Readme

In this assignment we were tasked with creating a compression algorithm which will compress long lines of similar characters through the use of threads and processes. The code implements the compressions in various ways, RLE(Run Length Encoding) encoding, LOLS(Length of Long Sequence Compression) encoding, and Multipart LOLS encoding. RLE encoding will go through a string and find repeated characters and replace them with a single character and the number of times it was repeated. LOLS encoding will do the same as RLE encoding except it will omit individual characters and pairs of characters. Multipart LOLS is LOLS encoding, but the end result is split into a set number of different documents with the names of LOLS0, LOLS1, LOLS2, etc.

The implementation through threads of the LOLS encoding is done in the compressT_LOLS.c file. First the code counts the number characters in the file. Then uses a struct which contains filename, thread number and the start and stop indexes. The code then creates various threads which iterate through the string and the populate the values of the structs. Then the code places the values of the structs in the specified files.

The implementation of the LOLS encoding through process is done in the files compressR_LOLS.c and compressRworker_LOLS.c. The file first calculates the number of characters in the file and then creates the number of processes based on the number of files that the user wants the file to split into. The processes then iterate through different parts of the file which is determined by the the size of the file and the number of process that are needed. Once the process iterates through the file, it will create a file with the above mentioned naming convention, and place the compressed portion of the code in the file.

If a file that doesn't exist in the same repository as the code, the code will attempt to open the file, output an error message, and then exit the program. If the input file for the program is not in the form .txt, the code will attempt to open the fill, provide an error statement and exit the program. With a blank input file, the tokenizer will immediately hit the end of the file and cause the program to print an error and exit. If the input file has fewer characters than the number of files requested, the program will print an error message and exit the program.