Lab 1

20 January 2023

This week's lab is about practicing three things in C: I/O (and strings), arrays, and functions.

Black box spec

Your program will read a sequence of words from the user, which are each supposed to be at most ten characters in length and all lowercase. After every five words, the program will print the alphabetically-first word in that batch and the length of the longest word in that batch (see format below).

When the entry is the special all-uppercase word "`DONE`", the statistics for the current partial batch (if any) are printed, and then the program ends. Invalid words (too long or not all-lowercase) get an immediate response of "`error`" and the program terminates.

Example

```
some
simple
testing
of
alphabetic
First: alphabetic
Longest: 10
various
short
words
DONE
First: short
Longest: 7
```

Internal and other requirements

All string manipulation should be protected from buffer overruns and malicious input. Simply making the buffers a bit bigger, e.g. size 20, is not sufficient.

Your words should be stored as a 2D array with appropriate bounds for the task.

Validation should be done, at least in part, by calling a function you write, which returns a value (and doesn't print anything itself).

Stats computation should be done by calling a function you write. The function should not do any of the printing itself. (Think carefully about what its parameters need to be.)

You *may* write functions other than the two just mentioned, if you find them useful to your design.

The headers of all functions you write other than `main` should be provided in a separate corresponding `.h` file, and the function definitions should be in a .c file separate from the one main is in.

Your handin should include a readme documenting how to compile and run the program (exact instructions on what to type, or better, a copy-and-pasteable command); and test cases and instructions for how to use them. You may use a makefile to control the build but should still include a readme with instructions. You should not rely on the `compile` command, but use `gcc` directly.

Note that a program that does not compile will get few or zero points (even if there's a lot of code that seems close), while a program that compiles and does only a few things will be eligible for much partial credit. Use test cases to show me what works.

Duedate and handin

The lab is due Friday at 4pm.

Hand it in using the handin script:

```
handin cmsc242 lab1 dir_of_stuff/
```

Instead of `dir_of_stuff/` you can use `.` (period) to refer to the current directory or `*` to refer to all files in the current directory, or you can name the files explicitly (but it's easier to just hand in the whole directory, right?).

Note that later handins effectively overwrite earlier ones. If you change something in one file, you can re-handin but make sure you hand in *all* the files, not just the changed one.

Rubric

RUBRIC

README and other files (3):
1 compile instructions
1 testing files or instructions
1 func headers in .h, func def'ns in separate .c
Main file: setup and declarations (6):
1 #incl stdio and/or other appropriate
1 int main()
1 declare int and char[]
1 declare 2d array
1 ...bounds are for word list length and word length
1 ...word length bound is correct
Main file: I/O (5):
1 scanf one word into valid buffer
1 ... limited to buffer length
1 printf anything
1 ... a string variable
1 ... a number variable
Main file: loop algorithm (7½)
1 loop w/ bound or reset at 5 lines
1 ... and keep going after 5 lines, w/o infinite
1 read 5 and print stats after 5
1 check if word is DONE
1 break if DONE
1 ... but still print stats
½ ... ... but don't print stats if DONE is 6th line (no stats to print)
1 ... and don't include DONE in stats
Main file: error and exit (4½):
1 error and exit immediately under some cond
1 ... if includes invalid characters
1 ... if too long
1 ... ... compare actual string length to constant (could be in fn)
½ ... ... with error at correct length
Validation function (4):
1 header works for validating lowercase and/or length
1 loop all: for loop, access element, stop at n or `\0`
1 correctly id lowercase
1 ret false if non-lower found, true if fallthru OR equiv
stats function: headers (3):
1 param is 2D array
1 param is number of vals in array to check
1 all stats in one fn with two out parameters or equiv
stats function: algorithm (7):
1 loop all: for loop, access element, stop at n or 5
1 bounds check: start at 0, stop at n, where $n \leq 5$
1 init both accumulators before loop
1 if longer than longest, update
1 if earlier than earliest, update
1 ... with valid alpha comparison (not just first letter)
1 update out params or return both (print not sufficient)