Final assignment: *Hash map*

1 Introduction

In this final assignment you will implement a *hash map*[1]. A hash map is a data structure that associates a *key* with a *value* (a chunk of data). Most hash maps are implemented as an array of so-called *buckets*. A *hash function* translates a given key (e.g., a name) to an index in the array, where the corresponding bucket is stored.

Below we will specify the data structures that you have to provide, and the functions that you have to implement. This assignment includes two bonus functions that can raise your score from pass (C) to good (B) to excellent (A).

2 Hash map structure

Define a type `HashMap`, which represents the hash map data structure.

Note: Use `typedef` such that a `HashMap` structure can be used without using the `struct` keyword, i.e. the following construction should be possible:

```
HashMap *hm;
```

3 Creating a hash map

1. Implement a function `create_hashmap` that returns a pointer to the newly constructed `HashMap` structure and has parameter

- `key_space`, a `size_t`[2] that represents the number of buckets in the hash map.

This function should allocate enough memory to fit `key_space` buckets, and the allocated memory should be zeroed (i.e., `NULL`ed).

2. A hash function maps a string (i.e. an array of `char`s ending with a null character) to an index, so it returns a `unsigned int`. The parameter of a hash function is simply a

- `key`, a null-terminated string of characters.

As the hash map can only hold up to `key_space` buckets, using the hash function –for example to lookup a mapping– requires some care; apply modulo `key_space` to the result such that the value will be in the available bucket range.

[1] `http://en.wikipedia.org/wiki/Hashmap`

[2] `http://en.wikipedia.org/wiki/Size_t`

3. A default hash function named `hash` should be implemented. This function should sum all *ASCII* values of the characters of the key.

For example:

```
char *key = "AC";
unsigned int h = hash(key);
=> h = 132
```

4 Inserting data

Implement a function `insert_data` that has parameters

- `hm`, a pointer to a hash map;
- `key`, a null-terminated string of characters;
- `data`, a `void` pointer to the source data;
- `resolve_collision`, a `ResolveCollisionCallback` (see below).

The function should store the data pointer and a copy of the key in the bucket that can be found by applying the hash function on the key. In case of a collision, i.e. when there already is data with the same key in the hash map, the `resolve_collision` function should be called with the the previously stored data and `data` as arguments and the returned `void` pointer should be stored in the bucket instead.

`ResolveCollisionCallback`, a *pointer to* a function that returns a `void` pointer and has two parameters:

- `old_data`, a `void` pointer to the previously stored data;
- `new_data`, a `void` pointer to the data that is being newly inserted.

The function should determine what data is stored in the has map in case of a key collision by returning the `void` pointer to the data that is to be stored.

5 Retrieving data

Implement a function `get_data` that has parameters

- `hm`, a pointer to a hash map;
- `key`, a null-terminated string of characters.

The function should return the data pointer (a `void` pointer) in the hash map that is associated with the key. If the key is not present in the hash map, `NULL` should be returned.

6 Iterator

Implement a function iterate that has parameters

- hm, a pointer to a hash map;
- callback, a pointer to a function that returns nothing (i.e. void) and has two parameters:
  - key, a null-terminated string of characters;
  - data, a void pointer to the data.

This function should iterate over the entire hash map. For each data element it finds, the callback function should be called with the two members of the element.

7 Removing data

Implement a function remove_data that has parameters

- hm, a pointer to a hash map;
- key, a null-terminated string of characters.
- destroy_data, a DestroyDataCallback (see below).

This function should remove the element in the hash map that is associated with the given key. If the destroy_data parameter is non-NULL it should be called with the data pointer of the element as argument. If the key is not present, the hash map should remain untouched. As the remove_data function cannot fail, its return type is void.

DestroyDataCallback, a *pointer to* to a function that returns nothing (i.e. void) and has one parameter:

- data, a void pointer.

The function should clean up the data (e.g. free allocated memory).

8 Deleting a hash map

Implement a function `delete_hashmap` that has parameters

- `hm`, a pointer to the hash map that is to be deleted;
- `destroy_data`, a `DestroyDataCallback` (see 7).

The function should deallocate *all* memory that was allocated by the hash map. If the `destroy_data` parameter is non-`NULL` it should be called for every data element that is stored in the hash map with the data pointer of the element as argument.

9 *Bonus:* New hash function

Implement a function `set_hash_function` that has parameters

- `hm`, a pointer to a hash map;
- `hash_function`, a pointer to a hash function that returns a `unsigned int` and a single parameter:
    - `key`, a null-terminated string of characters.

This function should set `hash_function` as the new hash function of the hash map `hm`. Changing the hash function means that a particular key may now be hashed to different bucket than it was with the previous hash function. The hash map must be updated (rehashed) to reflect this so that all data in the hash map can still be retrieved with their corresponding keys.

10 *Bonus:* Counting Words

Implement a function `count_words` that has parameters

- `stream`, a pointer to a `FILE`.

This function should count the number of times each word in the stream occurs using the hash map you implemented. A word is defined as a sequence of one or more alphanumeric characters. You may use `fscanf`[3] to read a particular set of characters from a stream but other solutions are also accepted. The data stored in the hash map should be properly allocated and deallocated, do not simply store an integer that is cast to a pointer type. The return type of the function is `void`.

[3] `http://en.cppreference.com/w/c/io/fscanf`

Given the input:

```
foo bar_, foo!
bar    "baz".
foo?
```

The program should write the following to the standard output:

```
bar: 2
baz: 1
foo: 3
```

The order in which the output is printed is not important.

11 Submission

Finished assignments should be uploaded to CPM, and submissions should adhere to the following requirements:

- Put all the hash map source code inside a file called `hashmap.c`;
- Put all the word count source code inside a file called `wordcount.c`;
- Include a header file `hashmap.h` that contains all your (public) hash map function prototypes and data types. In the header file, also place the following preprocessor directive *if* you have completed the first bonus exercise:

  ```
  #define NEW_HASH
  ```

- Include a header file `wordcount.h` *only if* you have completed the Counting Words bonus exercise that contains the public function prototype.
- Do *not* include a `main` function. (We will use our own test driver, just like the example test provided.)
- Create a ZIP file of the `hashmap.[ch]` files by executing `make submit`. Upload this ZIP file to CPM.

Submissions violating the above requirements will be rejected.