Project 1: Philspel (due 7/1)

This project is designed to serve as an introduction to the C language. To complete it, you will have to use the C file input/output library, do memory allocation, manipulate strings, and coerce strings to void pointers and vice versa. Although there is conceptually a lot to learn to complete this project, the actual amount of code you will need to write is short. For reference, the “official” solution only adds about 120 lines including comments.

`philspel` is a very simple and silly spelling checker. It accepts a single command line argument: the file name of a dictionary to use. For example, to use philspel, all you would need to type into the terminal is:

```
$ ./philspel dictionary.txt
```

Of course, you can replace `dictionary.txt` with whichever file you wish to use for your dictionary.

This dictionary consists of a list of valid words to use in checking the input. `philspel` processes standard input and copies it to standard output. For each word (strictly defined as a sequence of letters unbroken by any non-letter character) in the input, `philspel` looks for all of the following variations of the word in its dictionary:

1. The word itself.
2. The word converted entirely to lowercase letters
3. The word with all but the first letter converted to lowercase.

If any of the three variations are found in the dictionary, the word is copied directly to standard output. Otherwise, the word is copied to standard output with the string `" [sic]"` (without the quotation marks, but *with the spaces*) appended. All other input (e.g. whitespace) is copied to standard output unchanged.

Getting Started

1. Please follow the directions here to get a repository: https://forms.gle/QQiz3sh33fnfqyxTA
2. `git clone https://github.com/61c-student/su20-proj1-<Your User Name>.git`
3. `cd su20-proj1-<Your User Name>`

This copies over a Makefile for the project and several partially implemented source files.

`hashtable.c` and `hashtable.h` define a simple generic chained hash table. `hashtable.h` acts as an interface through which other files interact with our hash table implementation. Likewise, `philspel.h` declares the functions that are defined in `philspel.c`. `philspel.c` contains the main function, which serves as the program’s entry point.

For those who want a review of hashing, these Hug slides are a good reference.

To finish this project, you will have to fill in the definition of `struct HashTable` in `hashtable.h` and correctly implement the following functions:

1. `void insertData(HashTable *table, void *key, void *data)` in `hashtable.c`
2. `void *findData(HashTable *table, void *key)` in `hashtable.c`
3. `unsigned int stringHash(void *s)` in `philspel.c`
4. `int stringEquals(void *s1, void *s2)` in `philspel.c`
5. `void readDictionary(char *dictName)` in `philspel.c`
6. `void processInput()` in `philspel.c`

We suggest working on the subparts in the order listed above. Also, note that you may NOT import additional files and that all your changes should be contained to `hashtable.c`, `hashtable.h`, and `philspel.c`.

Remember to regularly check the piazza thread and pull from the starter repo to ensure that you get any updates/bug fixes.

Testing your program

Also included in the repo is a sample dictionary, input, and output. Your output should **EXACTLY** match ours, since we will be using automated scripts to grade your program. Another useful dictionary for testing is contained in `/usr/dict/words` or `/usr/share/dict/words` depending on the system. You can type `make test` in your project 1 directory to compile and test your program against the sample dictionary/input. You can also safely output all sorts of debugging information to `stderr`, as this will be ignored by our scripts and by the test routine provided in the Makefile.

Furthermore, you can initially assume that both the dictionary and the input won't contain words longer than 60 characters. This gets you the majority of the credit. However, for full credit, you must ensure that your program fully works even if you get words which are arbitrarily longer than 60 characters.

You should assume that the dictionary is well formatted (i.e., individual words delimited by new lines) if it exists. You CANNOT assume anything about what comes in through standard input except for the length of "words" for 80% credit.

Tip: Consider running your program under valgrind to detect any memory errors. For example, use `valgrind -q ./philspel ...` to have valgrind print out any invalid memory accesses it detects during the run. In C, it's easy to write a program that appears to work correctly for simple tests, but which fails or crashes with larger inputs. Valgrind catches many of these hidden bugs that might otherwise appear only in the autograder.

Submitting

Please submit using Gradescope to Project 1, using the GitHub submission option to ensure that your files are in the right place.

REMEMBER: the grading will be done almost entirely by automated scripts. We will be only using your `hashtable.c`, `hashtable.h`, and `philspel.c` files when grading! Your output must exactly match the specified format, making correctness the primary goal of this project. Upon submission, the autograder will give you the result of a basic sanity test but **will not** give you your complete grade. Rather, you are responsible for developing any tests you need to make sure that you meet the requirements of the project. We deliberately did not include a more comprehensive test. We want you to practice writing your own tests!

You are to do this work individually. It is OK to assist your classmates with their project, but don't copy code, and don't go searching for previous solutions!

FAQ

The autograder failed to execute correctly?!?!

This is typically due to you having a lot of print statements which will cause the autograder to hang or run out of memory when it is executing. If you run into this, please remove/comment out your print statements and resubmit. If you continue to have this issue after you have confirmed that you have done that **AND** you have confirmed that you pass the sanity test locally, please make a post on piazza **with** a link to your submission. If you do not include a link, we will not be able to easily find your submission!