Cosc 2030 Program 3
Due: Apr 27 100 points

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

You will be given a dictionary, in which the words are in 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 the file in word by word is easy. The dictionary will be entered into the hash data structure you wrote.

The book has almost a million words in it! This will be a bit more complicated, since the words read in may contain ASCII characters that are not in the dictionary. So, you will have 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 using >> instead of getline, since it will read based on spaces. You will need to write a "clean word" function that will any unnecessary ASCII characters from the word, see specs below. You will need to use the clean word method to sanitize books' words and the dictionary words (before adding them to the dictionary data structure). You will also be timing this as well.

"Clean word" method:

- This must be implemented as a separate method and in the same file as the main method, so it can be called with 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 be 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 hash data structure class and it must be a template class. It will be in its own file, likely myHash.h file. You cannot use the STL data structures to hold the dictionary and specifically the hash, map, order, pair, hash may not even be included. You can however use the rest of the STL methods and functions in other places. The hash must use a simple array, it cannot be a vector or other class.
2. "Clean word" method must be 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 is 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 starts at the point where you open the book file.
4. Output the following information. See the next page for required format of the output.
   - Time to spell check in seconds
   - Time to spell check in milliseconds
     Note, the time class has TimeMS() methods to get milliseconds.
   - 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. You must write out a file of all the misspelled words (every instance of the misspelled word too) to a file called misspelled.txt Print each word on a separate line. They should be printed in the order they were found. This should be done after the timer has been stop, otherwise if will affect your runtime.
6. The insert/find/getSize/isEmpty functions must be in the hash class and must use these names. You can NOT use an insert/find/getSize/isEmpty functions that are out of the class.
7. The collision resolution must be either separate chaining (linkedlist or tree) or open addressing (linear probing, quadric probing, or double hashing). You cannot make up your own resolution.
8. Your code must be commented and have good coding style (variable naming, indention, etc) and structure. Points will be taken for poor structure and bad code style.
9. No matter what else you do, the must be a string.compare operation to determine that the word matches a word in the dictionary. You cannot skip the compare.

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. Your program may not use any external information about the dictionary, other information provided in the dictionary itself.
3. You may not even include the following STL libraries in any of your code: map, pair, hash, unordered_map, and unordered_set
4. The dictionary data structure must be ONE complete data structure; you cannot, for example, declare 5 separate data structures for the dictionary in the main code. This would be done inside the dictionary data structure itself.
5. The Clean word method MUST be a separate method and may NOT be part of the dictionary data structure.
6. 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, nowhere can you hard code any ASCII table numbers into your code.
7. You must use one of the collision methods as covered in class, you CAN NOT make up your own collision method. Separate Chaining, linklists or tree and Open addressing Linear, Quadratic, or double hash are the only options in the slides. If you choose to use separate chaining, then you MUST use your code (fixed as necessary) from either program 1 (linklist) or program 2 (tree).
8. Comparing strings with the == anywhere in your code, this includes !=, >, >=, <, and <=.
9. For run time, the Dictionary should take no longer than a minute or two to insert the dictionary into the hash.

Output section: No other output, besides what is below, numbers in RED are expected to be different, but use the same formatting. Removing ALL 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: 3.22
finished in Milliseconds Time: 3225.00
There are 950068 words found in the dictionary
     7336673 compares. Average: 7
There are 27075 words NOT found in the dictionary
     100047 compares. Average: 3
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 specifications above**. The extra credit will be applied to the final exam.

- 2 extra credit points for finishing in under 4.0 seconds on the Pi device.
- Finally, 3 addition extra points if you can beat my time. Which is listed in the format section. On the Linux system, it ran in 1.2 (hive) OR 0.4 seconds, or 486 milliseconds on cslab or fish machines.

Turn in:

Hard copy: (THIS MUST BE TYPED)

A cover page with the following information

Cosc 2030
Program #3
your Name
Repo name
Competing: YES or NO

in large font at the top of the page. 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/CuabP-ro
   - DO NOT INCLUDE executable programs and output files. Only the .cpp and .h/.hpp files. (Or any. vscode files).
2. Edit readme.md file, add the following:
   - Name
   - Competing: YES or NO

    - And 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.