TD XL

FICHIERS

Sujet dû à Nicolas Pécheux (Lycée Carnot). Ce sujet est long mais pas difficile : il faudra donc le finir chez vous.

Le programme encourage explicitement l'utilisation de fichiers pour stocker durablement des données sous différents formats. L'objectif de ce TP est de découvrir et d'expérimenter la gestion des fichiers en C et en OCaml. Dans ce TP, on s'intéresse à la lecture d'un fichier texte et à l'écriture de données textuelles, mais nous verrons plus tard que l'on peut aussi traiter de nombreux types de données : audio, vidéo, images, etc.

1 Compléments sur le langage C

1.1 Chaînes de caractères

En C, une chaîne de caractères est simplement un tableaux de `char` (octets, essentiellement) terminé par un zéro. Attention, il s'agit d'un « caractère nul » (octet égal à zéro) et pas du tout du caractère `0` (dont le code ASCII est 48).

Chaînes littérales Il est possible de définir une chaîne de caractères avec la syntaxe suivante :

```c
char *string = "toto";
```

Dans ce cas, `string` pointe vers un bloc de *cinq* octets : le caractère nul final a été ajouté automatiquement. Une telle chaîne ne peut pas être modifiée : elle est typiquement stockée dans une zone mémoire en « lecture seule ».

Longueur d'une chaîne Une chaîne connaît implicitement sa longueur : c'est le nombre de caractères du pointeur fourni jusqu'au premier caractère nul, exclu. La fonction `strlen` est prédéfinie :

```c
size_t strlen(char *string);
```

```c
char *string = "toto";
char arr[8] = {'t', 'o', 't', 'o', 't', 'a', 't', 'a'};
printf("%d\n", strlen(string)); // OK, affiche 4

// Un appel strlen(arr) provoquerait une erreur : on dépasserait
// du tableau en cherchant un caractère nul.

arr[3] = '\0';
printf("%s : longueur %d\n", arr, strlen(arr));
// OK : affiche "tot : longueur 3"

printf("%s : longeuer %d\n", &arr[3], strlen(&arr[3]));
// OK : affiche " : longueur 0"

arr[7] = '\0';
printf("%s : longueur %d\n", arr, strlen(arr));
// OK : affiche "tot : longueur 3" (on s'arrête au premier caractère nul)

printf("%s : longueur %d\n", &arr[4], strlen(&arr[4]));
// OK : affiche "tat : longueur 3"
```

Remarque

La fonction `strlen` parcourt la chaîne jusqu'à trouver un caractère nul : elle a donc une complexité linéaire en la taille de la chaîne. De plus, elle est extrêmement dangereuse puisqu'elle peut facilement provoquer un comportement non défini si la chaîne n'est pas correctement terminée. Une version plus récente existe :

```
size_t strlen_s(char *str, size_t strsz);
```

Cette version lit au maximum `strsz` octets, et renvoie `strsz` si elle ne trouve pas de caractère nul dans cette portion.

Autres fonctions sur les chaînes

La fonction `strcmp` permet de comparer deux chaînes de caractères :

```
int strcmp(char *lhs, char *rhs);
```

Elle renvoie :

- zéro si les deux chaînes sont égales ;
- une valeur strictement négative si `lhs` est avant `rhs` dans l'ordre lexicographique (par exemple `"abc"` par rapport à `"abca"`, ou `"azzz"` par rapport à `"ba"`) ;
- une valeur strictement négative si `rhs` est avant `lhs` dans l'ordre lexicographique.

La fonction `strcpy` permet de copier une chaîne de caractères dans un tableau de caractères :

```
char *strcpy(char *dest, char *src);
```

- La fonction copie les caractères (jusqu'au premier caractère nul inclus) de `src` dans le tableau pointé par `dest`. Si ce tableau n'est pas assez grand, le comportent est indéfini.
- La valeur de retour peut typiquement être ignorée : c'est un pointeur vers la copie, donc égal à `dest`.

1.2 Fichiers

La structure `FILE` permet de gérer un flux de données (un fichier, au sens large). Nous l'avons déjà utilisé implicitement, puisque trois objets de type `FILE*` sont prédéfinis :

- `stdin` qui correspond à l'entrée standard (le clavier, sauf si l'on a fait une redirection `./program < data_in.txt`), et qui est utilisé implicitement par `scanf` ;
- `stdout` qui correspond à la sortie standard (la console, sauf si l'on a fait une redirection `./program > data_out.txt`), et qui est utilisé implicitement par `printf` ;
- `stderrr` (sortie standard d'erreur, dirigé par défaut vers la console comme `stdout`).

On peut aussi obtenir un `FILE*` à partir d'un fichier présent sur l'ordinateur grâce à la fonction suivante :

```
FILE *fopen(char *filename, char *accessMode);
```

- `filename` est une chaîne de caractères indiquant le chemin d'accès du fichier : `"data.txt"`, ou `"data/file1.csv"` (chemin relatif dans les deux cas), ou `"/home/toto/image.ppm"` (chemin absolu).
- `accessMode` indique le mode d'ouverture du fichier :
  - `"r"` pour *read* (le fichier est ouvert en lecture, et doit exister préalablement) ;
  - `"w"` pour *write* (le fichier est ouvert en écriture : le fichier est écrasé s'il existe déjà, créé sinon) ;
  - `"a"` pour *append* (le fichier est ouvert en mode ajout : il est crée s'il n'existe pas, sinon la position courante est fixée à la fin du fichier et l'on peut ajouter des données).

  Plusieurs autres modes existent (en particulier pour ouvrir des fichiers en mode binaire, tous les modes présentés étant pour des fichiers en mode texte), mais nous n'en aurons pas besoin aujourd'hui.

Un fichier ouvert par `fopen` doit ensuite être fermé par `fclose` :

```c
FILE *f = fopen("data.txt", "r");
// on traite le fichier
fclose(f);
```

Remarque
Le terme de *fichier* doit ici être compris en un sens très large : il peut certes désigner un fichier au sens courant, mais aussi un flux d'entrée ou de sortie associé à un périphérique (clavier, console, imprimante), une zone mémoire permettant la communication avec d'autres processus...
Ces différents types de flux n'autorisent d'ailleurs pas les mêmes opérations : tous permettent une lecture linéaire (c'est le comportement par défaut), certains permettent de se déplacer à une position arbitraire alors que pour d'autres (entrée sur le clavier, par exemple) cela n'a aucun sens...

Fonctions `fscanf` et `fprintf` Les fonctions `fscanf` et `fprintf` fonctionnent exactement comme `scanf` et `printf`, sauf qu'elles prennent un argument supplémentaire indiquant sur quel flux doit se faire la lecture ou l'écriture :

```c
int fprintf(FILE *stream, char *format, ...);
int fscanf(FILE *stream, char *format, ...);
```

- Pour `fprintf`, le flux `stream` doit avoir été ouvert en écriture ; pour `fprintf`, en lecture.
- Un appel à `printf` équivaut à un appel à `fprintf` avec le paramètre `stream` fixé à `stdout`.
- Un appel à `scanf` équivaut à un appel à `fscanf` avec le paramètre `stream` fixé à `stdin`.

Nous avons surtout utilisé `scanf` pour lire des nombres jusqu'à présent, mais dans ce TP il faudra lire des caractères et des chaînes. Les formats utiles sont :

- `fscanf(f, "%c", char_pointer)` qui lit un caractère ;
- `fscanf(f, "%s", char_pointer)` qui lit une chaîne de caractères en s'arrêtant juste avant le premier caractère d'espacement rencontré (espace, tabulation ou retour à la ligne). Comportement non défini si le tableau `char_pointer` n'est pas assez grand pour contenir la chaîne lue (y compris le caractère nul rajouté automatiquement à la fin).

Autres fonctions Même si la bibliothèque standard du C est assez rudimentaire, elle fournit bien sûr un certain nombre d'autres fonctions pour interagir avec les flux. Ces fonctions ne sont pas au programme, ce qui ne nous empêchera pas nécessairement de nous en servir (`fseek` dans ce sujet, par exemple), mais signifie que vous n'avez pas à les maîtriser.
On peut cependant mentionner une fonction qui peut être utile pour certaines questions :

```c
char *fgets(char *str, int count, FILE *stream);
```

> Reads at most `count - 1` characters from the given file stream and stores them in the character array pointed to by `str`. Parsing stops if a newline character is found, in which case `str` will contain that newline character, or if end-of-file occurs. If bytes are read and no errors occur, writes a null character at the position immediately after the last character written to `str`.

2 Gestion des fichiers (en OCaml et en C)

Cette partie est à faire tout d'abord en OCaml puis *à reprendre entièrement en C.*

► **Question 1** En ouvrant en écriture puis fermant immédiatement, créer un fichier vide. Vérifier avec les commandes Unix que cela a bien fonctionné.

▶ **Question 2** Écrire les lignes suivantes dans le fichier que vous venez de créér précédemment, en écrasant son ancien contenu (qui était vide, ce n'est donc pas bien grave)

```
Première ligne
Deuxième ligne
```

▶ **Question 3** Afficher à l'écran le contenu du fichier précédent.

▶ **Question 4** (En C uniquement) Ajouter une troisième ligne de votre choix dans ce fichier, en utilisant le mode `"a"`. Vérifier.

▶ **Question 5** Créer un fichier de nom `plot.txt` qui contient le tableau de valeurs de la fonction $x \mapsto x^2$ avec 10 000 lignes de la forme ci-dessous. Vérifier que le fichier est bien présent et contient bien ce qu'il faut.

```
0 0
1 1
2 4
3 9
...
```

▶ **Question 6** Écrire un programme `copy` qui prend deux arguments sur la ligne de commande `nom1` et `nom2` et qui copie le contenu du fichier `nom1` dans un fichier de nom `nom2`. Si le deuxième fichier existait déjà, son contenu est écrasé. Faire quelques tests.

Remarque

En OCaml, on pourra procéder ligne par ligne. En C, sans connaître la longueur maximale des lignes, c'est nettement plus problématique : le plus simple est de procéder caractère par caractère.

▶ **Question 7** Modifier le programme précédent pour que les lettres « a » et « e » soient échangées lors de cette copie corrompue.

3 Prénoms français (en C)

Le fichier `prenoms.txt` dans le répertoire `data/prenoms` contient les prénoms donnés aux petites françaises et aux petits français, à raison d'un prénom par ligne. Dans toute cette partie on demande d'écrire en C une fonction ou un programme permettant de répondre aux questions posées.

▶ **Question 8** Déterminer le nombre de prénoms.

▶ **Question 9** Combien il y a-t-il de prénoms qui commencent par une lettre donnée, par exemple par « e » ?

▶ **Question 10** Quels sont les prénoms les plus longs ? Quels sont les prénoms les plus courts ?

▶ **Question 11** Est-ce qu'un prénom a été donné, par exemple le vôtre ?

▶ **Question 12** Quelle proportion de prénoms comportent une lettre donnée, par exemple la lettre « e » ?
Remarque : cette proportion doit être inférieure à 100% *!*

▶ **Question 13** Quels sont les prénoms palindromiques ?

▶ **Question 14** Inventer une question intéressante et y répondre.

4 Cent mille milliards de poèmes (en OCaml)

En 1961, l'écrivain et poète Raymond Queneau proposa un recueil de sonnets potentiels à composer par le lecteur. Un sonnet est composé de 14 vers — ici des alexandrins — séparés en deux groupes de 4 vers (les quatrains avec une succession de rimes ABAB) et deux groupes de 3 vers (les tercets de rime AAB et CCB). Pour chaque vers Queneau propose 10 possibilités. Ces possibilités sont données dans les fichiers `q1v1.txt` (premier quatrain, premier vers), `q1v2.txt` (premier quatrain, deuxième vers), . . ., `t2v3` (deuxième tercet, troisième vers), qui se trouvent dans le répertoire `data/queneau`.

Par exemple, le premier poème est :

```
Le roi de la pampa retourne sa chemise
pour la mettre à sécher aux cornes des taureaux
le cornédbîf en boîte empeste la remise
et fermentent de même et les cuirs et les peaux

Je me souviens encor de cette heure exeuquise
les gauchos dans la plaine agitaient leurs drapeaux
nous avions aussi froid que nus sur la banquise
lorsque pour nous distraire y plantions nos tréteaux

Du pôle à Rosario fait une belle trotte
aventures on eut qui s'y pique s'y frotte
lorsqu'on boit du maté l'on devient argentin

L'Amérique du Sud séduit les équivoques
exaltent l'espagnol les oreilles baroques
si la cloche se tait et son terlintintin
```

▶ **Question 15** Écrire une fonction `premier_poeme : unit -> unit` qui affiche ce premier poème à partir des fichiers sources des vers.

Pour la question suivante, on utilisera la fonction `Random.int : int -> int` du module `Random` que l'on initialisera avec `Random.self_init : unit -> unit`.

▶ **Question 16** Écrire une fonction `compose_poeme : unit -> unit` qui affiche une possibilité de poème choisie aléatoirement parmi les 10^{14} poèmes possibles. On pourra commencer par écrire le premier quatrain. Tester plusieurs fois et admirer.

▶ **Question 17** Écrire une fonction `sauvegarde_poemes : unit -> unit` qui écrit dans le répertoire courant cent poèmes choisis aléatoirement de noms `poeme00.txt`, `poeme01.txt`, . . ., `poeme99.txt`.

Queneau ajoute : « En comptant 45 s pour lire un sonnet et 15 s pour changer les volets à 8 heures par jour, 200 jours par an, on a pour plus d'un million de siècles de lecture, et en lisant toute la journée 365 jours par an, pour 190 258 751 années plus quelques plombes et broquilles (sans tenir compte des années bissextiles et autres détails) ».

5 Décimales de π (en C)

Le fichier `pi.txt`, qui se trouve dans le répertoire `data/pi/` contient les 10 premiers millions de décimales de π. Par exemple, les 50 premières décimales de π sont :

```
> head -c50 data/pi/pi.txt
14159265358979323846264338327950288419716939937510
```

▶ **Question 18** Écrire en C un programme `decimale_de_pi.c` qui, une fois compilé, prend en argument un entier $n \in \mathbb{N}^*$ sur la ligne de commande et qui affiche la n-ème décimale de pi. Si le fichier n'est pas lisible, si l'argument en ligne de commande n'est pas donné ou si celui-ci ne correspond pas à une décimale que l'on peut trouver dans le fichier, on affichera sur la sortie d'erreur un message et on arrêtera le programme.

Remarques

- On se déplacera séquentiellement dans le fichier jusqu'à la décimale voulue.
- On rappelle que `atoi` permet de convertir une chaîne de caractères en entier.

  Par exemple :

```
> ./decimale_de_pi 939
7
```

▶ **Question 19** Écrire en C un programme `histogramme_decimales_de_pi.c` qui, une fois compilé, affiche l'histogramme des fréquences d'apparition des décimales de π, toujours en gérant correctement les éventuelles erreurs.

On doit obtenir :

```
> ./histogramme_decimales_de_pi
0 : 0.099944
1 : 0.099933
2 : 0.100031
3 : 0.099996
4 : 0.100109
5 : 0.100047
6 : 0.099934
7 : 0.100021
8 : 0.099981
9 : 0.100004
```

Remarque

Il semblerait que π soit un nombre *équiréparti*. On conjecture que π est un nombre *univers* c'est-à-dire que l'on peut trouver dans son développement décimal toute suite finie de chiffres. On pense même que π est un nombre *normal* c'est-à-dire que toute suite finie de décimales consécutives de longueur k apparaît avec une fréquence limite de $\frac{1}{10^k}$.

Par exemple, on peut trouver le nombre 9265 dans les décimales de π puisque $\pi = 1415\mathbf{9265}35\ldots$.

Dans la question suivante, on pourra utiliser les fonctions suivantes :

```
long ftell(FILE *stream);
```

> Returns the file position indicator for the file stream `stream`.
>
> If the stream is open in binary mode, the value obtained by this function is the number of bytes from the beginning of the file.
>
> If the stream is open in text mode, the value returned by this function is unspecified and is only meaningful as the input to `fseek()`.

```
int fseek(FILE *stream, long offset, int whence);
```

> Sets the file position indicator for the file stream `stream` to the value pointed to by `offset`.
>
> If the stream is open in binary mode, the new position is exactly `offset` bytes measured from the beginning of the file if `origin` is `SEEK_SET`, from the current file position if `origin` is `SEEK_CUR`, and from the end of the file if `origin` is `SEEK_END`. Binary streams are not required to support `SEEK_END`, in particular if additional null bytes are output.
>
> If the stream is open in text mode, the only supported values for offset are zero (which works with any origin) and a value returned by an earlier call to `ftell` on a stream associated with the same file (which only works with origin of `SEEK_SET`).
>
> Returns `0` upon success, nonzero value otherwise.

▶ **Question 20** Écrire en C un programme `indice_dans_decimales_de_pi` qui, une fois compilé, prend en argument un entier $n \in \mathbb{N}$ sur la ligne de commande et qui affiche l'indice de la position de la première occurrence de cet entier n dans les décimales de π, en commençant à l'indice 1 pour la première décimale. Si aucune occurrence n'est trouvée (vraisemblablement parce que 10 millions de décimales ce n'est pas assez), on affichera -1. Comme ci-dessus, on gérera les erreurs éventuelles.

Par exemple :

```
> ./indice_dans_decimales_de_pi 9265
5
```

6 Nombre de langages informatiques (langage libre)

Le fichier `liste_langages.txt` dans le répertoire `donnees/langages` est tiré de la page correspondante de Wikipedia.

▶ **Question 21** Donner le nombre de langages de programmation répertoriés dans ce fichier.

7 Admissibles aux mines (langage libre)

Le fichier `mines.tsv` qui se trouve dans le répertoire `donnees/mines` contient les résultats des admissibilités des candidats au concours Mines-Ponts 2015. Il est issu d'un `pdf` disponible sur le site du concours [1].

Dans ce fichier, sur chaque ligne figurent les champs : numéro de candidat, nom et prénom, résultat, série d'oral si admissible. Chaque champ est séparé par une tabulation `'\t'`. On appelle ce format *Tab-separated values* d'où l'extension.

Le fichier étant issu d'un fichier `pdf`, il a été nettoyé, mais il reste les numéros des pages initiales sur certaines lignes.

▶ **Question 22** Afficher avec un programme C ou OCaml le contenu de ce fichier.

▶ **Question 23** Écrire une fonction `nettoyage` créant un fichier dans le même répertoire dont le nom est `mines_propre.tsv` dans lequel ne figurent plus ces numéros de ligne.

▶ **Question 24** Écrire une fonction `nb_admissibles` renvoyant le nombre d'admissibles.

▶ **Question 25** Écrire une fonction `separe_series` créant, toujours dans le bon répertoire, un fichier pour chaque série d'oral : `serie1.tsv`, `serie2.tsv`, `serie3.tsv` et `serie4.tsv`. Chaque fichier contenant le numéro et l'identité des candidats de la série correspondante, au format *Tab-separated values*.

1. https://mines-ponts.fr/