Overview:

The indexer walks through a directory or individual file and tokenize them, discriminating against duplicate string tokens. Using linked lists, we designed the indexer to create a single linked list per input file, accounting for duplicate strings via structs via a count inside each node. We scan the file before building our linked list for the side of the file and after the linked list is populated, send our data to our xml output process.

directorywalk.c

We used the dirent.h library in a recursive function..

listdir(const char name, int indent)*

Time Complexity: O(n), where n is the number of directories in the path. listdir() is called recursively a maximum of n times before it reaches the base case of no directory.

Space Complexity: O(n), with a recursion depth of a maximum n directories.

argc[2] passes the directory or filename as a char* to the listdir function, where it passes conditions that check if it is a leaf node(file name) or a directory. In the case of a directory, the directory is passed into a while loop that uses recursion to check if the input is directory or file. In the case of a directory, listdir is called recursively with a dissociate path name. In the case of a file name, it's path is copied, opened as a FILE type and passed into the buildIndex function in buildindex.c.

buildindex.c

We used the stdio.h library to open files sent from directorywalk.c..

*int get_file_size(FILE *fp)*

int search(Node_t nodes, char word, int len)*

*int buildIndex(FILE *file, char* filename)*

Time complexity: O(n*m) where n is the number of tokens in the file and m is the number of lines in the file due to the nested while loops.

Space Complexity: O(n*m) where n is the size of the file and m is the number of files being tokenized. We use get_file_size to take in a FILE pointer and return the number of bits we need to allocate to tokenize the file, giving us our n. This is called m times as walkdirectory.c sends a file to buildIndex() in buildindex.c.

File names (char*) and file paths (FILE *) are sent from directorywalk to be tokenized into a linked list, accounting for tokens (char* token) and the amount of times they occur (int count). We use the function get_file_size() to get the size of the file and search() in order to find our duplicate tokens. The search function also is responsible for adding to a node's count in the case of a duplicate token. The linked list is then iterated through, passing the data to xmlwrite.c to build our xml output file.

xmlwrite.c