ECS 50 Homework 0: `philspel`, a simple, silly spellin [sic] checker

Nicholas Weaver

March 26, 2024

This homework 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, invoke pointers to functions, access structures, and coerce strings to void pointers and vice versa. Although there is conceptually a lot to learn to complete this homework, the actual code you need to write is short[1]. If you are familiar with C already this should be very straightforward but even if you aren't it is designed to cover the language well and also to test your programming maturity.

This homework is due on April 10th at 10pm Pacific time. This is a *hard deadline*.

`philspel`[2] is a very simple and silly spelling checker. It accepts a single command line argument, the name of a dictionary to use. 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 (sequence of letters unbroken by any non-letter character) in the input, it looks that word, that word converted entirely to lowercase letters, and that 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 to standard output. Otherwise, the word is copied to standard output, with the string `" [sic]"` (without the quotation marks *but with the space*) appended. All other input is copied to standard output unchanged.

You should download the hw0.zip file from Canvas and place that into a git archive for your use. Make sure your git repository is not publically available.

This file includes a Makefile for the homework and several code files. `hashtable.c` and `hashtable.h` are the code and header files which define a simple generic hashtable. You do not need to implement any functions in the hashtable as they are provided for you. A hashtable is an efficient key/value store with (expected) constant time lookup behavior. Python's `dictionary` type is a hashtable under

[1]For reference, the "official" solution only adds less than 100 lines including comments.

[2]The first incarnation of `philspel` was written by one Phillip "Edward" Nunez, a mythical student member of UC Berkeley's Computer Science Undergraduate Association, who's program generally worked 'as advertised' but in rather unusual ways.

the hood[3].

`philspel.h` defines the functions in `philspel.c`. You will need to implement 4 functions in `philspel.c`: `stringHash(void *s)`[4]., `stringEquals(void *s1, void *s2)`, `readDictionary(char *filename)`, and `processInput()`. You may modify `philspel.h` if you wish to declare additional helper functions which you implement in `philspel.c`.

Also included 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 homework 0 directory to compile and test your program against a sample set of inputs. 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 80% credit. However, for the final 20% credit, you should insure that your program fully works if you get words which are longer than 60 characters.

You can assume that the dictionary is well formatted if it exists[5] so individual words separated by newlines. You can NOT assume anything about what comes in on standard input except for the length of "words" for 80% credit.[6]

You do NOT need to free the contents of the dictionary when you are done processing standard input: you need to use the dictionary for the life of the program so manually freeing it just before you quit makes no sense. Additionally, Valgrind will use this to mark the data in the dictionary as "still reachable" when doing an analysis for memory leaks. However you should NOT have memory leaks in `processInput` as we will check your program with large inputs that will fail if you have a memory leak in that function.

Please submit using Gradescope: you will submit both `philspel.c` and `philspel.h`.

REMEMBER: the grading will be done almost entirely by automated scripts. Your output must exactly match the specified format, which makes

[3]One problem with the C language in general is it doesn't have a rich standard library, unlike Java, Python, Kotlin, or modern C++. Thus we can't just declare a `dictionary` and use that, instead we have to provide our own implementation. Additionally, C's support for proper generic typing and object-oriented paradigms is likewise nonexistant, which is why the initialization of the hashtable has to accept pointers-to-functions to specify the hash function and equality operations, and the arguments are `void *`, that is, pointers to unspecified data.

[4]A hashfunction is a deterministic mapping: taking a string and turning it into a number. Identical text MUST return identical hash values, and different text should return different hash values (but it can be the same due to *collisions*). You do not need to implement a particularly good hash function, but the hash function must be correct and, if you use an outside resource such as StackOverflow, you *must* cite your source

[5]that is, each line has a separate word on it, so you can just read in individual lines. The dictionary MAY have spurious non-"word" items however, such as `l33t`, but if such entries are also added into the hashtable it won't actually pose any problems.

[6]Important hint: if you use `getc()` or `getchar()` for processing standard input, look at what the return type for these functions are. You will notice it is *not* a character.

correctness the primary goal of this homework. In submitting 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 you meet the requirements of the homework.

We deliberately did not include a more comprehensive test... That is, you will be unable to use the autograder as an “oracle”, a service to see if you have a correct solution.

You are to do this work individually. It is OK to assist your classmates with their homeworw, but don’t copy code, and don’t go searching for previous solutions! The latter is especially critical: I *know* there are existing solutions to this homework available, and if you can find a solution, assume I’ve already found it myself and will include it in my checking for plagarism.

Tip: consider running your program under valgrind to detect any memory errors. For example, use `cat sampleInput | valgrind ./philspel sampleDictionary` to have valgrind run Philspel on the sample input and provide you a heap and leak summary. Similarly, adding the `-q` flag to valgrind will have it just print out any invalid memory accesses it detects during the run. In C, it is easy to write a program that appears to work correctly for simple tests but will fail or crash with larger inputs. Valgrind catches many of these hidden bugs that might otherwise appear only in the autograder. Valgrind is installed on the lab computers.

Requirements for debugging help: If you are asking the TA or instructor for help debugging your program you must have input that demonstrates the bug, you must show how to run the buggy input under valgrind if the bug is causing a segmentation fault, and you must also show how to run the buggy input in the debugger with a breakpoint set in the buggy function.

You can use whichever IDE or programming environment you want (emacs, VSCode, CLion, `vi`, etc), but your IDE must support setting a breakpoint and you must show the TA your program running in the debugger, paused at the breakpoint, before before they will help you[7].

Special policies for this homework: This homework, unlike other homeworks, *will not be accepted after the due date*! That is, *you can not use slip days to turn in this homework late.* The deadline of April 10th at 10pm Pacific Time is a hard deadline.

Any student who does not turn in this homework *will be dropped from the class.* Students on the waitlist *must turn in the homework* or they will be removed from the waitlist. Finally, any student who is found to commit academic misconduct on this assignment *will receive an F in the class.*

The reason for this policy is unfortunate necessity: Davis does not have the infrastructure necessary to teach this class in a fully scalable manner, letting in every student on the waitlist which is currenty at over 20 students. And due to

[7]We enforce this policy for two reasons. The first is that it is often really hard to diagnose a bug (especially a bug in code you didn’t write) without this information. The second is by the time you do the above you will often have found the bug yourself so you no longer need assistance.

high demand all Computer Science classes at Davis have a 10 day drop deadline, which for this spring is April 12th.