CSE101 Assignment
HW1: Much ado about linking

- All code must be written in C/C++.
- Please be careful about using built-in libraries or data structures. The assignment instructions will tell you what is acceptable, and what is not. If you have any doubts, please ask the instructors or TAs.

1 Problem description

Main objective: We're going to do a word analysis of all the compositions of William Shakespeare. In your assignment directory, you will have the full text of all compositions of William Shakespeare. The file is called `shakespeare_cleaned.txt`. (It was obtained from Project Gutenberg, so there are some copyright messages in it.)

To make your life easier (and faster), this file is a cleaned version of the original source. Each line has a separate word, all in lowercase. All the tokenizing has been done for you, so you can read each line directly as a separate word. To make the file manageable for this assignment, *all words less than five letters long have been removed.*

We want to answer the following queries from the text.

- For any word length ℓ and number k, what is the k most frequent word of length ℓ?

Setup: You can access a Codio unit (which I also call a Codio box) for this assignment. There is a directory "Bard" with the file "shakespeare_cleaned.txt". You must write all your code in that directory, which is where the executable should be created. There are also some testing scripts and example input/output files. Please check out the README for more details on that.

Format and Output: You should provide a Makefile. On running `make`, it should create an executable "bard". You should run the executable with *two* command line arguments: the first is an input file, the second is the output file. You must provide a README with a short explanation of the usage and a description of the files involved.

All your files must be of the form *.c, *.cpp, or *.h. When we grade, all other code files will be deleted. (So do not try to script some part in another

language.)

Each line of the input file corresponds to a new query. The query is a pair of numbers: LENGTH RANK. The first number is the length of the word, the second number is the *rank*, which starts from 0. The ranking is done in decreasing order of frequency, and *increasing* lexicographic order. Thus, if two words have the same frequency, then the word that is earlier lexicographically has the smaller (earlier) rank.

So, "7 0" refers to the most frequent word of length 7. Or, "9 3" refers to the fourth most frequent word of length 9.

You do not need to worry about error handling on the input. You may get ranks or words length that do not exist, so your code must be prepared to handle such cases.

The output for each line is the corresponding word. If no word exists, the output is '-'. This can happen if the Bard decided not to use any words of that length, or there is no word of that rank. For example, if the input file is:

6 3
10 0
9 2
8 14
8 15
26 0

The output file is:

father
gloucester
messenger
business
personal
-

So "gloucester" is the most frequent word of 10 letters, etc. It turns out that both "business" and "personal" (both with 8 letters) have the same frequency of 230, but we rank "business" earlier than "personal".

Data structure instructions: You cannot use inbuilt data structures in C++. No vectors or inbuilt lists to store the words. No inbuilt iterators. You must store the words in a linked list, which you write yourself. The whole point of this assignment is to built the list from scratch, as a linked list. *Just to be clear: you cannot use inbuilt data structures, or store the words in an array. You must store them in a linked list.*

You need to maintain a linked list that stores words and their frequencies. As you parse and process the input file, for every word encountered, you must update this linked list of words and frequencies. At this point, you have all the information on the Bard's work, and now you have to figure out how to actually produce the output.

You cannot use any built in lists for this step either. I suggest the following: create an array of linked lists, where the k linked list stores all words of length k. Once you have found all words, it is easy to figure out what the size of the array should be.

You may want to reorder or sort each of these linked lists. Then, producing the output should be fairly easy.

2 Grading

You code should terminate within 2 minutes for any input file with at most 200 queries. If it doesn't, we will not give you credit. It's quite hard to give partial credit for this problem. You'll notice that once you get a few corner cases correct, your code will completely work.

1. (10 points) For a full solution as described above.
2. (3 points) Well, if you pass some test cases, but not others. If your code works on `more-input.txt`, this is highly unlikely.

3 Fun facts

Have some fun with this assignment. Can you find out the number of unique word that dear Bill used? What if you also tokenize by hyphens? What changes?

Some things that I learned.

- Shakespeare wrote a lot of text involving "father", as well as "daughter".
- The longest word without hyphens is "honorificabilitudinitatibus".
- The coolest word, IMHO, is "anthropophaginian".
- "caeser" is more frequent that "brutus", who both beat "othello".
- Clearly, "tennis" was played in Elizabethan times.
- There are two "impossibilities" in all of Shakespeare's work.

3.1 In case you care

How I parsed the original file: This is not relevant for the assignment, but just letting you know if you're interested. In general, parsing literary texts is a fairly messy task, since we have to make numerous decisions on how to split words. There is no one right way. I decided to tokenize by whitespace *and* any of the following punctuation marks/symbols:

- Question mark (?)
- Comma (,)
- Period (.)
- Exclamation mark (!)
- Colon (:)
- Semicolon (;)
- Square brackets ([or])

It was quickest and easiest to do in Python, first replacing the above symbols by whitespace, and then call the `split` function.