**ADS 1 Mid-term CW**

**A poetry assistant**

**Section 1:**

For this assignment, I have chosen to utilize the Dynamic Array and Associative Array (Map) data structures. In section 2, I will provide a detailed explanation of how these data structures are employed. In summary, the combination of these data structures enables the program to achieve remarkably fast lookup speeds when the user searches for rhymes. The Associative Array (Map) allows for constant time complexity (O(1)) when accessing a value by its corresponding key. In contrast, searching an Array typically requires linear time complexity (O(n)).

**Section 2:**

To find rhymes, I will use a two-step process. First, I will read the list of words from the file and store them in a dynamic array data structure. Then, I will iterate over each word and for each word, I will create a temporary list to store its rhymes. Within a nested loop, I will check if the last three letters of the current word match the last three letters of another word. If they do, I will add that word to the temporary list of rhymes.

Once I finish comparing the current word with all other words, I will check if the temporary list of rhymes is empty. If it's not empty, I will associate the current word with its list of rhymes in a data structure called an associative array.

If the temporary list of rhymes is empty, it means there are no rhymes for that word. In this case, I will associate the current word with a message indicating that there are no rhymes.

After iterating over all the words, the associative array will contain each word as a key and its associated rhymes (if any) as values.

This implementation allows for efficient retrieval of rhymes, as finding a value by its corresponding key in the associative array is a fast operation that doesn't require searching the entire array every time a user inputs a new word.

If the word that the user types is not found in the associative array, I will then compare it with the entire array of words. However, since my associative array is built with a list of almost a quarter of a million English words, the chances of not finding a match with the user's input are extremely low.

**Section 3:**

dataArray = CREATE\_DYNAMIC\_ARRAY()

rhymesData = CREATE\_ASSOCIATIVE\_ARRAY()

f = OPEN\_FILE(“data.csv”)

READ\_CSV(f, dataArray)

CLOSE\_FILE(f)

k = 1

for i = 1 to LENGTH(dataArray)

rhymesWords = CREATE\_DYNAMIC\_ARRAY()

for j = 1 to LENGTH(dataArray)

if LAST\_THREE\_LETTERS(dataArray[i]) is equal to LAST\_THREE\_LETTERS(dataArray[j])

rhymesWords[k] = dataArray[j]

k++

END if

END for

if rhymesWords is not empty

SET(rhymesData, dataArray[i], rhymesWords)

else

SET(rhymesData, dataArray[i], “There are no rhymes for this word”)

END if

END for

userInput = GET\_USER\_INPUT()

rhymes = CREATE\_DYNAMIC\_ARRAY()

if rhymesData contains userInput then

rhymes = GET(rhymesData, userInput)

for i = 1 to LENGTH(rhymes) do

display rhymes[i]

END for

if rhymesData doesn’t contain userInput

inputRhymes = CREATE\_DYNAMIC\_ARRAY()

j = 1

for i = 1 to LENGTH(dataArray)

if LAST\_THREE\_LETTERS(userInput) is equal to LAST\_THREE\_LETTERS(dataArray[i])

inputRhymes[j] = dataArray[i]

END if

END for

for I = 1 to LENGTH(inputRhymes)

display inputRhymes[i]

END for

else

display “No rhymes found for the input word”

END if

**Section 4:**

I have opted to utilize the Dynamic Array and Associative Array (Map) abstract data structures for their simplicity and efficiency in the algorithm. By avoiding the use of a dynamic array and comparing each input with the entire array repeatedly, we can conserve resources. Instead, we can perform this comparison only once for each word and define the corresponding rhymes, which are then assigned as key-value pairs in an associative array (map). This approach allows for efficient retrieval of values by key using the associative array, as it operates in constant time complexity (O(1)), whereas searching an array would typically require linear time complexity (O(n)).

**Section 5:**