L2 Informatique

TP Algorithmique – Listes, Révisions

Fréquence d'apparition des mots et des lettres dans un texte

Ce TP consiste à écrire un programme en C++ permettant de compter le nombre d'occurrences de mots et de lettres dans un texte, parmi d'autres fonctionnalités.

Le programme nécessitera la déclaration des types suivants :

- `occMot` : enregistrement composé d'un mot (chaîne de caractères) et d'un nombre d'occurrences (entier) ;
- `listeMots` : liste chainée de variables `occMot` ;
- `tabMots` : enregistrement composé d'un tableau de mots distincts (chaînes de caractères) et de sa taille ;
- `occLettre` : enregistrement composé d'une lettre (caractère) et d'un nombre d'occurrences (entier) ;
- `tabLettres` : tableau de 26 variables de type `occLettre`.

Les deux derniers types ne seront utilisés que dans la partie 2.

Le programme peut être testé en utilisant le fichier texte **bouledesuif.txt** disponible sur l'espace moodle. Il ne contient que des mots bruts en minuscule et ne comporte pas de caractère accentué, apostrophe, tiret ou signe de ponctuation.

Partie I

Écrire les sous-programmes suivants. Les différents types devront avoir été préalablement définis.

1. `void initialise (listeMots & L)` : crée une liste vide de mots.
2. `void ajouteDebut (std::string mot, listeMots & L)` : ajoute un mot en début de liste ; L étant une liste de `occMot`, pensez à initialiser le nombre d'occurrences du mot ajouté.
3. `void ajoute (std::string mot, listeMots & L)` : ajoute à la bonne position un mot dans une liste que l'on suppose déjà triée (selon le classement alphabétique des mots) ; si le mot apparait déjà dans la liste, on incrémente son nombre d'occurrences ; la liste doit rester triée après l'ajout.
4. `void affiche (listeMots L)` : affiche la liste des mots, avec leur nombre d'occurrences entre parenthèses.
5. `int taille (listeMots L)` : calcule la taille de la liste de mots (nombre de mots différents).
6. `int nombreMots (listeMots L)` : calcule le nombre de mots total du texte associé à la liste de mots (toutes les occurrences d'un même mot sont comptabilisées).
7. `bool estValide (listeMots L)` : détermine si la liste de mots est valide, c'est-à-dire que chaque mot n'apparait qu'une fois dans la liste, que celle-ci est triée, et que chaque mot a au moins un nombre d'occurrences de 1.
8. `int plusLong (listeMots L)` : détermine la longueur du mot le plus long de la liste.
9. `void afficheLongueur (listeMots L, int longueur)` : affiche les mots de la liste ayant une longueur spécifiée en paramètre.
10. `void afficheTriLongueur (listeMots L)` : affiche tous les mots de la liste, du plus court au plus long. Il n'est pas demandé de retrier la liste.
11. `void saisit (listeMots & L)` : demande à l'utilisateur de saisir un ensemble de mots et les ajoute dans la liste L.

12. `void construit (listeMots & L, std::string nomFichier)` : construit une liste de mots en fonction d'un texte contenu dans un fichier texte dont le nom est donné en paramètre.

13. `void remplit (tabMots & T, listeMots L)` : remplit un tableau mots T à partir d'une liste L ; le tableau comportera, dans l'ordre, la liste de tous les mots de L (chaque mot sera ainsi en un seul exemplaire, sans mention du nombre d'occurrences).

14. `void affiche (tabMots & T)` : affiche un tableau de mots.

15. `bool appartient (std::string mot, tabMots T)` : détermine si un mot appartient à un tableau de mots trié T ; vous devrez utiliser la recherche par dichotomie au moyen d'un sous-programme récursif et ainsi définir un sous-programme annexe permettant d'appliquer cette récursivité.

16. Concevoir le programme principal de manière à :

    - construire une liste de mots à partir d'un fichier texte ;
    - afficher la liste des mots du texte (qui sera normalement classée alphabétiquement), avec leur nombre d'occurrences ;
      → *pour tester, remplacer cet affichage par celui de la liste classée par longueur des mots*
    - demander à l'utilisateur un mot, puis afficher s'il apparait dans le texte ou non.

Partie II

Compléter le fichier avec les sous-programmes suivants afin d'ajouter des traitements sur les lettres :

17. Déclarer les types `occLettre` et `tabLettres`.

18. `void initialise (tabLettres & T)` : initialise un tableau d'occurrences de lettres ; les 26 variables T[i], de type `occLettre`, seront composées d'une lettre minuscule (respectivement 'a', 'b', 'c', . . . , 'z') ; le nombre d'occurrences de chaque lettre sera initialisé à 0.
    → *Astuce : on peut incrémenter une variable de type char de la même manière qu'un entier ; par exemple, après les instructions char c = 'a' ; ++c ; la variable c vaut 'b'.*

19. `void ajoute (char lettre, tabLettres & T)` : ajoute dans un `tabLettres` une occurrence d'une lettre donnée en paramètre.

20. `void comptabilise (tabLettres & T, listeMots L)` : remplit un `tabLettres` en fonction d'une liste de mots ; T contiendra à la fin de la procédure le nombre d'occurrences de chacune des 26 lettres de l'alphabet présentes dans la liste de mots L ; lorsqu'un mot est associé à plusieurs occurrences dans L, chacune de ses lettres devra être comptabilisée autant de fois.

21. `void trie (tabLettres & T)` : trie le tableau T par ordre décroissant du nombre d'occurrences des lettres.

22. `void affiche (tabLettres T)` : affiche les lettres de T suivies, entre parenthèses, de leur nombre d'occurrences (dans l'ordre où elles apparaissent dans le tableau).

23. Compléter le programme principal afin de :

    - calculer un tableau d'occurrences de lettres en fonction de la liste de mots, puis le trier dans l'ordre décroissant du nombre d'occurrences des lettres ;
    - afficher le nombre d'occurrences des 26 lettres de l'alphabet dans le texte.