Lab 13 (Bonus) – Hash Tables and Dictionaries

General Requirements

None of the functions you write should call `calloc` or `realloc`.

None of the functions you write should perform console input; i.e., contain `scanf` statements. Unless otherwise specified, none of your functions should produce console output; i.e., contain `printf` statements.

You must format your C code so that it adheres to one of two commonly-used conventions for indenting blocks of code and placing braces (K&R style or BSD/Allman style). Instructions were given in the VS Code installation instructions.

Finish each exercise (i.e., write the function and verify that it passes all of its tests) before you move on to the next one. Do not leave testing until after you have written all your functions.

Instructions

Step 1: Depending on your operating system, download (**lab13_win.zip** or **lab13_mac.zip**) the starting code from Brightspace and extract the zipped file.

Step 2: Double click on lab13.code-workspace

Step 3: lab13_xxx folder contains the following files:

- dictionary.c contains an incomplete implementation of a dictionary module.
- dictionary.h contains the declarations for the dictionary data structure and the nodes the dictionary (see the `typedef`s for `entry_t` and `entry_t *`), followed by the prototypes for functions that operate on the dictionary.
- main.c contains a simple *test harness* that exercises the functions in dictionary.c. Unlike the test harnesses provided in several labs, this one does not use the sput framework. The harness doesn't compare the actual and expected results of each test and keep track of the number of tests that pass and fail. Instead, the expected and actual results are displayed on the console, and you have to review this output to determine if the functions are correct. **Do not modify `main()` or any of the test functions.**

Exercise 1

Add the declaration:

```
void print_dictionary(dict_t *dict);
```

to the list of function prototypes in dictionary.h (not dictionary.c).

Add this header comment and the incomplete definition of `print_dictionary` to dictionary.c:

```
/* Print the dictionary pointed to by dict, using the format:

0: key_0: value_0, key_1: value_1, ...
1: key_0: value_0, key_1: value_1, ...
...
n-1: key_0: value_0, key_1: value_1, ...
*/
void print_dictionary(dict_t *dict)
{
}
```

Each line of output produced by `print_dictionary` contains a slot number in the dictionary's hash table, followed by a comma-separated list of all the key/value pairs in that slot's chain. If the chain is empty, `NULL` is output. An example is shown below.

Finish the definition of `print_dictionary`.

Edit `main` (in main.c), adding a call to `print_dictionary`, immediately above the statement:

```
return EXIT_SUCCESS;
```

Build the project, correct any compilation errors, then execute the project. Verify that the console output produced by `print_dictionary` looks like this:

```
0: NULL
1: NULL
2: Babak: babak@sce.carleton.ca
3: NULL
4: Donald: donald.bailey@carleton.ca
5: NULL
6: Don: donald.bailey@carleton.ca
7: NULL
8: Jason: jason.jaskolka@carleton.ca, Lynn: lynn.marshall@carleton.ca
9: Greg: gregory.franks@carleton.ca
10: NULL
```

Use the console output to help you identify and correct any flaws.

Exercise 2

Add the declaration:

```
_Bool replace(dict_t *dict, char *key, char *value);
```

to the list of function prototypes in dictionary.h (not dictionary.c).

Add this header comment and the incomplete definition of `replace` to dictionary.c:

```
/* If key is in the dictionary, replace its associated value with
 * a copy of the string pointed to by value, and return true.
 * If key is not in the dictionary, return false to indicate that
 * the dictionary has not changed.
 * Terminate via assert if memory couldn't be allocated while
 * updating the dictionary.
 */
_Bool replace(dict_t *dict, char *key, char *value)
{
}
```

Finish the definition of `replace`.

Edit `main` (in main.c), adding statements to test `replace`. There should be tests that help you determine if `replace` updates the dictionary correctly if a person's name is in the dictionary. You'll also need to test the case in which a person's name is not in the dictionary. Remember to check the value returned by `replace`. Use `print_dictionary` to help you check the entire dictionary after a call to `replace` returns.

Build the project, correct any compilation errors, then execute the project. Use the console output to help you identify and correct any flaws.

Exercise 3

Add the declaration:

```
void clear(dict_t *dict);
```

to the list of function prototypes in dictionary.h (not dictionary.c).

Add this header comment and the incomplete definition of `clear` to dictionary.c:

```
/* Remove all the key/value entries from the dictionary.
 * The dictionary will be empty after this call returns.
 */
void clear(dict_t *dict)
{
}
```

Finish the definition of `clear`. Be careful: the nodes in the chains were allocated from the heap (see `put`), so they must be deallocated correctly when the dictionary is cleared; otherwise, you can end up with *memory leaks* (blocks of memory that were not released when they are no longer needed, but can no longer be accessed, because the pointers to the blocks no longer exist). Remember, the C standard doesn't specify whether `free` modifies the contents of blocks of memory when they are returned to the heap's free store. You should never access the members of a linked list's node after the pointer to the node has been passed to `free`, because you don't know if the node's members were overwritten by `free`.

Edit `main` (in main.c), adding statements to test `clear`. If you call `print_dictionary` after the email contact list is cleared, this should be displayed:

```
0: NULL
1: NULL
2: NULL
3: NULL
4: NULL
5: NULL
6: NULL
7: NULL
8: NULL
9: NULL
10: NULL
```

Build the project, correct any compilation errors, then execute the project. Use the console output to help you identify and correct any flaws.

Wrap-up

Submit main.c, dictionary.h, and dictionary.c to Brightspace.

Before submitting your lab work

- Make sure that main.c, dictionary.h, and dictionary.c have been formatted to use K&R style or BSD/Allman style.
- Ensure you're submitting the files that contains your solutions, and not the unmodified files you downloaded from Brightspace!