Cosc 2150 Program 1
Due: Nov 1, 2019 100 points

In this assignment you will implement your second spell checker. For this version, you are going to implement the spell checking with a tree data structure. You can NOT use any STL data structures for the dictionary, instead you are going to write the tree code and then use it in the spell checking.

You will be given a dictionary, in which the words are in a random order. The words in the dictionary are unique, but some may have capital letters. The dictionary is large, having over 125,000 words in it. Reading this in word by word will be easy, since there is one word per line. The dictionary will be entered into the tree data structure you wrote.

The book has almost a million words in it! This will be a bit more complicated, since every line will have multiple words. Read each word and then send it to a "clean word" method. When reading the book, a word will be defined as between two spaces (or end of line). I suggest you read use >> instead of getline, since it will read based on spaces. You will need to write a "clean word" function that will then remove any unnecessary "stuff" from the word, see the specs below. You will need use the clean word method to sanitize books word and the dictionary words (before adding them to the dictionary data structure). You will also be timing this as well.

"Clean word" method:

- This is to be implemented as a separate method, so it can be called for either a dictionary word or book word.
- The function will remove any non-letters, except an apostrophe (') and numbers from the word.
- All letters will changed to lower case.
- It will then return the updated word. The word could now be blank. The calling method will need to deal with this correctly.

Program requirements:

1. You must write a tree data structure. You cannot use the STL data structures to hold the dictionary. You can however use other STL methods and functions in other places.
2. "Clean word" method must implemented as described above.
   - Any word that starts with as a non-letter (after returning from the clean word function) will be skipped for spell checking. It's not in the dictionary.
   - Any word that comes back blank is not checked nor counted as skipped.
3. You will use the time code provided to time the spell checking. The time to initialize the dictionary IS NOT part of the timing. It is start at the point you open the book file.
4. Output the following information. See the next page for required format of the output.
   - Time to spell check
   - Number of words spell correctly
   - And the Number of compares, and average number of compares for spelled correctly.
   - Number of words not spelled correctly
   - And the Number of compares, and average number of compares for misspelled
   - Number of words skipped.
5. Use the timing code provided, to time the run time.
6. You must write out a file of all the misspelled words (every instance of the misspelled word too) to a file called misspelled. Print each word on a separate line. This should be done after the timer has been stop, otherwise if will affect your runtime.

7. You code must be comment and have good coding style (variable naming, indention, etc) and structure. Points will be taken for poor structure and bad code style.

Specifically forbidden items:

1. You may not preload the book into any data structure. You are to read a word from the book file, process it in the clean word method, and finally check it against the dictionary data structure.
2. You may not add any other dictionaries or dictionary files and must use the dictionary provided.
3. The dictionary data structure must be ONE complete data structure, you cannot as an example declare 5 separate data structures for the dictionary in the main code. This would be done inside the dictionary data structure itself.
4. The Clean word method MUST be a separate method and may NOT be part of the dictionary data structure.
5. You may not have any ASCII based numbers in your code or check based on an ASCII number.
   - While you can use the ASCII table, no where can you hard code any ASCII table numbers into your code.
6. Comparing strings with the == anywhere in your code.

Output section: No other output, besides what is below, number is RED are expected to be different, but use the same formatting. Removing ALL of your debugging lines, this is the only output your code will produce in the version you turn in.

```
dictionary size 133168
Done checking and these are the results
finished in time: 5.62
There are 950068 words found in the dictionary
     15186771 compares. Average: 15
There are 27075 words NOT found in the dictionary
     441201 compares. Average: 16
There are 2544 words not checked.
```

Run time for extra credit.

To compete for extra credit, first the program must run correctly and meet all the specification above.

- 5 gears for finishing in under 7 seconds on the Pi device.
  - Roughly 1 second or less on the Linux systems, but all timings will be on the Pis
- Finally, 10 addition gears if you can beat my time. Which is listed in the format section. On the Linux system, it ran in .8 seconds.

Turn in:

Hard copy:

A cover page with the following information in large font at the top of the page

Cosc 2030
Program #2
your Name

Repo name
Competing: YES or NO

At the bottom of the page, include a non-empty statement of help delivered and help received. It is OK to state that no help was given or received. It is **NOT** ok to omit the statement of help.

Soft Copy:

1. Copy ONLY any .cpp and .h files to the repo. Use this link to create the repo: https://classroom.github.com/a/oIjH4A3p
2. Edit readme.md file, add the following:
    - Name
    - Competing: YES or NO
        - Also list your best run time
    - How to compile it on the linux systems if not competing or Pi systems if competing
    - List anything that doesn't work (that you know of)
3. Lastly, verify all the necessary files are on the github website. If the files are missing then you **DID NOT** turn it in.