readme.pdf for Indexer

Our indexer utilizes a modified version of tokenizer from Asst0 and the sorted list from Asst1. Tokenizer was modified to work with a file and sorted list was not modified.

The Index is created using a sorted list of sorted lists. The outer list contains the token list and is a list of WordEnt_ structs.

WordEnt_ structs are comprised of 2 fields: Word, RecordList.

> The word field is a simple pointer to a char array that represents the word token found when tokenizing a file in the directory.
> The RecordList is itself another sorted list that will maintain RecordEnt_ structs indepdently of where in the sorted list the WordEnt_ is found. In this way we can easily manage sorting the outer list by word order and the inner lists by record order
>
> WordEnt_ struct's are ordered in the Index (which is a sorted list) in increasing order by their Word field.

Each WordEnt_ struct has a RecordList that is comprised of RecordEnt_ structs and is organized in the following way:

RecordList sorted lists are lists of RecordEnt_ structs that contain two fields: File, Occurences

> The file field is a simple char pointer to the filename that the word token was found. It is important to note that given this implementation if there are two or more files in separate directories with the same filename, the occurrences will continue to accrue for that filename and a separate record will not be created. This is as per the assignment instructions wherein word tokens found in different directories but with equivalent filenames are indexed together in a single record. Relative path names are not considered.
>
> For example: Indexing equivalent word tokens from the file /adir/boo.txt and /adir/folder/boo.txt will be stored in a single record with the filename boo.txt.
>
> The Occurences field simply keeps track of how many times the token (represented in the WordEnt_ struct appeared in the file.
>
> RecordEnt_ structs are sorted first by occurences (greatest to least) then (if occurences are equal) by their file names in alphabetical order
>
> Since each WordEnt_ struct contains a RecordList, we can mange RecordEnt_' struct positions independently of their greater position in the Index itself
>
> In this way we keep the data structure of our index (sorted list of sorted lists) simple, concise, and easy to navigate.

The Indexer is broken into two main parts: CreateIndex and WriteIndexToFile

CreateIndex: Searches the directory recursively and tokenizes each file, inserting and maintaing the index as it continues to search each file.

WriteIndexToFile takes an Index data structure, traverses it, and writes it out to a specified output file
while it is traversing

In this way, by separating these two functions, we can maintain the Index data structure in memory, without modifying it, and write it out to multiple files (if need be).
Also: the index only needs to be created once (for each directory indexed) and can be rewritten to as many files as need be during program execution.