MMFAI
Cours de programmation

TP numéro 4
semaine du 2 novembre 1998

Manipulation de Fichiers et de Chaînes de Caractères (Anagrammes)

1 Manipulation de fichiers

Nous souhaitons nous constituer ici une librairie de fonctions utiles pour travailler sur des fichiers *de taille arbitraire* que nous voulons considérer comme vecteurs de *lignes*, ce qui est une suite de caractères ne contenant pas le caractère spécial `'\n'` (fin de ligne) et terminée par `'\n'`.

Nous allons représenter en mémoire des lignes par des chaînes de caractères.

1.1 Les chaînes de caractères

En C une chaîne de caractères est à son tour un vecteur d'éléments de type `char`, donc on peut déclarer ces objets de differentes façons:

```
char *s; /* pointeur sur le debut de la chaine, sans allocation */

char *s = "majoliechaine"; /* pointeur sur le debut de la chaine,    */
                           /* allocation implicite et initialisation */

char s[100]; /* pointeur sur le debut de la chaine, allocation de 100 cases */

char s[100] = "monjolivecteur"; /* pointeur sur le debut de la chaine, */
                                /* allocation de 100 cases et initialisation */
```

Attention: certains compilateurs, dont `gcc` considèrent la deuxième declaration comme une declaration de *constante*, donc vous ne pourrez pas écrire ensuite dans la chaîne `"majoliechaine"`, à moins de compiler votre programme avec des options cryptiques telles `-fwritable-strings` (à temps perdu, faire man gcc pour plus d'infos).

Il y a un caractère spécial, `'\0'` qui indique la fin d'une chaîne de caractères, et il faut bien s'assurer que toutes nos chaînes sont bien terminées ainsi dans nos programmes.

1.2 Les fichiers

Un fichier est une structure de donnée persistente qui contient une suite de octets. Pour manipuler un fichier, on doit savoir au moins l'ouvrir, le refermer, le creer, le detruire et lire et écrire son contenu.

En C, toute ces opèrations se font en utilisant des pointeurs sur le type de donnée predéfini `FILE`. Par exemple, le fragment de code suivant ouvre le fichier `t`, lit le premier caractère et referme le fichier (notez le traitement des erreurs).

```
FILE *pf;
char c;

    if ((pf=fopen("t","r"))==NULL) {printf("Erreur en ouvrant t.\n"); exit(1);};

    if ((c=getc(pf))==EOF) {printf("Fichier trop court\n");exit(2);};

    fclose(pf);

    /* reste du programme */
```

Exercice 1.1 (readline) *Écrivez une fonction* `int readline(FILE *fp, char **res)` *qui lit sur le fichier (déjà ouvert) pointé par* `fp` *une* ligne *terminée par un* `'\n'` *et la mémorise dans un vecteur de charactères terminé par* `'\0'`*. Le résultat sera 1 si le fichier n'est pas fini, et 0 si on est arrivé à la fin du fichier.*
Par contre, le pointeur sur la ligne lue sera renvoyé à l'appellant dans le deuxième paramètre, (et donc on est obligé à demander que `res` *soit l'adresse du pointeur de l'appellant, vu que C passe les paramètres par valeur).*
Tout ça est fait de façon à pouvoir écrire une boucle comme

```
char *buf;

while (readline(fp,&buf)!= 0)
  {
/* fait qque chose avec buf */
};
```

Attention: *comme nous sommes des programmeurs sérieux, notre fonction doit être capable de gérer des lignes de taille arbitraire. Utilisez donc la méthode d'allocation dynamique vue en cours (dont le coût amorti est linéaire).*

Exercice 1.2 (readfile) *Écrivez maintenant une fonction qui lit tout un fichier sous forme de vecteur de lignes. On veut donc* `int readfile(char *filename, char ***lignes)` *qui renvoie un code d'erreur entier (succés, erreur de lecture etc.) et qui ouvre le fichier de nom* `filename` *et rende dans* `*lignes` *le pointeur a un vecteur de lignes (i.e. à un* `char **`*).*
Encore une fois, faites de l'allocation dynamique.

Exercice 1.3 (optionnel) *Modifiez la fonction* `readfile` *de façon que si le nom de fichier est* `-` *l'opération soit effectuée sur l'entrée standard.*

Exercice 1.4 (Comparez deux lignes) *Écrivez une fonction de comparaisons de chaînes de caractères* `int comparelines(char **a, char **b)` *qui prend deux pointeurs sur des chaînes de caractères et les compare lexicographiquement. Le résultat doit être un entier positif, 0 où un entier négatif selon que la chaîne* `*a` *soit* *plus grande, égale ou plus petite que* `*b`.

Exercice 1.5 (Trier un fichier) *Écrivez un programme* `trie` *qui trie en ordre croissant un fichier dont le nom est passé sur la ligne de commande. (Oui, c'est vrai, on ne sait pas encore écrire une fonction de tri, mais cela n'est pas grave: faites* `man qsort` *et utilisez la fonction de librairie* `qsort`*).*

2 Anagrammes

En utilisant les fonctions de la section précédante, écrivez maintenant un programme qui lit un fichier contenant une liste de mots un par ligne (un dictionnaire), et donne sur la sortie standard tous et seuls les anagrammes contenus dans ce dictionnaire. Par exemple, si le dictionnaire est

```
inse'parabilite'
malle'abilite'
alte'rabilite'
bilate'ralite'
radiote'le'graphie
te'le'radiographie
```

on veut en sortie seulement les anagrammes

```
alte'rabilite' bilate'ralite'
radiote'le'graphie te'le'radiographie
```

Bien entendu, on vous demande de faire cela en temps moyen $O(nlog(n))$, en sachant que *qsort* démande en moyenne un temps $O(nlog(n))$.
Vous pouvez tester votre programme sur le petit dictionnaire français qui se trouve sur la machine `clipper` dans `~dicosmo/MiniDico` (ou aussi `~dicosmo/MiniDicoAccents` pour la version avec lettres accentuées).

3 Occurrence de mots dans un texte.

Un de vos copains de lettres vous demande si vous pouvez l'aider à trouver les 100 mots les plus utilisés par Racine dans ses oeuvres, qui viennent d'être digitalisées sur un CD-Rom. Expliquez comment résoudre efficacement ce problème en temps $O(nlog(n))$, et en effet, pour un k quelconque, non pas seulement 100.
Ensuite, réalisez votre algorithme à l'aide des commandes Unix `sort` `head` et `uniq`.

N'oubliez pas de vous déconnecter avant d'abandonner le clavier (voir notes du stage Unix).