# Lab 6: Hash Table

Deadline: see calendar

**Objective**: To have an opportunity to practice class template hash_map/unordered_map. Additionally, to have a better understanding of string operations and class template.

**Description**: One natural use of a hash table is to hold the words in a spelling dictionary. This makes looking up the words very efficient. In this assignment, you will implement and use such dictionary.

Your main program will behave like the standard Linux utility program *ispell*, when that program is used interactively to check individual word. To try it, enter the command *ispell* on the command line. You will be prompted to enter words. When you type a word, there are three possible responses: An output of "ok" means that the word is in the dictionary. An output of "not found" means that the word is not in the dictionary and neither are any near misses. Otherwise, the output is a list of near misses, that is, words that are in the dictionary and are similar to what you typed. The *ispell* program recognizes several types of near-misses: single-letter omissions, single-letter additions, transpositions of two neighboring letters, substitution of one letter for another, and two words run together. Your program should do the same.

**Design:** Create an appropriate hash table from the class template hash_map/unordered_map, and insert all words of a dictionary which is specified by the file **dict.txt** in the folder of source files. This file contains 88984 words, with one word per line.

Your program should prompt the user to enter words. For each word, it should check whether the word is in the dictionary. If so, it should say "ok". If not, it should look for all possible near misses. If the program finds any near misses in the dictionary, it should print them. If not, it should say "not found".

One issue that you will have to settle is what to do with upper-case letters. Note that both the words in the dictionary and the user's input can contain upper case letters. I suggest that you simply convert all letters to lower case.

Searching for near misses will require a significant amount of programming. You will have to generate every possible near miss and look for it in the dictionary. Specifically:

- Construct every string that can be made by deleting one letter from the word. (n possibilities, where n is the length of the word)
- Construct every string that can be made by inserting any letter of the alphabet at any position in the string. (26*(n+1) possibilities)
- Construct every string that can be made by swapping two neighboring characters in the string. (n-1 possibilities)

- Construct every string that can be made by replacing each letter in the word with some letter of the alphabet. (26*n possibilities (including the original word n times, which is probably easier than avoiding it))
- Construct every *pair* of strings that can be made by inserting a space into the word. (n-1 pairs of words; you have to check separately in the dictionary for each word in the pair)

## Requirement:

- You must create the hash table from the class template hash_map/unordered_map.
- You must create a class *WordChecker* to wrap all functions that generate all possible near misses.
- list the near misses in alphabetical order and each near miss must be listed only once.

**How to download assignment files:**
Go to the URL: http://www.cs.mtsu.edu/~zdong/3110/public/OLA/ and download the file OLA5.zip, which contains the following files:

- Ola6Description.doc: this file
- OLA6Rubric.doc: a rubric used to grade this assignment
- OLA6.exe: an executable solution provided by the instructor.
- Dict.txt: the dictionary

**How to submit your Lab:**

- **softcopy:**
  - Commit to your repository.
- **Hard copy**:
  - print all source files
  - Enclose the hardcopy of the program and the rubric in a folder (at least 9''x12''), and put C# (the one I give you), your name, section #, instructor name on the folder. (**Note**: You can buy folders from computer lab.)