Homework 7 C Programming - Dictionary as ordered linked list

Due Wednesday 10/23/2019 11pm

This assignment you will define a few functions that can complete certain tasks.

Download hw7handout.tar

Use instructions similar to homework 2 to download hw7handout.tar file, copy it to w204 machine's 311 folder, use `tar xvf` command to unpack it to hw7handout folder.

Functions to implement

You are to implement four functions, see comments about what each function should do. We recommend you implement them in the order given, from the easist countKeys to the hardest addKey.

```
// count number of keys in a dict.
int countKeys(const dictNode *dict) {
  return 0;
}

// given a key, look up its corresponding value in the
// dictionary, returns -1 if the value is not in the dictionary.
// your search for key should end when the key in the next node
// is bigger than the lookup key or you reached the end of the
// list.
int lookupKey(const dictNode *dict, const char *key) {
  return -1;
}

// delete the node in a dict with given key, return the value of
// the deleted node if it was found, return -1 if it wasn't found.
int deleteKey(dictNode **dictPtr, const char *key) {
  return -1;
}

// given a key/value pair, first lookup the key in the dictionary,
// if it is already there, update the dictionary with the new
// value; if it is not in the dictionary, insert a new node into
// the dictionary, still make sure that the key is in alphabetical
// order.
// IMPORTANT: When creating a new node, make sure you dynamically
```

```
// allocate memory to store a copy of the key in the memory. You
// may use strdup function. DO NOT STORE the input key from the
// argument directly into the node. There is no guarantee that key
// pointer's value will stay the same.
// YOU MUST KEEP THE ALPHABETICAL ORDER OF THE KEY in the dictionary.
void addKey(dictNode **dictPtr, const char *key, int value) {
  return;
}
```

Useful library functions

The following library functions would be helpful during your implementation. use the command

%**man functionName**

to learn more about how to use these library functions.

```
int strcmp(const char *s1, const char *s2);
char *strdup(const char *s);

void *malloc(size_t size);
void free(void *ptr);
```

Edit/Compile/Test your C code

1. Compile your code

Assuming you successfully copied and unpacked hw3handout.tar file in ~/311 folder on your W204 account, the following command will help you compile the given C program. The original tar file contains a complete C program that compiles and works.

```
cse-p204inst11.cse.psu.edu 160% cd ~/311/hw7handout
```

You use **make** command in the hw7handout folder to compile all the .c files to create an executable called **dict**.

Make sure that make does not generate any error/warning messages while compiling your C code. Any warning/error messages while compiling your code will result in a zero for this assignment. You must address these warnings/errors and remove all compiler warnings/error messages before you submit. To be safe, do the following before you submit to ensure no warnings/errormessages were produced.
%make clean
%make

2. Test your code

2.1 Testing with test functions provided in main.c

You start with code that contains functions that don't really do anything. If you test your code by executing ./dict before implementing anything, it will fail assertion of the first test function. When you successfully implemented all functions, your output should look like this:

```
cse-p204inst22.cse.psu.edu 106% make
gcc -Wall -Wuninitialized -Og   -c -o main.o main.c
gcc -Wall -Wuninitialized -Og   -c -o dict.o dict.c
gcc -Wall -Wuninitialized -Og   -c -o dict_support.o dict_support.c
gcc -Wall -Wuninitialized -Og -o dict main.o dict.o dict_support.o
cse-p204inst22.cse.psu.edu 107% ./dict
passed testCount
passed testLookup
passed testDelete
passed testAdd
```

2.2 Testing with your own tests examples

If you managed to pass all the existing tests provided in main.c file, we recommend you to try to write your own testing functions. Read examples in main.c to see how to create a list from two arrays (using **makeDict**) and how to assert if your dict is identical as a dict described by two arrays (using **assertEqualDict**). It might also be helpful to use **displayDict** to print out what your dictionary looks like in your code. Just make sure that if you add any printf/displayDict in dict.c's function definition during the debugging phase, take them out before submitting.

3. Edit your code

Before you start, please make sure to write your name and email at the beginning of dict.c code to replace Prof. Wang's name and email.

```
// Author: Yanling Wang
// Email: yuw17@psu.edu
```

Edit your code in `dict.c` with vim and save it and repeat step 1/2 to compile and test your code.

Always start with small changes before your compile and test your code.

Keep your code properly indented.

Add comment in your function to explain your algorithm for this assignment. Use block comments that are on its own lines. Do not use tail comments at the end of each line of the code.

Submit your C code

You are to submit only the **dict.c** file to gradescope.

Gradescope is not responsible to provide a thorough test for your submitted code, we will also read your code and grade your code with more tests as well as looking out for warning/error messages from gcc. The testsuites we provided is also rudimentary to catch common errors but it is up to you to find ways to thoroughly test your functions.