John Eng (je283) and Marielle Jurist (mlj85)
Systems Programming, Spring 2016

Assignment 3: Indexer

Implementation:

*Modifications were made to sorted-list.c and tokenizer.c to be especially used in our indexer.c file.

Each file is found using a recursive function that accesses every file in each directory, disregarding empty directories. Once the file has been reached, it is processed line by line using function calls within token.c and sorted-list.c.

Sorted-list.c is the main implementation of our code. It contains three structs:

```
struct fileData{
     char* filename;
     int frequency;
}
struct SortedList{
     node head;
     CompareFunct cmp;
     DestructFreq dfreq;
}
struct Node{
     void* data
     struct Node* next;
     int word count;
     SortedList fileList;
}
```

By using a void pointer within the Node, we allow the Node to contain either a fileData or a token. The way this is implemented allows us to shorten our code and reuse the sorting function on both the tokens and the fileData structs. Of course, the compare function also takes in generic data.

After successfully creating a sorted-list, the data is accessed through multiple loops and outputted into a output file in JSON format. As always, freeing of pointers is carefully managed.

Time Complexity:

Creating a the list:

On a given file with m lines, n tokens in within each line,and k fileData structs, the efficiency of the program is: O(n^2) to create SortedList of tokens + O(n*k^2) to create a SortedList of fileData structs within each node of the SortedList of tokens. This is done for m lines.

Big O = m*[O(n^2)+O(n*k^2)] = **O(m*n*k^2)**

printList:

O(n*f), where n is the number of tokens and f is the number of files