Programming Project 1

EE 312

General: This first big project will give you a chance to dust off all the cobwebs on your programming skills. The project requires that you understand characters and strings, arrays, functions, loops and conditionals (all prerequisite knowledge). However, just because this is technically "a review", don't assume it's easy! You'll write a very simple spell checker. You must produce the output EXACTLY as specified in this document.

There is some flexibility permitted in your solution to encourage you to adopt your own ideas for an algorithm. The limits to that flexibility are explained below. Any extra output, or output that is inconsistent with these requirements will result in lost points.

Your Mission: Edit the file "Project1.cpp". You must implement the function *printSuperStrings*. You may find it useful to write several other functions as well. In fact, I encourage you to avoid writing the whole project as one big (ugly) function. You will find big (ugly) functions are very difficult to debug, and we will find it very difficult to help you. For this assignment, you should keep all your functions in one file -- Project1.cpp. They should all be C, and you should not use C++-only features.

For the purpose of this assignment, a superstring *S* of a string *s* is a char array such that if we remove 0 or more chars from *S*, we get *s*.

Examples:

superstrings of *hello*: *hello*, *xhxexlxlox*, *helolo*

non-superstrings of *hello*: *helo*, *helon*, *sandwich*

S and *s* consist of only letters, upper- and lower-case. A word is a sequence of one or more letters with no intervening whitespace. Whitespace characters can occur anywhere -- at the start, just before the final null termination, or anywhere between words. Characters are case-sensitive.

There is only one stage on this project, writing the *printSuperStrings* routine. The *printSuperStrings* function has two parameters. The first parameter (*strings*[]) is a pointer to an array of characters. The contents of this array are a list of words that you need to spell check. Words in *strings* are separated by spaces, tabs, and possibly newlines. The end of the article is marked with the normal '\0' (marking the end of a C-string). The second parameter of the function (superstrings[]) is a pointer to another array of chars similar to *strings*. Your function must print every word in *superstrings* that is a superstring of every word in *strings*, in order.

Finding words is easy. Just look for a letter, thus marking the beginning, and keep scanning for consecutive letters. When you run out of letters, that marks the end. For each word in *strings*, you must go through *superstrings* to print all the superstrings of that word, each accompanied by a newline. Do not print empty lines, except perhaps for the very last line in your output.

You can print the words in several ways, depending on how you store the words in memory. If the word is terminated with a 0 like a regular string, then you can use printf:

> printf("%s", word);

If your strings are not terminated with the zero, then you will need to print them one letter at a time (using a loop). You can print a single letter in one of two ways:

> printf("%c", letter);

or

> putchar(letter);

When you're debugging your program, you will want to make sure that all the characters you have printed show up on the screen. That may sound silly, but it is a real problem. The computer does not always put characters on the screen, even if you've used *printf* or *putchar* to print them! If you want to make sure characters have been forced to the screen, print a new line ("\n") or use fflush(stdout).

Bounds on input: *strings* and *superstrings* may each have up to 5000 words. Each word is at most 19 chars long, not including any null termination or whitespace. Your program should run in < 5 seconds on mario or your laptop.

Output Requirements: Your function **must only print the superstring words from *superstrings***. Each word that you print must be printed on a line by itself (i.e., you must print a '\n' after each word of output). You may print words in the same case as they appear in *superstrings*. Please keep in mind that we will test your program with different *strings* and different *superstrings* than have been provided to you. Your program should work with any *strings* or *superstrings* that satisfies the requirements listed above. Not all *strings* start with a letter or end with a letter. Not all *strings* have at least one word. Not all *superstrings* have at least one word.

Guidance/Advice: We strongly recommend that you develop the project in phases. Your first goal should be to produce a working program that correctly reads (and prints) all the words from *strings*. Don't just sketch pseudo-code of this phase -- really write it, compile it, run it and debug it. Once you can discover and print each word, proceed to the next phase of development where you begin consulting *superstrings* for each word and printing the superstrings only. It is a truism for EE312, if you can make the time to write each program twice (even if you don't build all the functionality into your first version of the program) you will learn more and learn more quickly than if you try to do the entire project at once.

Project 1 requires no knowledge of pointers. If you are very comfortable with pointers, feel free to use them. If you are not comfortable, then don't use them. Don't use variable length arrays in your solution. Think about what size you want to make your arrays. While normally global variables are discouraged in good program style, for this project, you will probably need to use them. Still, minimize their use and pass parameters as inputs to your functions, and let your functions have return values.

Follow the instructions in the Assignments link on the course home page to test your code. In particular, it doesn't matter how small your error is, but if it doesn't run with our grading script, and you never ran and passed the grading script we gave you, you won't get any credit. Our strict policy is designed to familiarize you with industry standards of submitted code.

We are also expecting combination of good program design (good coding style, including comments, indentation and variable/function names as well as a concise implementation of the required functionality), and a robust implementation (your program works correctly under all correct sets of input).

Good luck!

Sample I/O:

string1[]: `hi be'\0'`

superstrings1[]: `hi hibe \nbet box'\0'`

printed output of `printSuperStrings(string1, superstrings1)`

```
hi

hibe

hibe

bet
```

Submission: You must copy and paste *header.c* at the top of your submission file. Then submit *Project1.cpp* only. If Canvas renames your file because of multiple submissions, that's OK.

FAQ

Q: What files do I copy over to mario.ece?

A: For the 'mario phase' copy over Project1.cpp, Makefile, the input.txt files -- everything in the starter_files directory. Don't copy stuff from your CLion folder except your source code. Don't copy CLion's Makefile. For the grading script phase, copy over Project1.cpp, and the sample_grading_script.zip file only. Test both phases in different folders on mario.

Q: I am getting an error related to C++11.

Q: On mario, you should type in the command 'module load gcc' each time you log in.

A: The Makefile is giving strange errors!

A: First, if you are on a 64 bit computer, go into the Makefile and remove the -m32 flag. Just search for it and remove all instances of it.

If it still doesn't work, try using

```
> touch Project1.cpp
```

This updates the timestamp on your file.

Then try

```
> make clean
```

```
> make
```

This rebuilds the project from scratch.

Q: How can I test and debug my project?

A: You can test it by typing in

```
> make test
```

in your terminal. This runs the created executable with the given input file.

Make sure to write your own tests, though.

Debugging is a little trickier. You can use your IDE's debugger, or text-based GDB, or for now you can use printf to output variables you want to see to the terminal.

Q: Are we allowed to call functions we write in the same file?

A: Please do! Using one function would be harder to read. Just know that we won't call them from our main while testing.

Q: What do I submit?

A: Submit your Project1.cpp file. Make sure that all your functions are in this one file.

Q: May I use the C++ standard library?

A: No. You may use the standard C library, but you don't have to.

Q: What is a segmentation fault?

A: A segmentation fault means your program tried to access an address that it couldn't get to. This usually means you tried to read past the end of a string, but it could be any invalid memory access.

Note that the files look *something like this* to your program:

superstrings:

word1\n word2 word3\n....lastword \0

strings:

word1 \n word2 word3\n...lastword \n \0

CHECKLIST -- Did you remember to:

- Re-read the requirements after you finished your program to ensure that you meet all of them?
- Include the contents of the header file (filled out) at the top of your submitted file?
- Make sure that your program passes all our testcases?
- Make up your own testcases?
- Check that your solution works on mario.ece using our grading script? This is not optional.
- Upload your solution to Canvas?
- Download your uploaded solution into a fresh directory and re-run all testcases on mario and CLion? This part is not optional.

Last modified: June 3, 2020, 7:18 PM