Base du Développement Logiciel

20 janvier 2022

Une *anagramme* est une permutation des lettres d'un mot pour obtenir un nouveau mot. Par exemple, les mots *onirique* et *ironique* ayant les mêmes lettres avec le même nombre d'occurrences (un *n*, deux *i*, etc), ce sont des anagrammes l'un de l'autre.

1 Détection d'anagramme

Définissez une fonction `std::string normalize(std::string const &)` ayant la spécification suivante : la chaîne renvoyée doit être la même pour toutes les anagrammes du mot passé en argument ; elle doit être différente pour deux mots qui ne sont pas des anagrammes. Par exemple, `"eiinoqru"` est une forme normale convenable pour *onirique* et *ironique* (d'autres façons de normaliser sont possibles). Indice : jeter un oeil à `std::sort`.

Utilisez cette fonction pour définir une fonction prenant deux chaînes de caractères en argument et renvoyant `true` si ces deux chaînes sont anagrammes l'une de l'autre.

Testez votre fonction sur quelques exemples.

2 Chargement de mots

Définissez une fonction `std::vector<std::string> load()` qui lit le fichier `words.txt` fourni avec le sujet et renvoie un vecteur de chaînes de caractères contenant un mot par case. Vous utiliserez préférentiellement `std::getline(std::istream&, std::string &)` pour effectuer la lecture du fichier depuis un `std::ifstream`.

Testez votre fonction en affichant la longueur du vecteur (le nombre de mots lus) ainsi que le dernier mot. (Note : le *zizzyva* est un coléoptère tropical.)

3 Construction de dictionnaire

L'objectif est maintenant de construire un dictionnaire permettant d'inverser la fonction `normalize`. Autrement dit, pour toute chaîne `s`, une expression ressemblant à `dict[normalize(s)]` doit permettre de retrouver la valeur `s`. Utilisez pour cela le conteneur `std::multimap` qui est une table de hachage à valeurs multiples :

```
typedef std::multimap<std::string, std::string> dictionary;
```

Question : pourquoi est-ce que `std::map` ne convient pas ?

Définissez une fonction `dictionary convert(std::vector<std::string> const &)` qui crée et remplit un dictionnaire à l'aide des mots passés en argument.

Note : `std::multimap::insert` prend en argument une paire formée de la clé et de la valeur associée.

4 Liste des anagrammes d'un mot

Définissez une fonction qui renvoie toutes les anagrammes d'un mot (excepté lui-même) présentes dans un dictionnaire :

```
std::vector<std::string> anagrams(dictionary const &, std::string const &);
```

Note : la méthode `equal_range` de `std::multimap` renvoie une paire d'itérateurs qui délimitent la zone contenant toutes les entrées d'un dictionnaire ayant la même clé.

Testez votre fonction en affichant les anagrammes des mots suivants (ou un message d'erreur le cas échéant) :

```
char const *test_words[] =
  { "anagram", "parrot", "abba", "insert", "silent" };
```