The core of the data is stored in a single map named songWords.

```
songWords key is type string, value is structure number

stuct number
      value: type int, value of word counter
```

Words are stored in an key-value relationship in the map, with the key being individual cleansed words from file. The value represents the number of occurrences of the word in the file.

```
string promptForFile();
```

Prompts the user for the name of the file

do-while loop utilizing temp string and cin to grab user input

```
int readThisFile(map<string, number> &songWords, string name);
```

Reads words from file and appends into map

Opens file and utilizes stringstream to get individual words

Cleanse words using regular expression to strip punctuation

Uses transform to ensure all words are lowercase

Check if word already exists in map

If yes, grab the value and increment it by one

If not, add the key to the map and assign value of 1

Program returns longest word in the file

```
int printMap(map<string, number> &songWords, int longest);
```

Prints the map

Using a for loop and C++ 11 auto loop, loop pre-sorted Map and pring out results

Returns the most frequent word's count

```
void writeToFile(map<string, number> &songWords, int longest, int most);
```

Writes the results to a file

Opens a file to prepare for writing

Writes information, including loop to write results of Map

Exits program

Note on why I used Map instead of Array:

Search: Map: O(1) // Array: O(n) **(assuming no collisions)*

If utilizing an array in combination with a search (can't use binary search because array is not sorted), the running time of filling the array would be:
O (n (each word) * n(searching each word)) = O(n^2)

Utilizing a map:
O(n (each word)) = O(n)

Hence, it's far more efficient to use a Map, not only because of the above, but because of a Map's presorted nature.