Student ID: 201535181
User ID: sgronuma

Problem 1086:
To solve the first part of this problem, I used a step-wise approach with two variables (the reoccuringChar variable and the artFileChar variable). While reading in each character from the input, the previous character would be saved in reoccuringChar variable and the current character would be saved in artFileChar. A separate character counter variable would increment if the previous character is equal to the current character. For the compression task, if the previous character is not equal to the current character and the character count is greater than 1, then two of the previous character and the character count would be printed. On the other hand if the previous character is not equal to the current character and the character count is equal to 1, then the individual previous character will be printed (there is an additional else statement that takes the same conditions to execute if the new line character is encountered)

To solve the Encrypting task, I implemented the similar steps as I did for the compression task with a few differences to reverse the effects of compression. If the character count was greater than 1, then I would check if the value is a digit. If the character is a digit I would convert it to an int and multiply it by ten then add a the next value if its also an int (to represent the whole number instead of individual numbers), or else I would print the previous character by the number of times of the converted int value (called multiplyValue). If the character count is not greater than 1, then the character count is incremented, the current character is printed and then the previous character gets assigned to the current character.

No additional material was used in these tasks, only the lecture materials. However, I used an additional library (<ctype.h>) that wasn't taught in lectures, in order to use the "isdigit" function to identify which characters were digits.

Problem 1090:
To solve this problem, I created a malloc called allWordsPtr that would be able to contain all possible strings (character pointers) as possible. Then, there is a while loop that reads in each line to the inputLine variable, and multiple things are done in that same loop. There's another while loop that separates the string (character array) into tokens divided based on the space, comma or period characters, and creates a malloc that contains characters (that would be used to store only alpha characters, that would be later be stored in allWordsPtr). After every possible word is extracted and stored in the allWordsPtr variable, two additional mallocs are created that would be used to store all individual strings and all individual counts (called individWrdsPtr and individWrdsCount, respectively), and would both be connected based on an index called individWrdsIdx.

In a separate for-loop, the array containing all possible words (allWordsPtr) would be iterated, then there would be three separate for-loops nested on it. The first for-loop is to check if the word gotten from all possible words in the text (allWordsPtr) was already extracted (if it wasn't extracted, then it could be added to individWrdsPtr array and the other for-loops can be executed). The second for-loop is to count the amount of times that individual word appears in the text, so that it can be added on to the individWrdsCount array later on. The third for-loop (that contains a while-loop in itself) is used to sort the individWrdsPtr (which would also sort the individWrdsCount) in alphabetical order.

A for-loop is then used to print out the sorted words and their respective counts, and finally the mallocs that I created are freed.

Just like in the last problem, I used <ctype.h> library, that wasn't taught in lectures, to use the "isalpha" function to identify which characters were alphabets. However, unlike the last problem, I used additional material to help with the understanding and formation of tokens (from https://www.geeksforgeeks.org/strtok-strtok_r-functions-c-examples/) and how to compare strings without the strcmp function in <string.h> library (from https://overiq.com/c-programming-101/the-strcmp-function-in-c/).