Project notes

Index uses next data types:

```
/**
 * Data type to represent pair: <file_name> : <number_of_occurrences>
 * @sample: [ "file_name" : <occurrences> ]
 */
typedef struct
{
	char * file;
	size_t count;
} index_pair_t;

/**
 * Utility data type to store index_pairs as array
 */
typedef struct
{
	index_pair_t * pairs;
	size_t size;
	size_t capacity;
} pairs_list_t;

/**
 * Data type to represent pair: <word> : <list_of_pair_index>
 * @sample: {  "word":  [ "file_name" : <occurrences>, ...] }
 */
typedef struct
{
	char * word;
	pairs_list_t list;
} list_record_t;

/**
 * Utility data type to store all the word with its occurrences
 */
typedef struct
{
	list_record_t * records;
	size_t size;
	size_t capacity;
} mapped_list_t;
```

As it can be seen all parsed data is stored in simple dynamic arrays (something like vector from c++):

- Array or pairs {<file>: <count>}
- Array of {<word> : <array of pairs >}

This simple interface allows to implement program logic in a simple way, but has several disadvantages:

- Searching file name through files takes linear time O(n)
- Searching word through all words takes also linear time O(n)
- Before writing to the json file it is required to sort all the data that takes additional time according to the quick sort implementation

Using of hashtable instead of dynamic arrays will give linear time of search that will highly increase performing time.

Used algorithm:

1) Read file line by line
2) Split read line by separators
3) Update information related to each word
4) After the data was successfully read – sort the mentioned arrays
5) Write data to file
6) Release all allocated memory (files names, words, arrays)