RhymeBot

CSE 250 - Fall 2015

RhymeBot will be a data structure that can efficiently find rhymes and alliterations for given words. Your task is to apply the data structures you've learned about this semester to store words with pronunciations and parts of speech to be queried later. For full credit, your data structure must find rhymes and alliterations efficiently with respect to time.

We will be utilizing the The CMU Pronouncing Dictionary for word pronunciation and WordNet from Princeton for the parts of speech. The code to parse these datasets has been provided so you can focus on the data structures that store and query this information.

Relevant code can be found in the course repository:
https://bitbucket.org/hartloff/cse250-fall2015/src/557be3b01ac4/homework/RhymeBot/?at=master

RhymeBot - All Parts

Due: Friday, December 11 @ 11:59pm

Inserting Words

There are two functions that will be used to add information about words into your RhymeBots data structures (dictionary). The first function is used to insert a new word along with its pronunciation.

```
void insertWithPronunciation(string word, vector<Sound>
                pronunciation);
```

The pronunciation is given as a vector of instances of the Sound class which has been provided for you. The Sound class stores the pronunciation information for each sound that has been extracted from the CMU dictionary. For this assignment, the most important information about a sound will be whether it is a vowel or not.

The last part of this assignment will involve parts of speech (adjective, adverb, noun, verb) for words that will be provided through the following function call.

```
void addPartOfSpeech(string word, PART_OF_SPEECH partOfSpeech);
```

This will add a part of speech to `word` if it has been added to the dictionary previously. If `word` is not already in RhymeBot, then this function should not do anything (ie. you don't need to store words that have a known part of speech but no known pronunciation). The part of speech can be viewed as extra information that may or may not be provided for each word, but the pronunciation is required for any word stored by RhymeBot.

There are no points associated with these two functions, but they are necessary to accomplish to remaining functionality. The data structures chosen to store the dictionary should be chosen with the rest of the assignment in mind.

Helper Functions (30 points)

1. Count Syllables (10 points)

The following helper functions may prove useful while working on the rest of the functionality of RhymeBot. The first function will count the number of syllables in the given word.

```
int countSyllables(string word);
```

The number of syllables in a word is equivalent to the number of vowel sounds it contains. Note that the Sound class has an isVowel() function.

2. Count Rhyming Syllables (10 points)

The next function will count the number of rhyming syllables in two words.

```
int rhymingSyllables(string word1, string word2);
```

Two words are said to rhyme if their suffixes upto and including the last vowel have identical sounds. The number of rhyming syllables is the number of vowel sounds in the maximum size suffix that has identical sounds in the two words. If the two words do not rhyme, return 0.

3. Count Alliterating Sounds (10 points)

The next function counts the size of the alliteration formed between two words.

```
int alliterationSounds(string word1, string word2);
```

Two words form an alliteration if they begin with the same sound. We will define the size of an alliteration as the maximum number of matching sounds in the prefixes of the two words. Note that when computing the size of an alliteration we will count all sounds in the matching prefix, but the size of a rhyme will be the number of vowel sounds (syllables) in the matching sounds of the suffixes.

Finding Rhymes and Alliterations (30 points)

1. Get All Rhymes (15 points)

After the previous parts are complete, you will have data structures filled with word pronunciations and several functions to compare words. Now we will query this data to find rhymes and alliterations.

```
vector<string>* getAllRhymes(string word);
```

This function will query the data structures used to store the words that have been added with `insertWithPronunciation` and return all known words that rhyme at least one syllable with the given word. The vector can store the words in any order. If no rhyming words are found, or if the given word in not in the dictionary, an empty vector will be returned.

2. Acquire All Alliterations (15 points)

When we wonder what works wonderful with one word.

```
vector<string>* getAllAlliterations(string word);
```

This will return all known words that form an alliteration of at least 1 sound with the given word. If no alliterations are found, or if the given word in not in the dictionary, an empty vector will be returned.

Finding Best Rhyme and Alliteration (25 points)

1. Find Best Rhyme (10 points)

In this section we will implement the end use case for RhymeBot. Given a word and some criteria we will return the “best” rhyming word. Some of these functions will make use of the parts of speech that were added to some of the words in the dictionary. The best rhyme will be the word that rhymes the most syllables with the given word. If there is a tie in any of the functions in this section, any of the tieing words can be returned.

```
string findBestRhyme(int numberOfSyllables, string rhymesWith);
string findBestRhyme(PART_OF_SPEECH, string rhymesWith);
string findBestRhyme(int numberOfSyllables, PART_OF_SPEECH, string
rhymesWith);
```

These build up to the third of the overloaded functions which asks for a word with a particular number of syllables, part of speech, and rhymes with the given word. This function can be used in poetry or song lyrics where most of the structure is in place, but one last word is needed to complete the work. Note that the number of syllables refers to the total number of syllables in the returned word and not the number of rhyming syllables.
If no word if found that fits all the criteria, or if the given word is not in the dictionary, return “”.

2. Find Best Alliteration (10 points)

Same as the previous section with alliterations instead of rhymes. The best alliteration is defined as the word with the most number of matching sounds that form an alliteration with the given word.

```
string findBestAlliteration(int numberOfSyllables, string alliterationWith);
string findBestAlliteration(PART_OF_SPEECH, string alliterationWith);
string findBestAlliteration(int numberOfSyllables, PART_OF_SPEECH, string
alliterationWith);
```

If no word if found that fits all the criteria, or if the given word is not in the dictionary, return “”.

3. Find Perfect Word (5 points)

When you need a perfect word, combine `findBestRhyme` and `findBestAlliteration`.

```
string findPerfectWord(int numberOfSyllables, PART_OF_SPEECH, string
                rhymesWith, string alliterationWith);
```

This function is similar to the previous two except it takes 2 input words. One word that must rhyme with the return word and one that must form an alliteration with the return word. The perfect word must be returned which is defined as the word that maximizes the sum of the number of syllables rhyming with `rhymesWith` and the number of sounds alliterating with `alliterationWith`.

Efficiency (15 points)

To receive full credit on this assignment, the following three functions must run in O(log(n)) time where n is the number of words added to RhymeBot by calling `insertWithPronunciation` (5 points each).

```
string findBestRhyme(int numberOfSyllables, PART_OF_SPEECH, string
rhymesWith);

string findBestAlliteration(int numberOfSyllables, PART_OF_SPEECH, string
alliterationWith);

string findPerfectWord(int numberOfSyllables, PART_OF_SPEECH, string
rhymesWith, string alliterationWith);
```

This is your last coding task in CSE250 and it will test your ability to utilize data structures in C++.

Submission

Submit all your code in `RhymeBot.h` and `RhymeBot.cpp`. Submissions must be made on the CSE servers using the `submit_cse250` command.