COMS W3157 Advanced Programming, Lab #1

Read this assignment carefully and follow the instructions EXACTLY.

Retrieval and Submission

Refer to the lab retrieval and submission instructions, which outlines the only way to submit your lab assignments. Please do not email your submission to the teaching staff.

Please write a README.txt in the top level directory. At a minimum, README.txt must contain the following info:

- your name
- your UNI
- lab assignment number
- description of your solution

The description can be as short as "My code works exactly as specified in the lab." You may also want to include anything else you would like to communicate to the grader such as extra functionalities you implemented or how you tried to fix your non-working code.

Part 1: Academic Honesty Quiz

To ensure an effective and fair learning environment, it is essential for every student to know what constitutes academic dishonesty and uphold the class/university policy. The following questions are intended to help each student reinforce a thorough understanding of that policy. These scenarios are inspired by actual academic dishonesty cases that have been encountered in the past. Please refer to COMS W3157's Academic Honesty Policy and the CS Department's Academic Honesty Policy to help answer these questions.

In this lab's skeleton code, you should find a file at part1/honesty-quiz.txt. To complete this quiz, you should write your answers there, commit your changes, and submit.

For each of the following scenarios, write YES or NO as to whether they constitute cheating and justify your response.

[1] You hire a tutor to assist you with labs. You find that your solution has lots of errors and you feel overwhelmed and unsure where to begin. The tutor addresses this by asking leading questions and pointing you to areas in the code that may have caused these errors.

[2] You're struggling to fix an error and know that a friend enrolled in the class has finished the lab. You reach out to explain your error and they send you a screenshot of the relevant portion of their code. You look at it to understand your error but do not copy any lines.

[3] At the beginning of the semester you are concerned that you might not be able to complete all the labs on your own. For peace of mind you ask a friend from a previous semester for all solutions. You

wind up not needing them and never consult them once.

[4] After lecture, you're eating dinner with a friend and discussing that week's lab assignment. Determine whether each of the following scenarios is cheating:

  [a] Your friend finds a concept from lecture unclear and asks you to explain.
  [b] Your friend asks you to explain a small conceptual element of their solution.
  [c] Your friend asks you what you think the lab assignment is trying to teach you.

[5] You are stuck on a lab in which you have to build a server. You know that the lab specifications want you to figure out how to take on multiple requests, but you're not sure exactly how to order your code so that it works. After a few Google searches, you read a post that lays out some code that would be a solution to the lab. You make sure to not use any of the code in your submission.

[6] You are working on the lab. You try to compile your code, but you get an error. You paste the error into Google and find a page that explains the nature of the error. You understand the error, go back to your code and apply what you have learned to fix the error.

[7] You're working on the lab but having difficulty applying concepts learned in class. You search for a lab solution from a previous semester, but only copy in small parts at a time to test the code, look up man pages for functions used and try to fully understand the solution.

Part 2: Hello Wordle!

In part2/ of the skeleton code, you will find a partial implementation of Wordle, a word guessing game that became popular in 2021. Your assignment is to complete this implementation, adapting existing C code from the skeleton code.

You should already be able to build and run the skeleton code as is. To build it, make sure you are in the same directory as the included Makefile, and run:

```
make
```

This will produce an executable named wordle, which you can run with following command:

```
./wordle
```

Note that the leading ./ says that you mean to execute a file in the current working directory.

When you run wordle without arguments (like shown above), it will complain and show the help menu you get if you run it with the -h or --help flag:

```
./wordle --help
```

The help menu documents the arguments that wordle must be run with, to indicate what file it should use and what it should do with that word list. For example, to play the wordle game, picking the fifth word (position 4) from the word list

words/short-list as the answer:

```
./wordle --file words/short-list --number 4 play
```

Equivalently, you can run the program with shortened versions of each flag:

```
./wordle -f words/short-list -n 4 play
```

The answer is "bread"; try putting in some correct and incorrect guesses. You can quit the game by pressing Ctrl-D.

You can also tell wordle to show you the selected word rather than playing the guessing game, using the "show" mode:

```
./wordle --file words/short-list --number 4 show
```

If you don't specify the --number argument wordle will pick a random word from the word list:

```
./wordle --file words/short-list show
```

Testing wordle

The skeleton code also includes an automated test suite, which you can run using the following command:

```
make test
```

The skeleton code should pass some test cases but fail others. You should aim to get all tests passing.

Code organization

The provided C code is organized as follows:

- words.c: This file implements all functionality associated with reading words from the word file.
- game.c: This file implements the guessing loop, which prompts the user for a guess, checks the guess against the answer, and provides a hint for the next guess.
- wordle-main.c: This file contains the "entry point" of the game program. When you compile and run the program, execution will start with the main() function in this file. This file is in charge of parsing command-line arguments, and calling the appropriate functions from words.c and game.c.
- words.h and game.h: These header files contain declarations for the functions implemented in words.c and game.c.

For your assignment, you should only submit modifications to words.c and game.c. You may modify other files (for example, to insert print statements for debugging), but you should NOT submit those changes.

In addition to the C code, the skeleton code also includes the additional supplementary files:

- Makefile: This file contains instructions for building the C program. It declares all the dependencies for each build target, and provides recipes for how to build each target from its dependencies.

- words/: The words/ directory contains some lists of words for you to test with. words/short-list contains 20 hand-picked 5-letter words, while words/common1000 contains the 1000 most common English words.

- tests/ and run-tests.sh: The tests/ directory contains a small suite of test cases and expected outcomes, while the run-tests.sh script runs these tests. You should not need to run run-tests.sh on its own; instead, use make test.

- .gitignore: This file tells Git not to suggest tracking build artifacts like .o files and the wordle executable.

- README.txt: Make sure you fill in the required information!

You should not modify any of these supplementary files except the the README.txt. You are free to add additional word lists and test cases, though if you do so, you should document these in your README.txt.

Implementing wordle

There are a few problems with the given skeleton code:

- The given implemention does not support selecting words beyond the tenth. Confirm this using some --number greater than 10.

- In addition to the "play" and "show" modes, there is also a "dump" mode that is supposed to print out all of the valid words it found in the word list. However, the skeleton code's implementation of dump does not do anything.

- The guessing game does not give very helpful hints; in fact, the hints it gives are not Wordle hints at all!

Your goal is to fix these issues.

To fix the first issue, implement count_words() in words.c. This function should count the number of valid words in the word list at the given path. You should look at home read_words() is implemented and adapt your solution from there.

Next, implement print_words() in words.c. This function should print all the valid words in the word list at the given path. Once again, you should read and adapt read_words().

Finally, implement give_hint() in game.c. You must give a Wordle-style hint: for each character in the guessed word, it print '+' if it matches the character at the same position in the answer; it should print '-' if it does not match, but matches a character at a different position in the answer; it should print '_' if the guessed character is absent in the entire answer. You should adapt your answer from give_stats().