Dylan Dunsheath HW #2 Description

CS-288-006

**Problem #1**:

This problem was very straightforward and wasn’t as involved as the other problems, which required more thought to complete it. I began my checking if an argument was passed (doesn’t matter if more than one argument was passed, we would just use the first argument) and if there was no argument passed, it would return an error, show the format that’s required and exit the script. If there are one or more arguments, use the first and assign that to filename. I then implemented various conditions to validate the input; checking if it’s a directory, if the file exists or not, and then did a method of extracting the extension and verifying if the file ends in ‘c’ (or .c rather). If this is the case, I used grep -c of both “scanf” and “printf” in the given file to count these pattern occurrences. I also used wc -l to get the total lines of the file for the percentage of these two keywords (and how often they appear in the c file). Using scale=2 allowed me to approximate to two decimal places and by using the given formula in math along with scale, get the percentage. If there is an error with the extension however, I output the error and exit the script.

**Problem 2:**

This problem wasn’t too bad and was somewhat straightforward, not as much as the previous problem. In this script, I made two functions, ‘extract\_emails’, which gets the passed file from traverse\_directory and grabs only the valid content (that are emails) in the txt file, which ranges from a-z, A-Z, 0-9, -, \_, and ‘.’ along with @ and then the domain.com portion. Along with the regex, the | pipes the input/content and along with sorting the emails, removes duplicates for each file.

The traverse\_directory function was also used in HW #1, and simply traverses directories and subdirectories for all txt files to extract the content (emails) within.

The given script collects the input via command-line arguments. Here, we get a directory. We validate if the user inputted an argument, ensured it was a directory that existed, and if there were no emails, we let the user know. The emails are saved in all\_emails after traversing through the directory and subdirectories. We then we do a sort of the emails we found to remove duplicates (not just from the same file which may have multiple, but also other files in general). We then said where these emails can be found.

**Problem 3:**

This one was kind of difficult compared to the previous two problems but ending up solving it wasn’t as bad as I expected. Given that I chose to pass the file (as a txt file) to the script via the command line, I checked if the user entered any number of arguments, if none were passed, I displayed the error and exit the script. Then, I passed the file to a variable, input\_file to be used for the rest of the script. I checked if it was a directory (when it should be a file) and if the file existed. Along with that, I verified via the same method as problem #1 of getting the extension to verify if it was a txt file and alerted if it isn’t. It was also important to verify that content existed in this file (otherwise there’s no point in continuing the script). I initialized a variable to hold a string of 2024 for age calculations and then a regex of the pattern ‘firstName lastName, DOB [YYYY-MM-DD], City, and Country.’ Using a while loop, I read through the file and if it matched the given pattern (especially for cities with two or more words). I used awk -F which makes a field separator based on some delimiter. I did this for both the name and dob (since we needed to print the name and then the age). I called a function which passed the dob variable into it and then initialized the year the person was born via the date command and then subtracted 2024 – year of birth. This passed the echo statement (age) into a variable called age outside the function and appends ‘[fullName]’ and ‘dob’ to a temp file for all that meet the criteria of a city with 2+. After the loop, it sorts the content, using the comma as the delimiter and uses the ‘age’ key/field as the method of sorting in reverse numerical order (nr). The content in the text file is what’s sorted. I then pipe so it outputs the name and how old they are with awk.