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 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).

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`.

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 le solde d'une personne dans le tableau de soldes à partir de son nom ? Quels sont les domaines de ses entrée(s) et sortie(s) ?

Q 3.4. À quoi devriez-vous faire attention en lisant une ≪ligne de transaction ≫ ?

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. Si la lecture échoue (au moins pour un des % du format) `fscanf` retourne la valeur `EOF` (égale à -1). Sinon, elle retourne le nombre d'éléments lus (nombre de % dans le format).

— 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, ≪faux≫ sinon.

— 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.

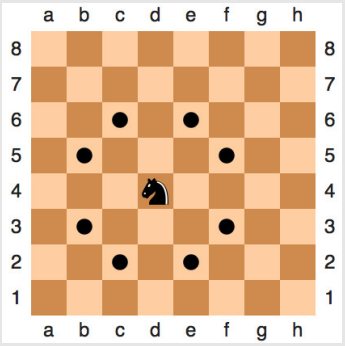
Q 4.3. Écrivez la fonction `compute_amount` qui implémente le reste de l'algorithme de l'exercice 3. Vous rajouterez un `main` prenant le nom du fichier de transactions en argument via la ligne de commande.

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
```