N': 'queens', 'c': 'fun', 'Tim':
'Britton', 'Asterix': 'Obelix'}
==6621==
```

```
==6621== HEAP SUMMARY:
==6621==     in use at exit: 0 bytes in 0 blocks
==6621==   total heap usage: 37 allocs, 37 frees, 8,646 bytes allocated
==6621==
==6621== All heap blocks were freed -- no leaks are possible
==6621==
==6621== For counts of detected and suppressed errors, rerun with: -v
==6621== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x1A. Hash tables$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `6-hash_table_delete.c`

Done! Help Check your code Get a sandbox QA Review

7. $ht['Betty'] = 'Cool'

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

In PHP, hash tables are **ordered**. Wait… WAT? How is this even possible?

Before you continue, please take a moment to think about it: how you would implement it if you were asked to during an interview or a job. What data structures would you use? How would it work?

For this task, please:

- Read PHP Internals Book: HashTable
- Use the same hash function
- Use these data structures:

```
/**
 * struct shash_node_s - Node of a sorted hash table
 *
 * @key: The key, string
 * The key is unique in the HashTable
 * @value: The value corresponding to a key
```

```
 * @next: A pointer to the next node of the List
 * @sprev: A pointer to the previous element of the sorted linked list
 * @snext: A pointer to the next element of the sorted linked list
 */
typedef struct shash_node_s
{
     char *key;
     char *value;
     struct shash_node_s *next;
     struct shash_node_s *sprev;
     struct shash_node_s *snext;
} shash_node_t;


/**
 * struct shash_table_s - Sorted hash table data structure
 *
 * @size: The size of the array
 * @array: An array of size @size
 * Each cell of this array is a pointer to the first node of a linked list,
 * because we want our HashTable to use a Chaining collision handling
 * @shead: A pointer to the first element of the sorted linked list
 * @stail: A pointer to the last element of the sorted linked list
 */
typedef struct shash_table_s
{
     unsigned long int size;
     shash_node_t **array;
     shash_node_t *shead;
     shash_node_t *stail;
} shash_table_t;
```

Rewrite the previous functions using these data structures:

- `shash_table_t *shash_table_create(unsigned long int size);`
- `int shash_table_set(shash_table_t *ht, const char *key, const char *value);`

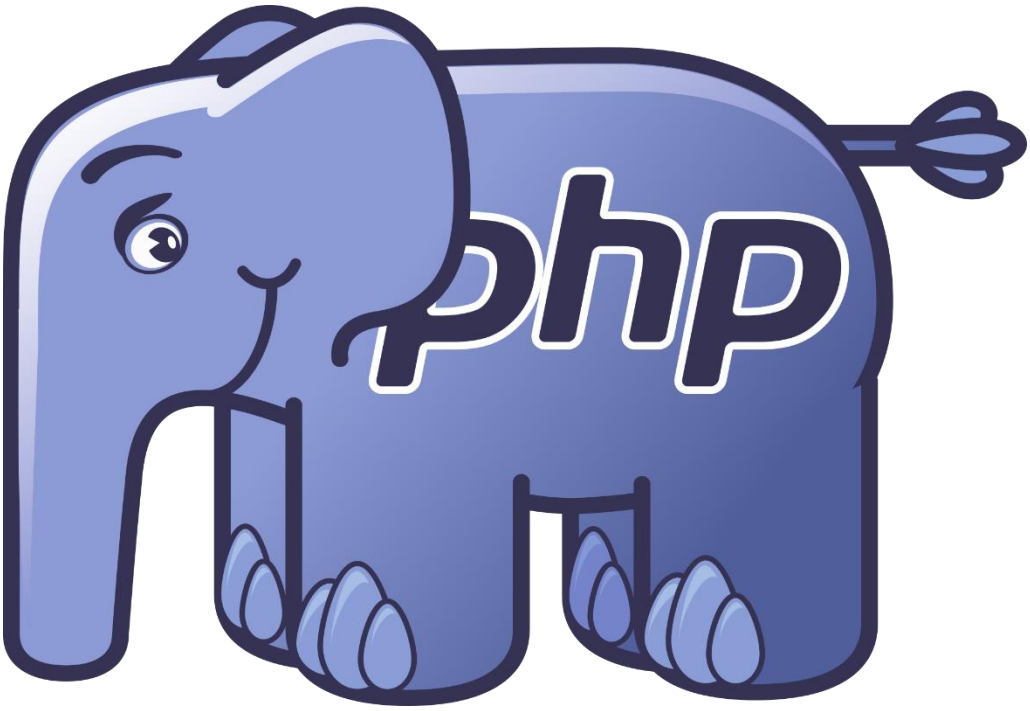
    - The key/value pair should be inserted in the sorted list at the right place
    - Note that here we do not want to do exactly like PHP: we want to create a sorted linked list, by key (sorted on ASCII value), that we can print by traversing it. See example.
- `char *shash_table_get(const shash_table_t *ht, const char *key);`
- `void shash_table_print(const shash_table_t *ht);`
    - Should print the hash table using the sorted linked list
- `void shash_table_print_rev(const shash_table_t *ht);`
    - Should print the hash tables key/value pairs in reverse order using the sorted linked list
- `void shash_table_delete(shash_table_t *ht);`
- You are allowed to have more than 5 functions in your file

```
julien@ubuntu:~/0x1A. Hash tables$ cat 100-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "hash_tables.h"

/**
 * main - check the code
 *
 * Return: Always EXIT_SUCCESS.
 */
int main(void)
{
    shash_table_t *ht;

    ht = shash_table_create(1024);
    shash_table_set(ht, "y", "0");
    shash_table_print(ht);
    shash_table_set(ht, "j", "1");
    shash_table_print(ht);
    shash_table_set(ht, "c", "2");
    shash_table_print(ht);
    shash_table_set(ht, "b", "3");
    shash_table_print(ht);
    shash_table_set(ht, "z", "4");
    shash_table_print(ht);
```

```
    shash_table_set(ht, "n", "5");
    shash_table_print(ht);
    shash_table_set(ht, "a", "6");
    shash_table_print(ht);
    shash_table_set(ht, "m", "7");
    shash_table_print(ht);
    shash_table_print_rev(ht);
        shash_table_delete(ht);
    return (EXIT_SUCCESS);
}
julien@ubuntu:~/0x1A. Hash tables$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 100
-main.c 100-sorted_hash_table.c 1-djb2.c 2-key_index.c -o sht
julien@ubuntu:~/0x1A. Hash tables$ ./sht
{'y': '0'}
{'j': '1', 'y': '0'}
{'c': '2', 'j': '1', 'y': '0'}
{'b': '3', 'c': '2', 'j': '1', 'y': '0'}
{'b': '3', 'c': '2', 'j': '1', 'y': '0', 'z': '4'}
{'b': '3', 'c': '2', 'j': '1', 'n': '5', 'y': '0', 'z': '4'}
{'a': '6', 'b': '3', 'c': '2', 'j': '1', 'n': '5', 'y': '0', 'z': '4'}
{'a': '6', 'b': '3', 'c': '2', 'j': '1', 'm': '7', 'n': '5', 'y': '0', 'z': '4'}
{'z': '4', 'y': '0', 'n': '5', 'm': '7', 'j': '1', 'c': '2', 'b': '3', 'a': '6'}
julien@ubuntu:~/0x1A. Hash tables$
```



Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1A-hash_tables`
- File: `100-sorted_hash_table.c`

Done! Help Check your code Get a sandbox QA Review

Copyright © 2022 ALX, All rights reserved.