Mark Conley && Michael Newman
CS 214: Systems Programming
Programming Assignment 3: Indexer
readme.pdf

Our plan was to recursively traverse the directory and if a file is found we would call our function indexFile, which opens the file and indexes the words.

Handling of files for each word
> Described in handling of words from files
> We add to head always for new words because we know we will only open and index each file once
> this allows us constant time to add to the count of a word because the file will either be the first in the list or not in the list at all (will be added to head)
> This is also done because it is much more desirable to have ~O(1) time to add the counts to each word in the file, and then sorting before printing, than it is to sort as we go and sacrifice the O(1) add time to not have to sort at the end (especially becuase often the same word will occur multiple times in the same file, so the lookups would become costly)
> this is also preferable over a hash table because either way we would have to sort at the end, and the hash table has more overhead
- also, the hash table has the potential of non-constant time if there is any chaining

Handling of words from files
> Read in words from files and tokenize them
> Send each word off to function to check for occurance in the hashtable of words
- if doesn't exits, create a new Token object for the word, then a new file node for the linked list of files and counts
- if exists, check head of file nodes LL
* if head is null or is not the current file, create a new head with the current file and count = 1
* if head is the file we are currently in, add 1 to the count for that file
> once all words are read from all files we are set to print

Handling of printing (takes in a delete flag to print and delete)
> First sort the hashtable by the key (the word)
> next we will iterate through the hash table using HASH_ITER
> At each word we find we will print it and look at the file LL
- First sort the file LL by the count (greatest to least)
- Then print the list, deleting/ freeing each node once it is printed if the delete flag is set
> Once each file is printed for the word we delete/ free it if the delete flag is set

> Finally all words are printed and removed/ freed, we can return

This assignment my partner and I took a different approach to doing on this project. We very carefully planned out the design of our indexer and decided to split the parts up. Since we were splitting the parts of the assignment up and testing it individually we had to make sure our plan was sound. I worked on the directory traversing, while Michael worked on indexing the words and writing them to a file. I think this was a sound approach.