DESIGN DOCUMENT

*File Descriptions*

*Server.c*

This file will contain the code for setting up the communication for the server, first the server will send a question to the client and get answers back with a word and the number of syllables in the word.

The server will receive the text file, sanitize/standardize the input, and then add each word to a hashmap with the syllable count as the key. When requested, the algorithm for creating a haiku will be:

Note: we assume that the words in a haiku can span line breaks, as long as the number of syllables if 17. We will not be inserting hyphenation.  
Randomly generate a number < 17. This will be the number of syllables we start with. We add this word and syllable count to another hashmap. We have a pointer that keeps track of the current syllable count. We increment the pointer. If the current syllable total + this syllable count is over 17, we decrement the pointer until we reach a syllable count that is 17 or less. Once we have been able to reach 17 syllables with words (or after we reach a syllable count of 1 and are unable to add new words to reach 17), we have a hashmap containing words that sum to 17 syllables. We iterate through the hashmap and write each word to the buffer. We write the resulting haiku to the shared memory.   
If there is an included/excluded words list, the included words will be selected first. For the excluded words list, each word in that list will be added to the hashset. As each word in the source.txt is parsed and added to the hashmap, it will be checked that the word is not on the excluded list. If it is in the excluded list, it will not be added to the hashmap containing syllable counts.

*Client.c*

This file will contain the code for setting up the socket for the client. First, the client needs to connect to the socket and then send the input to the server. The user will run the program with an input .txt file. This .txt file will be uploaded to the server (to the shared memory). If there is an excluded or included words list, then that will also be uploaded.

The client then waits for the server to respond with a haiku, prints the haiku, and closes the socket.

*Sample input*

A syllable .txt file containing a rows formatted as WORD SYLLABLE\_COUNT (ex: Bananas 3)

A .txt file containing words to not include, formatted as rows containing WORD

*Major Data Structures*

*Source Words HashMap*

A Hashmap with the number of syllables as the key and the word itself as a string. We chose a hashmap for quick lookup and also to be able to look for words by key. The final implementation uses a Map<Integer, List<String>> with integers as the keys (number of syllables) and the value is a list of words with that syllable count.

*Haiku Hashset*

Once the words are picked from the hashmap, they’re written to this hashset. When the hashset is emptied and read out, the words will be unordered, as opposed to in an array, where they would have to be in a pre-allocated space and in descending order.

*Excluded Words Hashset*

Data structures considered: Hashmap/set, Array, LinkedList. LinkedList immediately excluded due to slow search time. A Hashset would be more practical than a hashmap because we don’t have a logical key for the words here. Lookup in a hashset is faster than in an array. As such, we create this hashset from the excluded\_words.txt the client sends. As we add words to the haiku hashmap, we can check if they are in this hashset and excluded them if they are.

*Sample Output - we weren’t able to get our code to work, so this is our mock output!*

*Yellow banana*

*Yummy looking banana*

*No more bananas*

CONTRIBUTIONS

Maria

* All content in client.c
* Hashmap/hashset/haiku creation algorithm

Sandy

* Server