Indexer

Harris Seth & Von Quilon

For Indexer we were asked to navigate through a given directory and to go through all the files and figure out how many times each word appeared in each file. We started by creating an Indexer struct in which we placed a SortedList, the one we created in the last assignment. This SortedList wasn't the same one of course, it was modified. In each SortedList we placed another SortedList2. The primary SortedList was for the words, and the secondary SortedList2 attached to every SortedList struct was for the files in which the word was found. The SortedList2 also held an additional count field to keep track of how many times each word appeared in each file. The second SortedList2 was then sorted by the count, highest to lowest. The words were also sorted in the same way.

The efficiencies for all the operations was decreased obviously. Each insert originally took O(n) for the n objects, but now it would take O(n + m) because each time we inserted a word into the SortedList we would have to insert its file into SortedList2.which was of size m. Same for deletes, though that operation was not used in this assignment. To free the IndexPtr it took O(n*m), which was the same as the SortedList free operation.

To read a file we created a method called readFile that would take a file and an IndexPtr, and then proceed to grab each line in the file, tokenize it, then insert each word into the IndexPtr struct along with the file in which it was found. So if there are q words in the file then it would take p*n*m amount of time to run all the inserts, or O(pnm).

The saveFile method writes the SortedList into a new file and saves it into the current directory. It takes O(n*m) time to write the file because you have to write the entire SortedList and SortedList2.

The readDirectory function is the backbone of this assignment. Given a path it will go through the entire directory looking for files to insert into the Index struct. If we run into a directory in the directory we recursively go through that directory as well looking for files. We keep recursing for as long as we find more directories. The traversal through directories is done through the use of the dirent.h . It creates structs for the directory and you can use operations like opendir() and closedir() to navigate the directories. The entire assignment runs in the main.c file.