

**CS61C Spring 2016 Project 1: philspel**

TAs: Alex Khodaverdian, Chris Hsu (originally by Nicholas Weaver)

02/07 @ 23:59:59

**Due Tuesday, February 9, 2016 at 23:59:59**

## Goals

The objective of this project is to serve as an introduction to the C language. This project will cover a lot of different features of the C language, including the C file input/output library, memory allocation, string manipulation, and string coercion to void pointers and vice versa. Although there is conceptually a lot to learn to complete this project, the actual code you need to write is relatively short.

## Background

Philspel 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 spaces appended**). All other input is copied to standard output unchanged.

Your assignment is to complete the implementation of `philspel`. To do this, you will need to implement 4 functions in `philspel.c`:

`stringHash(void *s), stringEquals(void *s1, void *s2), readDictionary(char *filename), and processInput()`.

Just as a reminder, this project is to be completed **individually**.

## Setup

To get starter code for this project, we will be using git.

1. First, you need to create a private repo with the format `proj1-xx` where `xx` is your login. The process for this is very similar to what was done in lab. Please make sure this repo is NOT your work repo however. Furthermore, please set this to **PRIVATE**. Just to make sure, set this to **PRIVATE**. If you do not set this to **PRIVATE**, horrible things will happen to you and you will be severely punished. So before continuing, make sure your repo is **PRIVATE**. Not setting your repo to private, even if by mistake, will be seen as an intention to cheat.
2. From your access management settings, give '`cs61c-spring2016`' admin access to your repo. You also must do this from the very beginning to avoid any penalties placed upon your grade.
3. Did you set your repo to **PRIVATE** and give the '`cs61c-spring2016`' admin access? If yes, enter into the directory of your class account that you would like to hold your `proj1-xx` repository. Once in this directory, run the following:

```
git clone https://mybitbucketusername@bitbucket.org/mybitbucketusername/proj1-xx.git
cd proj1-xx
git remote add proj1-starter https://github.com/cs61c-spring2016/proj1-starter.git
git fetch proj1-starter
git merge proj1-starter/master -m "merge proj1 skeleton code"
```

Once you complete these steps, you will have the `proj1` directory inside of your repository, which contains the files you need to begin working.

As you develop your code, you should make commits in your git repo and push them as you see fit. Be sure to commit often! This allows you to save your progress as you go along, as well as gives us an overview of how you approached the project. You should not have just a single giant commit of all your code when you finish the project.

## Overview

After you have a copy of the starter code, make sure you have the following files.

- `hashtable.c` – Code for a generic hashtable.
- `hashtable.h` – Header file for `hashtable.c`. For more information on header files and how they are used, check out [this link](#). You should not need to modify either hashtable file, though take a look at them and see how they work.
- `philspel.c` – Contains various functions for our spellchecker. You will need to implement 4 functions in `philspel.c`: `stringHash(void *s), stringEquals(void *s1, void *s2), readDictionary(char *filename), and processInput()`.
- `philspel.h` – Header file defining functions in `philspel.c`. You may modify `philspel.h` if you wish to declare additional helper functions which you implement in `philspel.c`.
- `Makefile` – Makefile for compiling `philspel`, as well as for running tests. For more information on Makefiles and how they are used, check out [this link](#).
- `sampleDictionary` – Sample dictionary. Another useful dictionary can be found in `/usr/dict/words` on the instructional machines.
- `sampleInput` – Sample input file.
- `sampleOutput` – Sample output file.

- `testOutput` – Test output file.

Spend some time looking over these files and figuring out what is in each file and what they are used for.

## Instructions

As noted above, your task is to implement the 4 functions in `philspel.c`: `stringHash(void *s)`, `stringEquals(void *s1, void *s2)`, `readDictionary(char *filename)`, and `processInput()`. More information about the purpose of each of these functions can be found inside `philspel.c`.

You can type the following in your project 1 directory to compile and test your program against a sample set of inputs:

```
$ make test
```

You can, however, 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.

While completing this project will require getting familiar with many parts of the C language (C file input/output library, do memory allocation, manipulate strings, and coerce strings to void pointers and vice versa just to name a few), your final solution should not be all that long.

Start early! You likely will have to learn and experiment a lot with C to complete this project, so make sure you budget enough time.

For grading purposes, we will be splitting the project into two parts.

1. When you first are writing your solution to the project, you can assume that both the dictionary and the input won't contain words longer than 60 characters. Successfully doing this will be worth 80% of your grade on the project.
2. For the remaining 20% of your grade, we will test your code against both dictionaries and inputs that contain words of an unbounded length. We suggest trying to get a solution with a bounded character count before attempting this part.

## Submissions

There are **two** steps required to submit proj1. Failure to perform both steps will result in loss of credit:

1. First, you must submit using the standard unix `submit` program on the instructional servers. This assumes that you followed the earlier instructions and did all of your work inside of your `git` repository. To submit, follow these instructions after logging into your `cs61c-XX` class account:

```
$ cd proj1-XX
$ submit proj1
```

Once you type `submit proj1`, follow the prompts generated by the submission system. It will tell you when your submission has been successful and you can confirm this by looking at the output of `glookup -t`. Remember, you only need to submit two files: `philspel.c` and `philspel.h`.

2. Additionally, you must submit `proj1` to your bitbucket repository. To do so, follow these instructions after logging into your `cs61c-XX` class account:

```
$ cd proj1-XX
$ git add -u                                     # should add all modified files in proj1 directory
$ git commit -m "Project 1 submission"
$ git tag -f "proj1-sub"                           # The tag MUST be "proj1-sub". Failure to do so will result in loss of credit.
$ git push origin master --tags                  # Note the "--tags" at the end. This pushes tags to bitbucket
```

## Grading

The grading for this project will be done almost entirely by automated scripts. Your output must exactly match the specified format, which makes correctness the primary goal of this project. Again, you are to do this work individually.