Department of Computer Science

CS260 Data Structures

Winter 2022

Programming Assignment 3

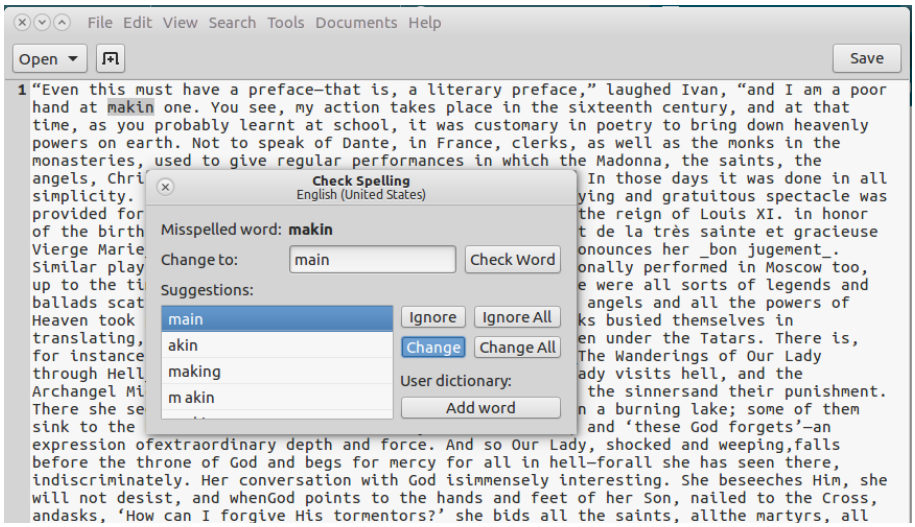
1 Regulations

Due Date : 2/11/2022, Friday, 11:59pm
Late Submission: 10% off for each late day, with at most 2 day late submission
Submission : via Gradescope.
Team : The homework is to be done and turned in individually. No teaming up.
Cheating : All parties involved in cheating get zero from assignment and will be reported to the University.

2 Spell Checker

A spell checker is a software feature commonly found in text processors that checks for misspellings in a text. It is also a useful feature provided in most email and texting applications.

A basic spell checker compares each word with a known list of correctly spelled words from a dictionary. In the case that the dictionary contains only root words, spell checker can further compare for different forms of the word, such as plural/singular for names, past/present form for verbs etc. if the query word is not found in the dictionary as it is. If the word and its derivations are not found in the list, the word is flagged as misspelled. In several forms of the spell checker, the system further suggest alternative words to replace the misspelled word, by evaluating common typos. Finally, it can also suggest whether the user wants to add the word into the dictionary.



Spellchecker of gedit, the default text editor of the GNOME desktop environment

The performance of spell checker is crucial, where the major operation that is repetitively done is **searching** for words in the dictionary. Considering that the there are 171,146 words currently in use in the English language, according to Oxford English Dictionary, it will be very time consuming to search through the entire dictionary for each word in a provided text along with its possible variations. Since there will be multiple search operations for each word of the text file, we need a data structure that will provide fast access to words in the dictionary. Hash tables are perfect tools for efficiently accessing dictionary elements!

3 Problem

In this assignment, you will write a C program that will implement a spell checker by using hash tables as the underlying data structure. You are expected to use Open Hash Tables in your implementation. You can come up with any hash function. Refer to the slides for an example hash function to be used on strings.

For this assignment, your program needs to accomplish the following for you to get full credit:

- **Load the dictionary into a hash table**. You should use Open Hash Tables where you'll implement linked lists for each entry of the hash table.
- You will **search for a given word** in the dictionary and flag it as misspelled if it does not exists in the dictionary (i.e., no need to check for singular/plural, past/present tense etc. variations).
- If the word is flagged as misspelled, then **check for potential typographical mistakes** (inverted adjacent letters (ex: word vs wrod), words with a missing (word vs wor) or extra (word vs worid) letter at the beginning or at the end, and suggest valid alternative spellings from the dictionary, if they exist.
- If the user chose to insert the misspelled word into the dictionary (which user will tell when running the program initially), your program should be able to update the dictionary by inserting the new word into the hash table.

4 Starter Package

- Along with this assignment instructions document, you will be provided with:
  - a **starter code** that handles reading the input dictionary and the input text to be spell checked,
  - a sample **dictionary** file that contains one word each line, and a sample **text input** file that you can use for testing your spell checker,
  - a simple **makefile**, that compiles the code to generate an executable
  - an **executable autocomplete program**, which you can compare against your program to match input/output specifications.
  - a starter **report file in LaTeX**, which you will fill with the analysis of your assignment.

5 Input/Output Specifications

- **Your executable program** (which should be named as "check") is going to take three command line arguments:
  1. the name of the dictionary file, which contains the correct writings of words
  2. the name of the input text file, contents of which you will spell check
  3. the word *add* or **ignore**, indicating whether to add misspelled words into the dictionary or not.

  Thus, an example call to your program will be as either one of the following:

  ```
  $pa3/> ./check dictionary.txt inputText.txt add
  $pa3/> ./check dictionary.txt inputText.txt ignore
  ```

- **Input dictionary file** contains one English word in each line. A sample dictionary file is provided for your reference in the starter package.
- **Input text file** contains some plain text written in English. A sample input text file is provided for your reference in the starter package.
  - Note that, since the dictionary that we will use contains words that exclusively start with lowercase letters, you can assume that the input text will always consist of lower case letters.
  - Also note that, the only non letter characters in the input test will be white space, comma, dot, colon, semicolon, exclamation mark, and new line. The provided starter code includes a line that splits the input text into words by these delimiters.
- **Updating the dictionary** is an all-or-none behavior for your program. If the user has run your program with the third command line parameter being **add**, then you should insert all misspelled words that your spellchecker detects into the dictionary. On the other hand, if the user has run your program with the third command line parameter being **ignore**, you shouldn't insert any new word into the dictionary.

  Thus, as an example, if the input text contains the same misspelled word twice and the program was run with *add* parameter, you should not tag the second appearance of the word as misspelled, as that word would have been inserted into the dictionary after its first appearance in the text, making the second appearance a valid word. On the other hand, if the program was run with *ignore* parameter, then each appearance of the same misspelled word should be reported (details of which will be listed below).
- Once the user enters these three command line parameters and runs your program, your program should load the dictionary into memory, read the input text word by word, and **make a search for each word** in the dictionary.
  - **If it finds the word in the dictionary**, it will proceed to the next word until it processes all of the words in the input text. If the input text does not have any misspelled word (i.e., all words in the input text are found in the dictionary), your program should print:

    No typo!

    to standard output (with a new line after exclamation mark) and quit the program.
  - For each of the words that are **not found in the dictionary**, your program should print the word to the screen and notify the user that the word is misspelled. For example, if the incorrect word was "spor", you should print:

    Misspelled word: spor

    to standard output (with a new line after the misspelled word).
  - For the words that are not found in the dictionary, your program will then **suggest alternative words that might be the correctly spelled versions of the misspelled word**. For this assignment, you are expected to check for the following three common typing errors:
    1. **inverted adjacent letters** (ex: 'wrod' would be an incorrect spelling of 'word' with inadvertent adjacent letters)
    2. words with **a missing letter at the beginning OR at the end** (ex: 'wor' would be an incorrect spelling of 'word' with a missing last letter)
    3. words with **an extra letter at the beginning OR at the end** (ex: 'wordi' would be an incorrect spelling of 'word' with an extra letter at the end)

    Although a real spell checker does more than that, implementing this much will suffice for the purposes of this assignment. When checking for missing/extra letters, only check for the 26 English letters. For the incorrectly typed word "spor", for example, your program's output to stdout should be as follows:

    Suggestions: spore, sport, por

    If alternate typings of the misspelled word are not found in the dictionary, your program should print:

    Suggestions:

that is, it should print an empty statement as above.

- Finally, **if** the user have called the program with the third command line parameter as *add*, then you will **insert the misspelled word into the dictionary** that you stored inside the hash table. **If** the third command line parameter was *ignore*, then you do not make any change to the internal dictionary that you hold in your hash table.

- Thus, you will be graded on whether your program can;
  - search for an already existing word in the dictionary,
  - suggest alternative words that would be possible correct spellings of the misspelled word, and
  - add a new word to the dictionary.

- **Output** will be printed to standard output and needs to strictly satisfy structure quoted above. Make sure you double check that your output matches that which is being produced by the provided executable file.

- **Report file** should explain the details of data structure you used for hashing, and contemplate about its running time, your observation about its behavior for different inputs, what is good/bad about your algorithm and also your implementation, how much it might be improved if you used different methods. You will be provided with a starter LaTeXfile for your report. You should answer the questions in there, and can add more information if you like.

6 Submission

- Your code **must be running on tux**. So, make sure that it compiles and runs on tux before you submit, even if you develop it elsewhere.

- Before submitting your code, compare the outputs of your program for various test data with that of the provided executable. If there exists differences in the outputs, that would lead to your program failing in test cases while grading. Make sure you match the input/output specifications.

- Your submission package **must include** your **source code**, a **makefile**, and a **report file in pdf format** that explains your approach for solving the problem. Put all these material in a folder named with your userid, (example: abc123), compress it into a zip file named as your user id (example: abc123.zip). Submit this zip file through Gradescope.

- Your makefile needs to produce an executable named "check", once we type "make" in command line.

7 Grading

- Assignment is going to be graded out of 100 points.

- You will provide your self assessment for your effort for the 40 point portion of the assignment.

- 40 point portion of the remaining part will be for the test cases of the assignment.

- The final 20 point will be for the report file.

- There will be **10 point penalty** if you do not follow submission guidelines.