CS 2420 Program 4 – 20 points  
Who says I can’t write poetry?

**Part 1: Becoming familiar with the code**

HashTable code has been given to you. No testing program has been provided. To become familiar with how the code works, try reading in a small input file and make sure you can create a hash table of those entries.

**Testing:** Make sure the following works:

a. Insert values

b. Delete values

c. Find values

d. Printing the contents of the hash table.

e. Control the size of the hash table.

What happens if you attempt to delete an item that isn’t there?

What if you add more things than can fit into the hash table?

**Modification:** You have been given the hash table code from your text for doing quadratic probing. Modify the code so it takes the key and the associated object as two separate parameters. Now, the key and the associated object are in a single data structure of type E.

Please retain the generic structure of the hash table. Be careful not to add anything to the hash table that only makes sense for this specific problem.

Part 1 is for your benefit. The code you turn in does not need to show the results of this experimentation.

**Part 2:**

The class WordFreqInfo is meant to store a word, how many times it occurs, and a list of all the words (and their frequencies) that follow it. Create a main program for the WordFreqInfo which tests the code.

**Part 3:** **Generate the WordFreqInfo for the file.**

This assignment is designed to give you experience with hash tables. In this assignment, you will randomly create a new verse of poetry (based on the pattern seen in the existing poem).

You will fit a bi-word language model to English and then use it to generate a poem. A uni-word model of English consists of a single probability distribution ***P****(W)* over the set of all words.

A bi-word model of English consists of two probability distributions: ***P****(W)* and ***P****(Wi+1 | Wi)*. The first distribution is just the probability a word in a document. The second distribution is the probability of seeing word *Wi+1* given that the previous word was *Wi*.

Given a set of documents (in our case, various poems), your job in this assignment is to generate a poem from a given word. The method is simple. You just need to probabilistically decide which word should follow the current word. The word selected is based on the probability in which it follows (in the file) the current word. So for example, in the text below, if I ask, “What should follow ‘sam’”?, you pick ‘I’ most of the time (6 out of 8 times) and ‘.’ the rest of the time. No other word follows ‘sam’ in this example.

I am Sam . I am Sam . Sam I Am .  
That Sam I Am ! That Sam I Am ! I do not like that Sam I Am !  
Do you like green eggs and ham ?  
I do not like them , Sam I Am .  
I do not like green eggs and ham .  
Would you like them here or there ?  
I would not like them here or there .  
I would not like them anywhere .  
I do not like green eggs and ham .  
I do not like them , Sam I Am .

Thus, you need to keep track of the number of times that each word follows every other word (so you can figure probabilities).

Keep track of all words (and their occurrence count) using a hash table. So when you are reading the file, the approach is

*currWord = read Word;*

*add currWord (as a WordFreqInfo) to hash table*

*While more words{*

*nextWord= Read word*

*update CurrWord’s follow list with NextWord*

*If nextWord already exists in hash table, update its occurrence count*

*If nextWord doesn’t exist, add it (with occurrence count of 1)*

*currWord = nextWord*

*}*

For the previous poem, contents of the hash table are shown below. The values in red are the locations in the hash table where the WordFreqInfo happens to be stored. Thus, “do” occurs 6 times in the file. Five times “do” is followed by “not” and one time it is followed by “you”

3: Word :them: (5) : , [2] here [2] anywhere [1]

33: Word :!: (3) : that [1] i [1] do [1]

41: Word :anywhere: (1) : . [1]

44: Word :,: (2) : sam [2]

46: Word :.: (9) : i [5] sam [1] that [1] would [1]

47: Word :do: (6) : not [5] you [1]

60: Word :green: (3) : eggs [3]

63: Word :?: (2) : i [2]

89: Word :and: (3) : ham [3]

92: Word :ham: (3) : ? [1] . [2]

97: Word :that: (3) : sam [3]

105: Word :i: (15) : am [8] do [5] would [2]

113: Word :sam: (8) : . [2] i [6]

126: Word :would: (3) : you [1] not [2]

138: Word :there: (2) : ? [1] . [1]

162: Word :am: (8) : sam [2] . [3] ! [3]

179: Word :or: (2) : there [2]

180: Word :not: (7) : like [7]

189: Word :like: (9) : that [1] green [3] them [5]

198: Word :eggs: (3) : and [3]

200: Word :here: (2) : or [2]

202: Word :you: (2) : like [2]

**Part 4: Generate Poems**

Write the function pickNextWord(myWord) which probabilistically picks the word that should follow myWord. Convince yourself that it picks the next word correctly.

From our data, we can compute the conditional probability that “i” occurs once you are looking at word “sam” *P(i|sam)* as 6/8. Similarly, we can compute the probability *P(them|like)* as 5/9.

Thus, pickNextWord(“sam”) should pick “i”, 75% of the time and “.” 25% of the time.

Given a specific input file, you are to generate a poem of a given length from a specific starting word. Output a newline after “.”, “,”, “!”, or “?”, End the poem with a “.”. So for example, poem(“green.txt”, “sam”, 20) is to generate a poem of 20 words that begins with the word sam using the probability distribution found in file “green.txt”.

Your poem might look like:

sam.

would you like green eggs and ham?

i am sam.

i am.

i do not.

Or possibly:

sam i do not like green eggs and ham.

that sam.

sam i am ! i am !.

**Specifics**

For the data, change everything to lower case.

**Output**

There are several data files associated with the test. You will generate poems from the following specification:

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Beginning word** | **Length of Poem** | **Print Hash Table?** |
| green.txt | sam | 20 | Yes |
| Lester.txt | lester | 30 | Yes |
| HowMany.txt | how | 30 | No |
| Zebra.txt | are | 50 | Yes |

For grading purposes, you are asked to print the contents of the hash table for some of the test cases. This will be a huge benefit to you in debugging. Because the next word is generated via a random number generator, you should not expect your output to match that of others.

**Hints**

In order to generate which word follows word WORD with the correct probability, I used the following strategy. I kept track of how many times WORD occurred (call it occurCt) and how many times each word followed WORD (followCt). I generated a random number between 0 and occurCt. Then I considered each word which followed WORD in order. I added up the followCts of each following word until the sum of the followCts passed the random number. When the sum passed the random number, that is the word I selected to be the next word in the poem.

However, don’t just “guess” that you are doing this correctly. Have pickNextWord talk to you.

You could have pickNextWord tell you

(a) What word frequency info are you looking at

(b) What random number did you generate

(c) What word was selected.

For example, my debug may look like:

 pickNextWord Word :to: (2) : forty [1] wish [1]   >> rand 2 picked wish  
 pickNextWord Word :wish: (5) : by [1] he [1] , [1] to [1] for [1]  >> rand 1 picked by  
 pickNextWord Word :by: (1) : the [1]  >> rand 1 picked the  
 pickNextWord Word :the: (4) : goblin [1] banyan [1] ground [1] lot [1]  >> rand 2 picked banyan

pickNextWord Word :,: (7) : he [2] plus [1] which [1] eighteen [1] sat [1] take [1] >> rand 6 picked sat

**Bonus (2 points)**

Try picking the next word based on the TWO previous words. For example, you hash table would be based on pairs (in my case a hyphen separated the two words) and may look like:

2: Word :am-sam: (2) : . [2]

3: Word :ham-?: (1) : i [1]

4: Word :i-would: (2) : not [2]

7: Word :here-or: (2) : there [2]

8: Word :green-eggs: (3) : and [3]

9: Word :anywhere-.: (1) : i [1]

10: Word :or-there: (2) : ? [1] . [1]

12: Word :them-,: (2) : sam [2]

13: Word :not-like: (7) : that [1] them [4] green [2]

16: Word :there-.: (1) : i [1]

18: Word :do-not: (5) : like [5]

19: Word :that-sam: (3) : i [3]

26: Word :,-sam: (2) : i [2]

29: Word :you-like: (2) : green [1] them [1]

Do your poems seem to make more sense when the next word is based on two previous words? In the submission comments, be sure to ask the grader to grade the bonus part.