READ ME

Lara Martin, Elana Braff, and Brian Battell

Our program uses the tokenizer provided in the assignment description as well as sorted-list from assignment 2. However, the tokenizer is tweaked to take in files. The sorted-list is reformatted so that if the filename is already found in the word's list, we increment the counter and reinsert the file to fit in descending order by number of occurrences. We then created an indexer.c to retrieve our files from the given directory and call the sorted-list on our files. It inserts each word at a time from the files. It recursively walks through the directory to determine if the directory is valid and it ignores and hidden and back up directories and files. If we find a file we tokenize it and insert it into a list and then output it into a file. All .c files have their respective .h files as well.

The data structure that we used is a linked-list of linked-lists. We have a linked list of words that we malloc for each token in the list and then frees them after the list is through being used. (Each alphabetized word in our main linked list has its own sorted linked list.) The worst case running time of the data structure is O(n^2*m^2) since it has to iterate through a linked list at O(n) and insert n nodes --> O(n^2); if it were to insert always at the end of the list. Then it creates another linked list for each of the n nodes and iterates/sorts through that list at O(m^2).

It has worst case O(nm) space since each word (n) can be in every file (m).

NOTE: Files starting with . or .. or files ending with ~ are only read when specified.
Otherwise, they are ignored (when the directories they are in are named).