Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Fall 2022

Lab 11 – Hash Tables and Dictionaries

Note that if you use a different IDE, e.g. CLion, you need to adjust the Pelles C instructions as required.

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 for selecting the formatting style and formatting blocks of code are in the Lab 1 handout.

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. Don't leave testing until after you've written all your functions.

Getting Started

Step 1: Launch Pelles C and create a new Pelles C project named dictionary. (Instructions for creating projects are in the handout for Lab 1.) Select Win 64 Console program (EXE) as the project type. **Don't click the icons for Console application wizard, Win64 Program (EXE) or any of the other "Empty projects" icons. These are not correct types for this project.**

When you finish this step, Pelles C will create a folder named dictionary.

Step 2: Download lab11.zip from Brightspace. Open lab11.zip and locate the files named main.c, dictionary.c and dictionary.h. Move these files into your dictionary folder.

Step 3: Add main.c and dictionary.c to your project. (Instructions for doing this are in the handout for Lab 1.) You don't need to add dictionary.h to the project. Pelles C will do this after you've added main.c.

As you add the files, icons labelled main.c and dictionary.c will appear in the Project window, below the Source files icon. An icon labelled dictionary.h will appear below the Include files icon.

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

Get your lab marked during your lab by a TA. Submit main.c, dictionary.h, and dictionary.c to Brightspace **immediately** after the TA has marked them. (Do not wait for the deadline as the TA may input a grade of zero if your work is not there by the end of your lab!)

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, as explained in *General Requirements*.
- Ensure you're submitting the files that contains your solutions, and not the unmodified files you downloaded from Brightspace!
- **Remember that your mark will be 0 if a TA did not check your work in the lab or if you did not submit your final version immediately after the TA checked your work**

History

Nov. 5, 2022: Initial release.