COMP281 Assignment 2 Report

Name: Jiashun Lu

Problem 1086

This problem needs compressing images composed of ASCII art files or expanding a string of compressed code into an ASCII art file.

First, I use the switch case to determine whether the first character in the file is C or E. Depending on the first character, I can perform different operations.

At the suggestion of Professor Phil, I decided to use scanf to read each character separately until I got to EOF. The first step is to determine whether the first character read equals the second character. When matching, define a variable count and add him by one. When the count is greater than or equal to 2, output in the compressed format and reset count, otherwise output the original characters directly, assign the second character to the first, continue reading the next character and compare it with the first character. After each comparison, the compressed value is output and the count is reset to 1. Since there is no way to compare the last character with the next one, the loop ends with another determination of the last count value and the corresponding normal or compressed character is output. After each comparison, the compressed value is output, and the count is reset to 1. Since there is no way to compare the last character with the next one, the loop ends with another determination of the previous count value, and the corresponding normal or compressed character is output. The function to expand the characters also reads the first character to compare with the second character first because if there are more than two characters, they must be abbreviated to a combination of 2 identical characters plus the number plus an asterisk. So first, compare the first and second bits to see if they are the same. Print the output if the first and second digits are not the same. If the first and second digits are the same, loop through the next characters until you get to the asterisk, which means the abbreviation has been read. Convert the characters read before reading the asterisk into numbers to get the number of repetitions needed, and loop through the previously repeated strings to print these numbers to complete the string expansion. After reading the first string, you need to write an extra getchar() to remove the extra carriage return because the next characters start on the second line. The knowledge of c language used in this question includes escaping numbers from characters to computable numbers, EOF for end-of-text file, and multi-conditional judgement by switch case.

Resources used 1: https://blog.csdn.net/weixin_44551646/article/details/98076863

Resources used 2: COMP281 Lecture 05.pdf

Problem 1090

For this problem, because the teacher prompted the use of structures plus dynamic memory management to do it, the constants and structures are declared at the beginning, each word is up to 30 characters long, and up to 300 unique words, so

two constants are defined to hold the two numbers. Define a structure containing an array of characters and the total number of occurrences of the word. The program starts by allocating 300 contiguous spaces of the size of each word in dynamic memory, and the function returns a pointer to the starting address of the allocation, which points to the structure. Then use the scanf loop to get the input word, split by space until EOF stops. In each loop, first, get the length of the current word, then first determine if the end of the word is punctuated and if so remove it, then determine if there is punctuation in the word and if so remove it. The change in length is recorded while the punctuation is removed and the current word is made up of all lowercase letters by the string_to_lowercase function. Next go through the for loop and make a judgement, if it is the first entry it means it is unique, record the word and make the unique judgement 0 so it will not be the following loop, if it is unique, increase the index value and copy it to word. Next, the sort_by_alphabet function is called. The built-in function compares the size of the characters before and after the two words, with the smaller ones going to the front and the larger ones climbing back, swapping their positions by bubble sorting. Just output the sorted structures in order and free the memory space at the end. This question uses c's knowledge of dynamically allocating memory space, and also includes defining structures, using typedef to define structures, comparing the contents of characters via strcmp, and copying the contents of characters via strcpy.

Resources used 1: https://www.cnblogs.com/20201212ycy/p/14904663.html

Resources used 2: COMP281 Lecture 09.pdf

Resources used 3: COMP281 Lecture 10.pdf

Resources used 4: https://www.geeksforgeeks.org/strlwr-function-in-c/: