

Bases de la Programmation

TP 8 – Fichiers

V. Padovani, PPS - IRIF

Le but de ce TP est d'implémenter un ensemble de fonctions permettant d'extraire d'un fichier texte son *lexique*, c'est-à-dire l'ensemble de mots.

Un lexique sera représenté en mémoire à l'aide d'un tableau d'adresses de chaînes de caractères. L'ajout de mots au lexique se fera en inscrivant dans ce tableau, à partir de la position 0, les adresses de zones-mémoire allouées par `malloc` et contenant les chaînes formant ces mots.

Le tableau d'un lexique ainsi que son nombre courant de mots seront regroupés dans une structure définie comme ci-dessous, où `CAPACITE` est une constante suffisamment grande :

```
typedef struct {
    char *mots[CAPACITE]; // tableau des adresses de mots
    int   nbr;             // nombre courant de mots, i.e. 0 pour un lexique vide
} lexique;
```

Chaque fonction ci-dessous suppose que `l` est l'adresse d'un lexique et que `s`, `t` sont celles de chaînes de caractères. Chacune peut, ou même doit se servir des précédentes. Notez que l'usage de Google pour ce TP sabordera votre processus d'apprentissage ; pour les mêmes raisons, je vous suggère de consulter le cours le moins possible pour le résoudre.

Exercice 1 : Ajout et libération

Ecrire les fonctions suivantes. N'oubliez pas de tester (4) et (5) à l'aide de (1).

1. `void afficher (lexique *l)` affichant, sur des lignes successives, la suite des mots d'un lexique.
2. `int longueur (const char *s)`, renvoyant la longueur de la chaîne d'adresse `s`¹.
3. Écrire `char *clone (const char *s)`. Cette fonction doit allouer par `malloc` un espace mémoire exactement de la taille nécessaire pour y stocker une copie de la chaîne d'adresse `s`, y inscrire cette copie, puis renvoyer l'adresse d'allocation².
4. `void ajouter (lexique *l, const char *s)`. Cette fonction allouer et construire une copie de la chaîne d'adresse `s`, puis ajouter cette copie au lexique d'adresse `l`.
5. `void vider (lexique *l)`. Cette fonction doit libérer tous l'espace-mémoire alloué pour le stockage des mots du lexique d'adresse `l`.
6. `void liberer (lexique *l)`. En supposant que le lexique d'adresse `l` a lui-même été alloué par `malloc`, cette fonction doit libérer la totalité l'espace-mémoire alloué pour le stockage du lexique et son contenu.

1. Une fonction de la librairie standard effectue le même traitement, `strlen`, mais le but de cet exercice est de retrouver son implémentation.

2. Là encore, une fonction de la librairie standard effectue le même traitement, `strcpy`, et il s'agit de retrouver son implémentation.

Exercice 2 : Consultation et suppression

Ecrire les fonctions suivantes et testez-les.

1. `int egales (const char *s, const char *t)` renvoyant 1 si les chaînes d'adresses `s` et `t` ont la même suite de caractères, et 0 sinon³.
2. `int position (lexique *l, const char *s)` renvoyant, si le lexique d'adresse `l` contient un mot de même suite de caractères que la chaîne d'adresse `s`, la position de son adresse dans le tableau du lexique, et une valeur négative sinon.
3. `int suppression (lexique *l, const char *s)`. Cette fonction doit supprimer du lexique le premier mot de même suite de caractères que la chaîne d'adresse `s` si un tel mot existe⁴, en libérant l'espace alloué pour ce mot. S'il n'existe pas, elle devra renvoyer une valeur négative.

Exercice 3 : Extraction

Par “lettre” on entend ici tout code de caractère égal à celui d'une lettre minuscule ou majuscule non accentuée – l'exercice n'a pas à gérer le problème des accents, ni à considérer comme égales les formes majuscule et minuscule d'une même lettre. Par “mot”, on entend tout suite de codes de caractères formée exclusivement de ces “lettres”.

Écrire les fonctions suivantes. Les fonctions (2) et (3) supposent que `fr` est un point d'accès à un fichier-texte ouvert en écriture. Testez la dernière sur un fichier contenant quelques mots à l'aide de la fonction (1) de l'exercice 1.

1. `int lettre (char c)` renvoyant 1 si `c` est une lettre, et 0 sinon.
2. `int debut_mot (FILE *fr)` renvoyant la première lettre rencontré à partir de la position courante dans le fichier, ou `EOF` si cette lettre est introuvable.
3. `int lire_mot (FILE *fr, char *buffer)`. Cette fonction suppose que `buffer` est l'adresse d'un tableau suffisamment grand. Elle doit, à partir de la position courante dans le fichier :
 - chercher la première lettre d'un mot,
 - stocker dans le tableau la chaîne formant ce mot,
 - renvoyer le nombre de lettres du mot lu.

Si un tel mot est introuvable, la fonction renverra simplement 0.

4. `lexique *lire_lexique (const char *nom_fichier)`. Cette fonction suppose que la chaîne d'adresse `nom_fichier` forme le nom d'un fichier-texte présent dans le répertoire d'exécution. Elle doit :
 - ouvrir un point d'accès en écriture au fichier,
 - allouer un lexique par `malloc`,
 - ajouter au lexique un seul exemplaire de chaque mot du fichier,
 - enfin, renvoyer l'adresse d'allocation du lexique.

3. La fonction `strcmp` de la librairie standard effectue un test comparable mais avec une convention de retour différent : -1, 0 ou 1 si la chaîne d'adresse `s` est respectivement plus petite, égale ou plus grande que celle d'adresse `t` pour l'ordre lexicographique – celui du dictionnaire.

4. Si aucun mot n'a été ajouté deux fois au lexique, il s'agit donc de la suppression du lexique de l'unique exemplaire d'un mot.