README

Design

Tokenizer

fileCompressor - the file compressor takes arguments in the format ./fileCompressor <flag> <path/file> <codebook>

The flags
-b for building the codebook
-c for compressing the file contents using the codebook
-d for decompressing the compressed contents using the codebook
-R for a recursive call on any of the above flags in the syntax
./fileCompress -R <flag> <path/file> <codebook>

The fileCompressor reads the path/file by utilizing lseek() to open the file and convert the contents into a string (char*). Depending on the flag, the contents are can be use to build the codebook.

Build - the build flag causes the funtion 'stuff' to tokenize the input into an array, and begin to parse the elements of the array, each unique 'word' is given a node in a priority queue, and each time a word is seen again, that nodes frequency is incrimented. When it reaches the end of the array, the queue is sorted via huffman tree, the two nodes with the lowest frequency are removed, and placed as 'leaves' on a 'branch' (lowest going left, second lowest going right) until the priority queue only contains one element, which will be a 'branch' Once this is finished, the program searches from the 'root' of the tree, creating a code for every word which is the binary sequence required to navigate to that word (zero going left, one going right). When this is finished, a list of all words and their respective codes is printed to a file

Compress - The compress flag allows the fuction 'stuff' to perfom huffman code compression based on the codebook generated before. each 'word' is replaed with its respective code, leaving one long binary string as its output.

Decompress - the decompress flag allows stuff to reverse engineer the work done by 'compress', using the same binary string it outputted and comparing it character by character to the list of codes, until a match is found. when a match is found, that 'word' is placed in a temporary string. This repeats until the binary string is empty, and all zeroes and ones are properly translated into the output string.

Each of those functions return a string that is ready to be written onto a file. Using the library call write(), it simply writes that string into a targeted file.

Recursive - the recursive call takes a path as an argument instead of a specific file. Given the directory, it accesses all the valid files and concatenates it into a single string. To build, it builds a codebook based on the occuring words of all the files in the directory and subdirectories. To compress, it uses the prepared codebook and compresses each valid file into a binary string to write to their respective <file>.hczThen to decompress, the function parses through the directory searching for .hcz files, decompresses them, trims the .hcz from the file to get to the original file and writes it there.