Indexer Design Spec

In the following Design Module we describe the input, data flow, and output specification for the indexer module. The pseudo code for the indexer module is also given.

The Design Spec should include:
[1] Input: Any inputs to the module
[2] Outputs: Any outputs of the module
[3] Data Flow: Any data flow through the module
[4] Data Structures: Major data structures used by the module
[5] Pseudo Code: Pseudo code description of the module.

[1] **Input**
Command Input
```
./indexer [SOURCE DIRECTORY] [TARGET FILE] [FILE TO READ] [TARGET FILE 2]
```

Example command input

```
Indexer data index.dat index.dat new_index.dat
```

[SOURCE DIRECTORY] data
Requirement: The directory must exist and be populated by files
Usage: The indexer needs to inform the user if the directory can't be found

[TARGET FILE] index.dat
Requirement: The file must not exist already
Usage: The indexer needs to inform the user if the file already exists

[FILE TO READ] index.dat
Requirement: The file must exist already or will be created during indexer, meaning it's identical to [TARGET FILE]
Usage: The indexer needs to inform the user if the files cannot be found

[TARGET FILE 2] new_index.dat
Requirement: The file must not exist already
Usage: The indexer needs to inform the user if the file already exists

[2] **Output**
For each file in the directory, indexer will parse through the html file and read each word. It will then keep track of the word found, the ID of the file it is in, and the frequency that the word appears in the file all inside of a hash-table. Once all files are read and accounted for, the index will be printed to [TARGET FILE] where each line is in the format, 'word' 'total count' 'doc_id' 'freq'…

[3] **Data Flow**
Each file in the [SOURCE DIRECTORY] is read and the contents, starting from the third line is stored as HTML code.

We then read through the stored HTML code and consider every word. For each word, we try to add it to our hash-table. The hash-table contains word nodes and doc nodes that hold what the word is, what document it was found in, and how many times the word came up in the document.

After all files are looked at and all HTML code has been parsed through, the hash-table will be passed into a function that prints out the contents of the hash-table in [TARGET FILE].

If only three arguments were passed, indexer will stop there, but if five arguments were passed, indexer then reads through [FILE TO READ], building another hash-table based off the contents of the file. It will then pass the hash-table into a function that prints out the contents to [TARGET FILE 2].

[4] **Data Structures**

> doc_id – An int that holds the doc_id of the file being searched
> freq – An int that holds the number of times a word appeared in a file
> Word – The word being searched for
> Page – A pointer to the corresponding DocumentNode
> Data – A pointer to the corresponding WordNode

Clearly, we need a data structure that will allow us to store the doc_id and freq for all unique files and words. We define a linked list of Document Nodes that hold this information.

> DocumentNode has a doc_id, freq, and a next pointer

We also need a data structure that will allow us to store all of the DocumentNodes along with the corresponding word that we are searching for and that is why we created the linked list of WordNodes

> WordNode has next pointer, word, and a pointer to DocumentNodes

In order to host all of the WordNodes that we are creating, we also created HashTableNodes which is a linked list

> HashTableNodes has a next pointer, and a pointer to WordNodes

For now it is fine just to understand that this is a list of some data structures that holds information about the word, freq and doc_id. In the implementation Spec we will flush out the detailed C structures for all of the above. But for now abstraction is fine. Details will come in the Implementation Spec.
We also need some way to track everything so we use HashTable.

[5] **Indexer** Pseudocode

```
// check command line arguments
Inform the user if arguments are not present or invalid
If [SOURCE DIRECTORY] does not exist or [TARGET FILE] and [TARGET FILE 2]
already exist, inform the user of usage and exit failed.
```

```
// initialize the hash table
Create a hashtable that will store all of our data and initialize it

//Read every file
Go through each file in the source directory

    // Read each file and store the HTML code in a string
        Make sure to take account for the fact that HTML starts in line three

    // Read each word
        Parse through the HTML and evaluate each word

    // Insert word into the hashtable
        Take the word and doc_id and input it into our hashtable
        Case 1: Word will be there with same doc_id so increase frequency
        Case 2: Word will be there with diff. doc_id so create new DocNode
        Case 3: Word is not there so create new DocNode and WordNode
        Case 4: Word is not there but HashTableNode!=NULL so add to end

// Print contents of the hashtable
Print all the contents of the hashtable into the target file

//If 5 arguments were passed continue
If five arguments were passed, more must be done

    //Read source file and input contents
    Take the file and read it
    Input the contents accordingly into a new hashtable

    // Print
    Print out the contents of the new hashtable into the target file2

//free all hashtables and resources.
```