Projet Dictionnaire – IN 301 Langage C

Table des matières

Le but de ce projet est de stocker tous les mots d'un texte dans un dictionnaire (sous forme d'ABR) puis d'être capable pour un mot donné d'afficher son nombre d'occurrences et les lignes (numéro et contenu) où il est présent.

Définition : Un **mot** est une suite de caractères qui sont des lettres minuscules (de `'a'` à `'z'`) ou des majuscules (de `'A'` à `'Z'`) et qui est précédée et suivie de caractères autre que ces lettres. Un mot peut commencer ou terminer le fichier et donc ne pas avoir de caractère autre avant ou après. On ne traitera que des textes en anglais, on suppose donc qu'il n'y a pas de signes diacritiques.

Définition : Une **ligne** est une suite de caractères qui est comprise entre deux caractères de retour à la ligne (`'\n'`) et qui ne contient pas ce caractère. Il y a un début de ligne au début du fichier et une fin de ligne en fin de fichier.

1 Données d'entrée

Il vous est fourni un fichier texte `pg31469.txt` contenant le texte du livre, en anglais, « The Shunned House » (La maison maudite) de H.P. Lovecraft : http://www.gutenberg.org/cache/epub/31469/pg31469.txt. D'autres livres téléchargeables gratuitement sont disponibles sur le site :http://www.gutenberg.org.

2 Texte vers tableau

Le texte lu dans le fichier d'entrée doit être stocké dans un tableau `T` faisant exactement la taille du fichier en nombre d'octet donc de caractères.

Vous devrez en même temps construire un tableau d'entier `L`, tel que `L[i]` contienne la position du premier caractère de la `i`ème ligne dans `T`.

Attention : la numérotation des lignes commence à 1 et les indices du tableau commencent à 0. Gérez cela de manière intelligente pour éviter les bugs de programmation.

3 Texte vers ABR

La première partie du projet consiste à lire le fichier d'entrée et à stocker les mots dans un ABR `A` avec les propriétés suivantes :

— Chaque nœud de l'ABR contient un mot différent.
— Tous les mots du sous arbre gauche sont plus petits au sens de l'ordre lexicographique, que le mot du nœud.
— À droite ils sont tous plus grand.
— Chaque nœud doit contenir le nombre d'occurence du mot dans le texte.
— Chaque nœud doit contenir la liste des numéros de ligne qui contiennent le mot. Cette liste doit être parcimonieuse en mémoire.

Attention : Ne tenez pas compte de la casse des lettres, les mots `"mot"`, `"Mot"` et `"MOT"` sont le même mot. Par contre `"mot"` et `"mots"` ne sont pas le même mot.

Attention : Pour être efficace par la suite l'ABR doit être équilibré.

4 Recherche d'un mot

La recherche d'un mot dans le texte doit afficher, le nombre d'occurences et la liste des lignes : numéro et contenu.

Attention : Il peut y avoir deux fois le même mot dans la même ligne.

5 Temps d'exécution

Il vous est fourni une bibliothèque `chrono.c`, `chrono.h` qui vous donne accès à deux fonctions :

— `void chrono_reset()` qui lance le chronomètre ou le relance à zéro s'il était déjà lancé ;
— `float chrono_lap()` qui renvoie le temps écoulé depuis le dernier `chrono_reset()`.

Le temps renvoyé est exprimé en secondes avec une précision de 10^{-6} (microseconde).

6 Fonctionnement du programme

Le programme exécutable final doit s'appeler `dico`.

Sur la ligne de commande (ou dans le `Makefile`), si on tape :

— `./dico pg31469.txt shunned house uvsq`, le programme devra effectuer la recherche pour chacun des trois mots et afficher clairement les différents temps de calcul.
— `./dico pg31469.txt`, le programme devra choisir au hasard 1000 mots présents dans le texte et rechercher chacun de ces mots. Dans ce cas vous n'afficherez pas les résultats des recherches mais uniquement les temps de calcul.

Les temps de calcul sont les différents temps pour lire le fichier, construire l'ABR, faire une recherche, faire 1000 recherches, temps total d'exécution.

7 Travail demandé

1. Programmez le projet
2. Il est interdit d'avoir des variables globales. Il peut être pertinent d'avoir une structure de données qui contient tout ce qui correspond à un texte : les tableaux `T` et `L` et l'ABR `A`.
3. Vous devez séparer votre code en plusieurs fichiers, par exemple un fichier pour la lecture du fichier texte et le stockage dans les tableaux `T` et `L`, un fichier contenant les fonctions de construction de l'ABR, un fichier contenant les fonctions pour la recherche, fichiers pour le chronomètre, ... et un fichier contenant le programme principal.
4. Comme déjà écrit, le programme exécutable doit s'appeler `dico`.

5. Sur la ligne de commande, on doit taper après le nom du programme exécutable, le nom du fichier source suivi du ou des mots que l'on veut rechercher. S'il n'y a aucun mot à recherche le programme devra choisir au hasard mille mots et les rechercher (sans faire d'affichage)
6. Fournir un fichier `Makefile` qui fonctionne.
7. La commande `make` sans argument doit lancer la compilation (si nécessaire) et l'exécution d'un exemple pertinent.
8. Vous devez fournir un fichier `NOM1_NOM2.zip` contenant un dossier appelé `NOM1_NOM2` et contenant uniquement : vos fichiers sources (`.c .h`), le Makefile et le fichier du texte source. `NOM1` et `NOM2` sont les noms de famille des deux memebres du binôme

8 Évaluation

— Le travail est à faire en **binôme**.
— Le fichier `zip` est à rendre sur moodle avant le **XX janvier 2021** (**cette date sera définie ultérieurement**).
— Il y aura une présentation orale avec zoom **en visio** le **Y janvier 2021** (**cette date sera définie ultérieurement**). Merci de vous munir obligatoirement d'une caméra et d'un micro pour le jour de la présentation.
— La date et les modalités de la présentation orale vous seront communiquées ultérieurement.
— **ATTENTION :** Le non respect d'une consigne :
  — nom de l'exécutable ;
  — nom du dossier ;
  — nom du fichier `zip` ;
  — non dépôt sur moodle ;
  entraînera une pénalité de 5 points par consigne non respectée, sur la note finale du projet.
  La notation sera de 10 points pour le code rendu sur moodle et de 10 points pour la présentation orale, et donc toute absence à la présentation orale vaudra 0/10 à cette partie..

9 Ce qui vous est fourni

Il vous est fourni deux fichiers :
— `chrono.zip` qui contient un dossier avec `chrono.h`, `chrono.c` et un exemple d'utilisation.
— `taille_fic.zip` qui contient le fichier `pg31469.txt` et un programme donnant deux méthodes pour récupérer la taille d'un fichier.

10 Travail facultatif en plus

Vous pouvez implémentez les fonctionnalités suivantes en plus de ce qui est demandé :
— Faire fonctionner le programme avec des fichiers sources en langue française et donc avec des signes diacritiques.
— Faire une recherche multi-mots, c'est à dire trouver les lignes qui contiennent simultanément un ensemble de mots.
— Gérer les options sur la ligne de commande en utilisant la fonction `getopt` et du coup permettre ainsi les différentes fonctionnalités à sélectionner avec une option. Par exemple `-t` suivi d'un nombre K pour faire un test de temps d'exécution sur K recherches.