TP5 – Chaîne de caractères et fichiers

Projet de programmation M1

20 Octobre 2015

On va écrier un programme qui affiche les lignes d'un fichier texte dans l'ordre alphabétique. Par exemple, si votre fichier contient

```
courgettes
aubergines
oignons
```

alors votre programme devra afficher

```
aubergines
courgettes
oignons
```

1 Préliminaires

1.1 Les chaînes de caractères

En C, il existe le type `char` qui représente un caractère sur 8 bits. Chaque caractère utilisé en anglais plus quelques autres est représenté par un entier selon le code ASCII. Par exemple, le code du caractère a est 97. On peut écrire `'a'` entre guillemets simple pour désigner l'entier qui représente le caractère a. On utilise `%c` avec `printf` pour écrire un caractère.

Les chaînes de caractères sont des tableaux de `char`. Pour marquer la fin de la chaîne, le dernier caractère est un caractère spécial : `'\0'`. Pour afficher une chaîne de caractères, on utilise `printf` avec `%s` pour désigner la chaîne de caractères. On les note entre guillemets doubles : `"hello"` désigne le tableau de `char` de taille 6 {`'h','e','l','l','o','\0'`}.

Exercice 1. Sachant que les codes des caractères '0' à '9' sont consécutifs, écrire une fonction `int is_int(char *s)` qui renvoie 1 si `s` est une chaîne de caractères représentant un nombre (positif ou négatif) et 0 sinon.

Écrire une fonction `int string_to_int(char *s, int b)` qui convertit une chaîne de caractères `s` représentant un nombre en base $b \leq 10$ vers un entier.

1.2 Les arguments de main

Sous Unix, quand vous appelez votre programme dans le terminal, vous pouvez lui donner des arguments. C'est ce que vous faites en appelant `gcc monfichier.c -Wall` : vous lui donnez l'argument `monfichier.c` et l'argument `-Wall`. Chaque argument est séparé par un espace.

On peut en C récupérer les arguments avec lesquels votre programme a été appelé. Pour cela, on remplace `int main()` par `int main(int argc, char **argv)`. La variable `argc` indique le nombre de paramètres passés à votre programme. Dans le cas de `gcc`, cette variable vaudrait 3 (une pour le nom du programme gcc, une pour le nom du fichier, l'autre pour `-Wall`). `argv` est un tableau de chaîne de caractères, c'est-à-dire un `char **argv`, de taille `argc+1` qui contient ces paramètres. Par exemple, si vous appelez votre programme ainsi : `./monprogramme.o mais allo quoi`, vous aurez `argc = 4` et `argv = {"./monprogramme.o", "mais", "allo", "quoi"}`. Vous n'avez donc qu'à parcourir le tableau `argv` pour récupérer les différents arguments.

Exercice 2. Écrivez un programme qui prend une suite d'entiers en paramètre et affiche la somme de ces entiers. Par exemple, si vous écrivez dans le terminal `./a.out 1 6 7 9`, le programme doit afficher `23`.

1.3 La manipulation de fichier

On peut utiliser les fonctions de la librairie standard pour lire/écrire dans les fichiers. Pour ouvrir un fichier en lecture seule, il faut utiliser la fonction `FILE *f = fopen("nomdufichier", "r")`. Cette fonction renvoie donc un pointeur vers une structure de données `FILE`. Imaginez ce pointeur comme un curseur vers un caractère du fichier. Pour lire le caractère sous votre curseur puis bouger le curseur vers le caractère suivant, on utilise la fonction `int fgetc(FILE *f)`. Si on est à la fin du fichier, cette fonction renvoie une constante nommée `EOF` (pour End Of File). Attention, cette fonction renvoie un `int` mais représente un `char` ! Il faut donc "caster" la valeur qu'on vous renvoie si vous voulez vous en servir comme d'un `char`. Par exemple :

```
int r = fgetc(f);
if (r != EOF) {
  char c = (char)r;
  printf("le caractere est %c",c);
}
```

Pour remettre le curseur au début du fichier, on peut utiliser la fonction `void rewind(FILE *f)`. Quand on a fini de travailler avec un fichier, il faut le refermer avec `fclose(FILE *f)`.

Exercice 3.

1. Écrire une fonction `int countlines(FILE *f)` qui compte le nombre de ligne d'un fichier. On se souviendra que le caractère "sauter une ligne" est le caractère '\n' en C.
2. Écrire une fonction `int maxlinesize(FILE *f)` qui renvoie la taille de la plus grande ligne du fichier.
3. Écrire une fonction `void readline(FILE *f, char *s)` qui écrit la ligne courante du fichier dans la chaîne de caractères `s`. Bien sûr, lorsque vous l'appellerez, il faudra avoir réservé assez de mémoire pour contenir toute la ligne !
4. Écrire une fonction `void readlines(FILE *f, char *s, char **t)` qui lit toutes les lignes du fichier et les stocke dans le tableau `t`. On pourra utiliser la fonction `strcpy(char *dest, char *src)` dans `string.h` qui copie la chaîne de caractères `src` dans `dest`.

1.4 La fonction de tri qsort

La fonction `qsort` permet de trier un tableau. Elle prend en argument un pointeur vers le premier élément du tableau, le nombre d'éléments du tableau, la taille (en octet) des éléments du tableau et un pointeur vers une fonction de tri. En effet, il est commode de pouvoir choisir sa fonction de tri, si on veut trier par ordre croissant, décroissant, taille de la chaîne etc. Pour en savoir plus, `man 3 qsort`. Dans ce TP, nous utiliserons la fonction de comparaison suivante, qui compare les chaînes de caractères selon l'ordre du dictionnaire :

```
int compare(const void* a, const void* b) {
  const char *ia = (const char *)a;
  const char *ib = (const char *)b;
  return strcmp(ia, ib);
}
```

Et nous l'appelerons ainsi, pour un tableau de chaîne de caractères `char[m][n] table` :

```
qsort(table, m, n, compare);
```

2 La fonction sort

Exercice 4. En réutilisant certaines des fonctions que vous avez écrites à la partie précédente, écrire un programme qui trie les lignes d'un fichier passé en paramètre de la ligne de commande et les affiche. Si aucun fichier n'est passé en paramètre, votre programme affichera un message d'explication sur l'utilisation de votre programme. Améliorez ensuite votre programme pour qu'il puisse prendre plusieurs fichiers en entrée (il faudra trier les lignes de tous les fichiers comme si c'était un seul).

Comment s'occuper pendant les vacances

Voici une liste d'exercices que vous pouvez faire pendant les vacances. Ils sont censés vous faire réviser tout ce que vous avez vu jusque-là. Si vous avez envie de challenges plus conséquents et plus mathématiques, vous pouvez jeter un œil au `http://projecteuler.net`.

2.1 Les yeux fermés

Exercice 5. Écrire une fonction qui calcule, étant donné N, la somme des entiers allant de 0 à N qui sont multiples de 3 et de 5 mais pas de 15.

Exercice 6. Écrire une fonction qui demande des entiers à l'utilisateur jusqu'à ce qu'il entre 0 puis calcule le pgcd de tous ces entiers.

Exercice 7. `https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Newton#Racine_carr.C3.A9e`

Écrire une fonction qui approxime la racine carrée d'un `float` grâce à la méthode de Newton, c'est-à-dire trouver un 0 de la fonction $f(x) = x^2 - a$ sur $\mathbb{R}_+$ (cette fonction est convexe, il n'y aura pas de problème de convergence).

2.2 Tableaux et chaînes de caractères

Exercice 8. [Secret Story] Soit $a = a_1 \dots a_n$ et $b = b_1 \dots b_n$ deux entiers codés sur n bits. On note $a \oplus b = ((a_1 + b_1) \mod 2) \dots ((a_n + b_n) \mod 2)$.

1. Expliquez comment on peut retrouver a à partir de $a \oplus b$ et de b.
2. Cette idée est à la base d'une technique de cryptage, les cryptage XOR. Si a, b sont deux `char`, c'est-à-dire codé sur 8 bits, on peut calculer $a \oplus b$ en C avec le xor bit à bit : `a^b`. Soit $M = m_1 \dots m_k$ un message de k caractère et $s = s_1 \dots s_l$ une clé secrète. On peut crypter le message M par $(m_1 \oplus s_1) \dots (m_i \oplus s_{(i \mod l)}) \dots (m_k \oplus s_{(k \mod l)})$. Implémentez une fonction `crypte(char *m, char *s)` qui crypte le message m avec la clé s (le résultat est stocké dans m). Comment décrypter avec cette fonction ?
3. Écrivez un programme qui prend en entrée une clé secrète, un fichier texte d'entrée et un fichier de sortie et écrit dans le fichier de sortie le fichier d'entrée crypté avec la clé secrète.

Exercice 9. [Voyage dans la matrix] Néo se trouve au-dessus de la première ligne d'une matrice $m \times n$ d'entiers. Il veut la traverser. Il peut rentrer dans la matrice par n'importe laquelle des n colonnes. Ensuite, il peut aller de ligne en ligne en allant soit juste en-dessous, soit en-dessous à droite, soit en-dessous à gauche. À chaque fois qu'il visite une case dont la valeur est $x \in \mathbb{Z}$, il doit payer x dollars (si $x < 0$, il en gagne bien sûr). Vous êtes Morpheus, vous devez aider Néo à traverser la matrix à moindre coût.

On pourra essayer de calculer $p_{i,j}$ qui est le coût du meilleur chemin partant de la ligne 0 et arrivant à la case $p_{i,j}$ en l'exprimant en fonction de $p_{i-1,j-1}, p_{i-1,j}, p_{i-1,j+1}$.

2.3 Des jeux

Exercice 10. [Fort Boyard] On se propose ici d'implémenter le jeu de Nîm suivant : on dispose $N(N+1)/2$ allumettes sur N lignes : la $i$ème ligne contient i allumettes. Il y a deux joueurs qui enlèvent chacun leur tour $1, 2, \dots$ ou k allumettes d'une des lignes. Le joueur qui retirent la dernière allumettes a perdu.

Écrire un programme qui prend en paramètre N er k et demande alternativement à chaque joueur de rentrer deux nombres i et l indiquant qu'il enlève l allumettes sur la $i$ème ligne, puis affiche l'état du jeu. Le programme doit s'arrêter quand un joueur a perdu et doit renvoyer une erreur si $l > k$, ou $i > N$, ou l est plus grand que le nombre d'allumettes restantes sur la $i$ème ligne.

Exercice 11. [Sudoku] On va se pencher sur le jeu du Sudoku. On représente un jeu de Sudoku partiellement rempli par un tableau d'`int` 9×9. La valeur de `t[i][j]` est la valeur de la case (i, j) si elle est remplie et 0 si elle est vide.

1. Écrire une fonction qui vérifie qu'une grille partiellement remplie n'est pas fausse, c'est-à-dire qu'il n'y a pas deux fois le même chiffre dans une même colonne/ligne ou dans un même bloc 3×3.

2. Écrire une fonction qui résoud une grille partiellement remplie si c'est possible et échoue sinon. **Idée :** on trouve la première case non-vide, on essaie une première valeur. On continue ainsi pour toute les cases jusqu'à trouver une incohérence. Quand on en trouve une, on revient à la dernière fois où on a essayé une valeur et on passe à la valeur suivante. Quand on a essayé toute les valeurs, on retourne à la case précédemment devinée etc. Utilisez le récursif.
3. Faites en sorte que votre programme puisse lire des fichiers texte représentant des Sudoku (9 lignes de 9 chiffres, 0 pour une case vide). Utilisez votre programme pour résoudre les Sudokus du journal !
4. (question très ouverte, à méditer) Sauriez-vous générer vous-même des grilles incomplètes ayant une unique solution ? Comment gérer la difficulté ?

Exercice 12. [Quartier libre] Avec ce que vous savez, vous êtes capables d'implémenter par exemple (avec une interface utilisateur certes rudimentaire) :

— un puissance 4
— un 2048
— un jeu du memory
— ...

Réfléchissez-y et lancez-vous.