IN 101 - TD 03

Énoncé

Attention : désormais l'implémentation des exercices sera à faire en fin de TD, après que nous ayons examiné toutes les questions de réflexion et de structuration des algorithmes. Laissez le clavier...

1 Point fixe d'un tableau

≪Écrivez une fonction qui vérifie s'il existe un point fixe dans un tableau ≫

On appelle ≪point fixe d'un tableau ≫ un indice `i` tel que `t[i] == i`.

Soit un tableau `t` contenant, **par hypothèse**, uniquement des entiers naturels tous différents et triés en ordre croissant strict.

Q 1.1. Proposez un algorithme pour une fonction `fixpt` qui prend en argument un tel tableau et détermine s'il existe un point fixe dedans.

2 Fou aux échecs

≪Écrivez une fonction qui vérifie si le déplacement d'un fou d'une position initiale à une position finale en **un coup** est conforme aux règles du jeu d'échecs. ≫

On rappelle que les échecs se jouent sur un damier de taille 8×8 et qu'un fou ne peut se déplacer qu'en **diagonale**.

On modélise le damier par une matrice de coordonnées comprises entre 0 et 7 inclus.

On **ne cherche pas** à vérifier si la position initiale est atteignable par les règles du jeu d'échec (par exemple, de par la disposition initiale des pièces, il est impossible de trouver un des deux fous noirs sur une case blanche au cours d'une partie respectant les règles).

L'échiquier ne contient que le fou (pas d'autres pièces qui pourraient le bloquer).

Q 2.1. Quel sont les domaines des entrées et des sorties ?

Q 2.2. Quelle sont les relations entre les positions initiales et finales d'un déplacement légal ?

3 État des finances

≪On souhaite écrire un programme permettant de connaître le solde de chacune des personnes d'un groupe où les membres se doivent de l'argent. ≫

Exemple pratique

- Charlie doit à Alice 10.
- Alice doit à Bob 15.
- Charlie doit à Bob 7.
- Bob doit à Charlie 5.

La question à laquelle répondre est « quel est le solde de chacun après règlement de ces dettes ? » La réponse (affichage souhaité) :

```
$ ./money.x bill2.txt
Alice : -5
Bob : 17
Charlie : -12
```

Q 3.1. Proposez une esquisse d'algorithme pour résoudre le problème de calcul des soldes (on s'occupera de l'acquisition des données plus loin).

Nous n'avons pas envie de saisir au terminal toutes les informations, nous allons utiliser un fichier **texte** comme entrée. Pour manipuler un fichier **texte** il faut :

1. L'ouvrir : fonction `fopen`.
2. Y lire (ou y écrire) ce que l'on veut : fonction `fscanf` (ou `fprintf`).
3. Le fermer une fois que l'on n'en a plus besoin : fonction `fclose`.
4. Pour tester si l'on est arrivé à la **fin du fichier**, il faut avoir fait **au moins une lecture** avant d'utiliser la fonction dédiée : `feof`. C'est important car ça joue sur la forme de l'algorithme.

On ne s'intéresse pas à toutes les fonctionnalités des fichiers, on ne regarde que ce dont nous avons besoin. Nous détaillerons ces fonctions dans la partie implémentation. Il faut actuellement juste savoir que `fscanf` fonctionne comme `scanf` (que vous connaissez déjà), sauf qu'au lieu de récupérer les données au clavier, elle les récupère dans un fichier.

La structure d'un fichier de données est fixe. La **première** ligne contient, le nombre n de personnes impliquées dans le problème. Ensuite, séparés par des espaces ou des retours à la ligne (ce qui ne change rien pour `scanf`/`fscanf`), les n noms des personnes.

Suivent un nombre **quelconque** de lignes comportant à chaque fois : le nom de la personne **devant** de l'argent suivi du nom de celle **recevant** l'argent suivi du **montant** (un entier), le tout séparé par des **espaces**.

```
$ more bill2.txt
3
Alice Bob Charlie
Charlie Alice 10
Alice Bob 15
Charlie Bob 7
Bob Charlie 5
MyMachine:~
```

Q 3.2. Complétez votre esquisse d'algorithme de la question Q3.1 pour acquérir les données d'entrée.

Q 3.3. Comment allez-vous retrouver l'indice du solde d'une personne dans le tableau de soldes à partir de son nom ? Quels sont les domaines d'entrée(s) et de sortie(s) de la fonction de reherche d'indice ?

Q 3.4. À quoi devrait-on faire attention en lisant une « ligne de transaction » afin que le programme soit robuste ?

4 Implémentation

Q 4.1. Écrivez en `C` l'algorithme esquissé dans l'exercice 2 pour vérifier la légalité d'un déplacement. Votre programme prendra ses arguments sur la ligne de commande. Vous pourrez tester votre programme comparant ses résultats avec ceux qui suivent :

```
move_bishop (1, 1, 1, 1)    → false
move_bishop (1, 5, 4, 5)    → false
move_bishop (1, 5, 2, 4)    → true
move_bishop (2, 4, 1, 5)    → true
move_bishop (7, 6, 1, 0)    → true
move_bishop (8, 7, 1, 0)    → false
move_bishop (2, 2, -1, -1)  → false
```

Manipulation de fichiers en `C`

ATTENTION : Savoir lire dans un fichier est à maîtriser **absolument** car cela resservira dans d'autres TDs et possiblement en **examen**.

— L'ouverture d'un fichier se fait grâce à la fonction

FILE ∗fopen (**char** ∗filename, **char** ∗mode)

qui prend en argument le nom du fichier et une chaîne de caractères représentant le mode d'accès (`"rb"` pour lecture, `"wb"` pour écriture) et renvoie un pointeur sur un ≪ *descripteur* ≫ du fichier (de type `FILE`). Si l'ouverture échoue, `fopen` retourne le pointeur `NULL`.

— La lecture dans un fichier se fait par la fonction

int fscanf (FILE ∗stream, **char** ∗format, ...)

qui opère comme `scanf`, mais en lisant dans le fichier désigné par `stream`. Par rapport à `scanf`, il faut juste passer en plus en premier argument le descripteur du fichier dans lequel lire. Chaque lecture consécutive ≪avance≫ automatiquement dans le fichier. Cette fonction retourne le nombre d'éléments effectivement lus (nombre de % réussis dans le format). Autrement dit, une lecture partiellement réussie retournera un entier strictement inférieur au nombre de % présents dans le format. Par contre, en cas d'échec total dû à l'atteinte de fin de fichier, elle retournera la valeur `EOF` (égale à -1).

— Pour savoir si l'on a atteint la fin du fichier, il faut interroger la fonction

int feof (FILE ∗stream)

après lecture. Elle renvoie ≪vrai≫ si la fin du fichier a été atteinte suite à cette tentative de lecture, ≪faux≫ sinon. Autrement dit, une lecture qui lit le dernier ≪ mot ≫ d'un fichier **ne** provoquera **pas** le signalement de fin de fichier. Ce sera la **prochaine** lecture qui, en échouant, permettra à `feof` de répondre ≪vrai≫.

— Finalement, lorsque l'on n'a plus besoin de travailler sur un fichier ouvert, il faut le fermer en utilisant la fonction

int fclose (FILE ∗stream)

Q 4.2. Écrivez la fonction `find_index_by_name` qui implémente l'algorithme de la question Q3.3.

Q 4.3. Écrivez la fonction `compute_amount` qui implémente le reste de l'algorithme de l'exercice 3. Vous rajouterez un `main` prenant le nom du fichier de transactions en argument via la ligne

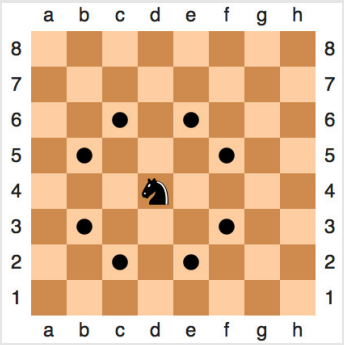
de commande.

S'il vous reste du temps ou pour continuer après la séance.

5 Cavalier aux échecs

≪Écrivez une fonction qui vérifie si le déplacement d'un cavalier d'une position initiale à une position finale en un coup est conforme aux règles du jeu d'échecs. ≫

Les déplacements possibles d'un cavalier sur un damier d'échec sont rappelés sur l'image ci-dessous (provenance Wikipédia) :



Q 5.1. Quelle sont les relations entre les positions initiales et finales d'un déplacement légal ?

Q 5.2. Écrivez en `C` la fonction `move_knight` qui vérifie la légalité d'un déplacement. Vous pourrez tester votre programme comparant ses résultats avec ceux qui suivent :

```
move_knight (1, 1, 1, 1)   → false
move_knight (4, 5, 3, 7)   → true
move_knight (3, 5, 2, 7)   → true
move_knight (0, 3, -1, 2)  → false
move_knight (4, 6, 5, 6)   → false
move_knight (5, 5, 6, 7)   → true
```

6 Textruction

Vous avez peut-être déjà lu ce résultat d'études qui mnotre que l'rdore des letrtes dnas un mot n'a pas d'icnopmrtae, la sulee cohse iornapttme est que la prmeière et la drenière sinoet à la bnnoe palce.

On se propose d'écrire un programme permettant d'afficher le texte contenu dans un fichier en mélangeant les lettres des mots comme dans la phrase ci-dessus. Les permutations de lettres doivent ne pas changer la première ni la dernière lettre d'un mot. Bien évidemment, la ponctuation du texte ainsi que d'éventuels chiffres ne sont pas affectés par les permutations.

Q 6.1. Décrivez les grandes étapes de ce programme.

Q 6.2. Partant du principe qu'un mot est une chaîne de caractères, décrivez le processus de permutation des lettres.

Puisque nous souhaitons permuter au hasard, il nous faut une fonction générant des nombres (pseudo-)aléatoires. La bibliothèque standard de C fournit la fonction `rand` à cet effet. Consultez sa documentation en utilisant la commande Unix `man` :

```
man rand
```

Q 6.3. Que fait cette fonction ? Quel fichier d'en-tête doit être utilisé pour y accéder ? Comment obtenir un nombre entre 0 et autre chose que le fameux `RAND_MAX` ?

Q 6.4. Donnez l'expression qui permet de calculer un indice de caractère tiré au hasard parmi les lettres d'un mot dont on connaît la longueur (par exemple stockée dans une variable `len`).

Q 6.5. Décrivez l'algorithme de la fonction `shuffle` qui prend en argument une chaîne de caractères et effectue les permutations directement dedans. On sait qu'en C, une chaîne passée en argument peut être modifiée en place (implicitement un passage d'argument par adresse, comme pour tout tableau).

Précédemment dans ce TD, les fonctions de manipulation de fichiers vous ont été présentées. Pour la suite il va falloir être capable de lire le contenu d'un fichier caractère par caractère. Il est bien entendu possible d'utiliser `fscanf` avec le format `%c`. Mais vous pouvez également utiliser la fonction

int getc (FILE ∗stream)

qui lit un seul caractère dans le fichier et le retourne en résultat (sous forme d'un entier entre 0 et 255). Si la lecture échoue car la fin du fichier a été atteinte, la fonction renverra la valeur `EOF` (autre façon dans ce type de lecture de tester la fin de fichier).

On souhaite maintenant écrire une fonction `next_word` qui devra lire le prochain mot dans un fichier dont le descripteur est passé en argument, transmettre ce mot à la fonction appelante et retourner « vrai » s'il reste des choses à lire dans le fichier, et « faux » sinon.

Q 6.6. Quels sont les domaines d'entrée et de sortie de cette fonction ?

Un mot est une suite contiguë de caractères **alphabétiques** (nommés ici « lettres »). Autrement dit, lecture et construction d'un mot s'arrêtent si le caractère lu est autre chose qu'une **lettre**.

Q 6.7. Décrivez le processus le lecture d'un mot.

Pour savoir si un caractère est une « lettre », vous pouvez utiliser la fonction

int isalpha (**int** c)

disponible via un `#include <ctype.h>` et dont vous obtiendrez la documentation avec la commande `man` d'Unix.

Q 6.8. Comment allez-vous savoir si un mot qui vient d'être lu doit voir ses lettres permutées ou bien si l'on ne doit pas y toucher car c'est de la ponctuation, des chiffres ou des blancs ?

Q 6.9. Imnleptaz en C les fnocitons précédnmeemt conçues et rtujaeor un `main` qui premet de pndrere en aurmengt de ligne de comnmdae le nom du feihcir à taetirr, pius ahiffce son ctneonu en aiunqapplt les pmatitournes.
Ouch, le dernier mot n'est quand même pas facile à retrouver !

Q 6.10. Lancez votre programme 2 fois de suite sur un même texte. Que remarquez-vous ?