Our program uses the tokenizer from assignment 1, and the sortest-list from assignment 2, and the indexer from assignment 3. The tokenzier is tweaked however, to take in files instead of strings. It also uses all characters that aren't letters or numbers as tokens instead of tokens the user can input. The indexer.c reads in a file or directory and calls the sorted-list, inserting 1 word at a time from the files. If the file is valid, it tokenizes it, inserts it into a list, then outputs into a file.

We also created a index_parser.c, which parses and loads an indexer into memory, given the file path of the indexer file, and a util.c, which reads the contents of a file, given the path, and returns it as a string. Returns null if there was an error.

The program reads in a file or directory, and the user can either type "sa", which checks to see which files contain ALL the search keywords, "so", which checks which files contain EITHER of the search keywords, and "q", which exits the program. It uses the indexer from assignment 3 to index the words, along with the sorted_list to sort them, for easy searching.

Analysis:

Writing the file into memory: Call the number of words in the file and check to ee if each word is equal to <list> or </list>. If they don't insert into memory.
O(f)

Logical OR on the lists of all the terms a user inputs and return the resulting list:
Number of words = n. Number of files = k.
Runtime is O(nk) because we have to search through each word and each list of files to figure out what to insert in the algorithm.

Logical AND on terms a user inputs and output the resulting list:
Identical to logical OR (O(nk))