CSE101 Assignment
HW1: Mark your pointers carefully

- 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 look at the words of Mark Twain. Yes, literally, we will look at all his words. In your assignment directory, you will have a file called `twain-cleaned.txt`. (It was obtained from Project Gutenberg, `http://www.gutenberg.org/ebooks/3200`.)

To make your life easier, 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 six letters long have been removed.*

We want to answer the following queries from the text.

- For any starting letter α and number k, what is the kth most frequent word (and corresponding frequency) starting with the character α?

Setup: You can access a Codio unit (which I also call a Codio box) for this assignment. There is a directory "Twain" with the file "twain-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 "twain". 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, *.h, or *.hpp. 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 character and a number: CHAR RANK. The first character is the starting character 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, "c 0" refers to the most frequent word starting with 'c'. Or, "i 3" refers to the fourth most frequent word of length 'i'.

You do not need to worry about error handling on the input. The character will always be something from 'a' to 'z'. The rank will always be a non-negative number. But you may get ranks that do not exist, so your code must be prepared to handle such cases.

The output for each line is the corresponding word and the frequency (separated by a single space). If no word exists, the output is '-'. This can happen if Mr. Clemens did not use that many distinct words starting from the letter.

For example, if the input file is:

```
c 0
x 2
x 3
c 10000
```

The output file is:

```
course 1534
xenocrates 1
xenophon 1
-
```

So "course" is the most frequent word starting with 'c', and occurs 1534 times. Both "xenocrates" and "xenophon" occurs once each, but the former has a lower rank. There are less than 10000 distinct words starting with 'c', so the answer to the last query is "-".

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 can use libraries for doing I/O.

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. You need to think about how to store all this information (maybe more than one linked list?).

You may want to first read all the words, and store them in a collection of lists with words and their frequencies. You may want to reorder or sort each of these linked lists. Then, producing the output should be fairly easy.

Common pitfalls:

- Hardcoding the input or output files: *Don't* do that. The input and output files are read as command line arguments. Do not confuse the input/output files with the file `twain-cleaned.txt` that contains all the words of Mark Twain.
- Trying the read the whole file to process each query: The whole point is to construct a fast data structure that allows for queries to be answered quickly. Your program will probably take some time to read all the words into a data structure, but the queries should be really fast.

2 Grading

You code should terminate within 2 minutes for any input file with at most 300 queries. If it doesn't, we will not give you credit. It's quite hard to give partial credit for this problem. 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 Stuff I learned

- The word "xhvloj" makes an appearance. I thought it was a typo, but no, it actually appears in Mark Twain's work. I have no idea what it means.
- There are a few more "gentlemen" than "gentleman".
- "Xylobalsamum" is the dried, fragrant wood of the Balsamodendron gileadense that produces resin known as Balm of Gilead. There are more occurrences of "zylobalsamum", which is a name in the book "The American Claimant".
- There is more "mother" than "father.

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])

- Parentheses
- Hyphen (-)
- Underscore (_)

It was quickest and easiest to do in Python, first replacing the above symbols by whitespace, and then call the `split` function. In addition, I removed any word with the substring “xx” or “xvi”, since there were Roman numerals denoting chapters.