CS21120 Assignment

## Introduction

In this assignment we were tasked to output a list of words that rhymes within a given word.  
To do this we were given a CMU dictionary text file that contains ~135,000 words, each with their very own pronunciation.   
To see if two words rhyme both pronunciations must have the same final stressed vowel and after must contain the same sounds.

For the coding assignment I didn’t use ChatGPT or IntelliJ Ai Assistant, only resources online such as java documentation on different data structures, Stack Overflow, W3Schools, GeeksforGeeks and other websites. As for the documentation I will be using a grammar checker website QuillBot.

## Task 1: Implementing IPhoneme

Implementing IPhoneme was relatively simple because I had to create a Phoneme class that complied with the IPhoneme interface, which then generated methods from the interface and added extra codes, such as before the constructor arguments are stored as attributes that the respective vowel or non-vowel phoneme stress value is valid. For example, a vowel phoneme cannot have a stress value of -1, whereas a non-vowel phoneme must have a stress value of -1.

## Task 2: Implementing IPronunciation

Implementing IPronunciation was a bit tricky for me at first because I couldn’t quite understand how the whole rhyming process worked therefore, I couldn’t really understand why finding the last stressed vowel was used for, but I managed to understand it and code it later down the line.

To find the final stressed vowel, I had two choices as to either go through the list of Phonemes front to back or vice versa. When looking through the CMU dictionary text file, I realized not all primary stressed vowel words are towards the end of the pronunciation of the word, but rather sometimes they can be towards the front or end and even sometimes both. Since the idea of is to find the final stressed vowel, I decided to go from back to front where upon finding a primary stressed vowel it will instantly return the index otherwise it will go through the whole list of Phonemes and return the index with priority to former highest stress vowel.

Initially I used a LinkedList as the data structure to store the list of Phonemes however, upon reevaluating my completed code, I realized an ArrayList is a more suitable data structure because of two specific operations I used for Pronunciation class, a add phoneme to list of Phoneme and find final stressed vowel index methods.  

When it comes to adding an element to an ArrayList it has a worst-case time complexity of O(1) if the ArrayList isn’t full however, when adding with the capacity full, then a new ArrayList will be created and all of Phonemes from the original ArrayList will be passed to the new ArrayList with a bigger capacity making it O(n).  
One thing I would like to point out is that when adding to an ArrayList, it will have a default capacity of 10 and when reading from the CMU dictionary text file, majority of the words will have a list of Phonemes of 10 or less therefore, making the average case time complexity of θ(1) and in the rare cases O(n) will happen.  
As for ArrayList indexing, the worst-case time complexity is O(1) which is constant time.

LinkedList on the other hand, holds references to the first and last element, so adding will have a worst-case time complexity of O(1) however, as for LinkedList indexing, it has to traverse the list until it finds the element, therefore it has a worst-case time complexity of O(n).

As for the finding final stressed vowel index methods with an ArrayList, its worst-case time complexity is O(n) assuming there is no vowels or primary stress within the pronunciation.   
However on LinkedList it would have to be reserved before traversing back to front but since it’s a doubly linked list which can be reversed in O(1) then worst-case is O(n) when it comes to finding final stressed vowel index. In the case of average time complexity, it would be O(n) when it comes to both ArrayList and LinkedList, because final stressed vowel can be found randomly anywhere within the list of Phonemes therefore on average it will take n/2 indexing time to find it which is linear time so O(n).

I concluded to use ArrayList over the LinkedList because both ArrayList and LinkedList insertion has an average case time complexity of θ(1), but the main differences with the ArrayList and LinkedList are the indexing worse case time complexity with ArrayList with O(1) and LinkedList with O(n) therefore, in this context an ArrayList is a better data structure to use over LinkedList.

## Task 3: Implementing IWord

## Task 4: Implementing IDictionary and parsing the CMU file

## Task 5: Rhyming!

## Self-evaluation