

- Login with
  - or email
  - Sign Up for a Free Account
- Help

By keyword

SPARKNOTES

NO FEAR

TEST PREP

VIDEO

SPARKLIFE

THE MINDHUT

Home → SparkNotes → Computer Science Study Guides → Hash Tables → Coding up a Hash Table

## CONTENTS

### General Info

[Introduction and Summary](#)
[Terms](#)

### Summary and Analysis

[What is a Hash Table?](#)
[Problems](#)
[Hash Functions](#)
[Problems](#)
[Coding up a Hash Table](#)
[Problems](#)
[Another use of hashing: Rabin-Karp string searching](#)
[Problems](#)

### Study Tools

[How to Cite This SparkNote](#)

## HASH TABLES



### Coding up a Hash Table



Let's implement a hash table in C. We'll write a hash table that stores strings, and to handle collisions we'll use separate chaining.

#### Data structures

First we define our data structures

1. We begin with our linked lists (for separate chaining):

```
typedef struct _list_t_ {
    char *string;
    struct _list_t_ *next;
} list_t;
```

2. Now we need a hash table structure.

```
typedef struct _hash_table_t_ {
    int size;          /* the size of the table */
    list_t **table;    /* the table elements */
} hash_table_t;
```

Why did we declare the table as `list_t **table`? We don't know up front how big we want the table to be. Therefore, we need to make the table a dynamic array. Remember that an array is just a big block of memory and is basically synonymous with a pointer (see the SparkNotes on [arrays](#) and [pointers](#)). What we have is a pointer to a pointer to a linked list; thus `list_t **table`.

#### Functions

What basic operations do we need to be able to perform with our hash tables?: 1) We need to be able to create a table. 2) We need to be able to hash; thus we need a hash function. 3) We need to be able to free a table. 4) We need to be able to insert into them. 5) We need to be able to lookup an element in them. That should do it for a basic implementation.

## FOLLOW US

Like 326,320 people like this. Be the first of your friends.

Follow @SparkNotes

## Take a Study Break!



**I DON'T GO BANANAS JUST FOR ANY MOVIE**

We give *Mockingjay Part 1* TEN FLAMING ARROWS!



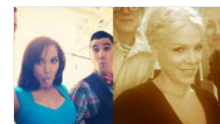
**IMPOSSIBLY BEAUTIFUL PEOPLE**

Willow tells SparkLife why the Prim-Katniss connection is EVERYTHING in *Mockingjay*



**THE BEST FRANCHISE IN THE HISTORY OF THE WORLD**

How much do JLAW and Josh Hutch love each other? We asked them!



**CELEBRITY TWITTER**

Which celeb just gave their hair the chop?

1. Creation. We need to be able to create a hash table, something like:

```
hash_table_t *my_hash_table;
int size_of_table = 12;
my_hash_table =
create_hash_table(size_of_table);
```

The creation function might look something like this:

```
hash_table_t *create_hash_table(int size)
{
    hash_table_t *new_table;

    if (size < 1) return NULL; /* invalid size
    for table */

    /* Attempt to allocate memory for the table
    structure */
    if ((new_table =
    malloc(sizeof(hash_value_t))) == NULL) {
        return NULL;
    }

    /* Attempt to allocate memory for the table
    itself */
    if ((new_table->table =
    malloc(sizeof(list_t *) * size)) == NULL) {
        return NULL;
    }

    /* Initialize the elements of the table */
    for(i=0; i<size; i++) new_table->table[i] =
    NULL;

    /* Set the table's size */
    new_table->size = size;

    return new_table;
}
```

2. Our hash function. We'll go with a relatively simple one.

```
unsigned int hash(hash_table_t *hashtable, char
*str)
{
    unsigned int hashval;

    /* we start our hash out at 0 */
    hashval = 0;

    /* for each character, we multiply the old
    hash by 31 and add the current
    * character. Remember that shifting a
```



**THE WINTER  
APOCALYPSE  
JUST GOT  
SERIOUSLY  
STYLISH**

Every gift you  
could give  
yourself, laid  
out  
BEAUTIFULLY



**THE GIFTIEST  
GUIDE**

Try our  
ULTIMATE S.O.  
GIFT  
GENERATOR

## Take a Study Break!



**TRAVEL**

We rank the  
geekiest cities  
in the world



**IDIOTS**

The top 14  
biggest IDIOTZ  
of tv and  
movies



**POKEMON**

Pokemon as  
past presidents

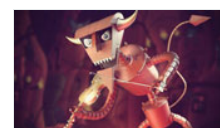


We rank the  
scariest clowns  
from anything  
EVER



**LETHAL FOODS**

These foods  
will KILL YOU



**POP CULTURE**

Satan's top 15  
appearances in  
pop culture

number left is equivalent to

\* multiplying it by 2 raised to the number of places shifted. So we

\* are in effect multiplying hashval by 32 and then subtracting hashval.

\* Why do we do this? Because shifting and subtraction are much more

\* efficient operations than multiplication.

\*/

```
for(; *str != '\0'; str++) hashval = *str +
(hashval << 5) - hashval;
```

/\* we then return the hash value mod the hashtable size so that it will

\* fit into the necessary range

\*/

```
return hashval % hashtable->size;
```

}

3. String lookup. Doing a string lookup is as simple as hashing the string, going to the correct index in the array, and then doing a linear search on the linked list that resides there.

```
List_t *lookup_string(hash_table_t *hashtable,
char *str)
```

```
{
```

```
    List_t *list;
```

```
    unsigned int hashval = hash(hashtable,
```

```
str);
```

/\* Go to the correct list based on the hash value and see if str is

\* in the list. If it is, return return a pointer to the list element.

\* If it isn't, the item isn't in the table, so return NULL.

\*/

```
for(list = hashtable->table[hashval]; list
```

```
!= NULL; list = list->next) {
```

```
    if (strcmp(str, list->str) == 0) return
```

```
list;
```

```
}
```

```
return NULL;
```

```
}
```

4. Inserting a string. Inserting a string is almost the same as looking up a string. Hash the string. Go to the correct place in the array. Insert the new string at the beginning.

```
int add_string(hash_table_t *hashtable, char
*str)
```

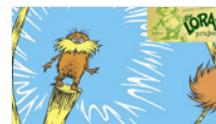
```
{
```

```
    List_t *new_list;
```



#### FICTIONAL CHARACTERS

We Love These YA Antiheroes



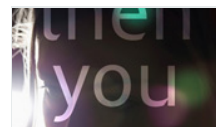
#### HAPPY MOVEMBER

The 6 Greatest Fictional Mustaches



#### ASK A LITERARY LADY

HELP—My Friend Takes Credit for the Books I Recommend!



#### YA NOVELS

4 Mistresses of Dark Contemporary YA



#### BRAINBENDERS

These Books Will Blow Your Mind

```

    list_t *current_list;
    unsigned int hashval = hash(hashtable,
str);

    /* Attempt to allocate memory for list */
    if ((new_list = malloc(sizeof(list_t))) ==
NULL) return 1;

    /* Does item already exist? */
    current_list = lookup_string(hashtable,
str);

    /* item already exists, don't insert it
again. */
    if (current_list != NULL) return 2;
    /* Insert into list */
    new_list->str = strdup(str);
    new_list->next = hashtable->table[hashval];
    hashtable->table[hashval] = new_list;

    return 0;
}

```

5. Deleting a table. Freeing up the memory you use is a very good habit, so we write a function to clear out the hashtable.

```

void free_table(hash_table_t *hashtable)
{
    int i;
    list_t *list, *temp;

    if (hashtable==NULL) return;

    /* Free the memory for every item in the
table, including the
* strings themselves.
*/
    for(i=0; i<hashtable->size; i++) {
        list = hashtable->table[i];
        while(list!=NULL) {
            temp = list;
            list = list->next;
            free(temp->str);
            free(temp);
        }
    }

    /* Free the table itself */
    free(hashtable->table);
    free(hashtable);
}

```

**When your books and teachers don't make sense, we do.**

[Contact Us](#)

[Legal](#)

[About](#)

[Sitemap](#)

[Mobile Apps](#)

[Advertise](#)

SparkNotes is brought to you by B&N. Visit B&N to buy and rent textbooks, and check out our award-winning tablets and ereaders, including Samsung Galaxy Tab 4 NOOK and NOOK GlowLight.

© 2014 SparkNotes LLC, All Rights Reserved