TP Noté

Algorithmes et programmation S1 FISA

2019-2020

Vous devez coder tous les exercices de la première section, puis ceux de la section suivante. Aucun point ne sera attribué si vous n'avez pas essayé de faire tous les exercices de base avant de passer à la seconde partie. Pour la seconde partie un fichier annexe `littlehorses.c` vous est fourni.

Rendre votre travail

Vous devez rendre une archive `nom_prenom.tar.gz` contenant tous vos fichiers sur `exam.ensiie.fr` dans le dépôt nommé `fisa-iap-tp-note-2019`.

Tous vos codes doivent compiler avec `gcc -Wall -Wextra -std=c99` sans erreur ni warning, sauf éventuellement des warnings de variables non utilisées.

1 Exercices de base (environ 20-30 minutes)

Exercice 1 — *Comparaison*

Créer un programme `01_comparaison.c` contenant uniquement la fonction `main` qui prend un entier en entrée (avec `argc` et `argv`) ; puis qui demande un entier avec `scanf`, et affiche avec `prinf` 1 si les deux entiers sont égaux et 0 sinon. Vous pouvez utiliser la fonction `atoi` pour transformer un `char*` en `int`.

Exercice 2 — *Appartenance*

Créer un programme `02_tableau.c` contenant une fonction `appart` qui, connaissant un entier `size`, un tableau `tab` contenant `size char` et un `char m`, renvoie 1 si `m` appartient au tableau `tab` et 0 sinon.

Exercice 3 — *Echange*

Créer un programme `03_echange.c` contenant une fonction `swap` qui échange les valeurs de deux variables entières ; comme dans l'exemple suivant :

```
1       int x = 3;
2       int y = 5;
3       // Appel de votre fonction avec x et y
4       printf("\d \d", x, y); // Affiche "5 3".
5
```

Exercice 4 — *Création*

Créer un programme `04_pointeur.c` contenant une fonction `point` qui prend en entrée un `double d` et renvoie un pointeur sur `d`.

Votre fonction `main` devra appeler votre fonction `point` pour créer ce pointeur puis le libérer.

Exercice 5 — *Fibonacci*

Créer un programme `05_fibo.c` contenant une fonction `fibo` qui prend en entrée un `int n` et renvoie le $\mathtt{n}^{\mathrm{e}}$ entier $\varphi_{\mathtt{n}}$ de la suite de fibonacci : si `n` vaut 0 ou 1, $\varphi_{\mathtt{n}} = \mathtt{n}$ et sinon $\varphi_{\mathtt{n}} = \varphi_{\mathtt{n}-1} + \varphi_{\mathtt{n}-2}$.

2 Petits chevaux (environ 2h30 - 3h)

Effectuez chacun de ces exercices en complétant le fichier `littlehorses.c` disponible en annexe du sujet, vous pouvez utiliser la fonction `main` pour tester. Tous ces exercices sont indépendants sauf le premier qui définit les structures et le dernier qui met tout en place.

Le jeu des petits chevaux consiste en les règles suivantes. Le plateau de jeu est comme suit, de taille `15 x 15`. On a nommé chaque colonne et chaque ligne avec un numéro ou une lettre pour simplifier.

```
          0123456789ABCDE
          ---------------

0|        RR     OOO     BB
1|        RR     O1O     BB
2|              O2O
3|              O3O
4|              O4O
5|              O5O
6|        OOOOOOO6OOOOOOO
7|        0123456X6543210
8|        OOOOOOO6OOOOOOO
9|              O5O
A|              O4O
B|              O3O
C|              O2O
D|        VV     O1O     JJ
E|        VV     OOO     JJ
```

Au début, chaque joueur dispose de 4 chevaux situés dans un des 4 coins du plateau (les écuries, une par joueur), représentés par les 4 symboles R (pour rouge), V (pour vert), B (pour bleu) et J (pour jaune).

L'objectif d'un joueur est d'amener un cheval (un seul) au centre du plateau, sur la case `X`, ligne et colonne `7`.

À chaque tour de jeu, un joueur lance un dé.

S'il fait un 6, il peut, s'il lui en reste, prendre un de ses chevaux en écurie et le placer sur le plateau. Le joueur rouge le place sur la ligne 6, colonne 0 ; le joueur bleu sur la ligne 0, colonne 8 ; le joueur jaune sur la ligne 8, colonne E ; et le joueur vert sur la ligne E, colonne 6. Si un autre cheval se trouve déjà sur cette case, ce dernier est renvoyé à son écurie, même s'il s'agit d'un cheval du joueur dont c'est le tour.

Quelque soit la valeur de son dé, il peut, à la place, déplacer un cheval sorti de son écurie placé sur une case `O` d'autant de cases `O` que la valeur de son dé, dans le sens des aiguilles d'une montre. Si le cheval rattrape ou dépasse un autre cheval (adverse ou non), ce dernier retourne à son écurie.

Une exception au déplacement est le cas où le cheval a fait un tour complet (donc, par exemple, si un cheval rouge atteint la case `O` de la ligne 7, colonne 0). Dans ce cas, le cheval s'arrête sur cette case, même si son jet de dé lui permet d'aller plus loin.

Une fois cette case atteinte, le cheval peut essayer d'atteindre la case `X` au centre, en passant successivement par les cases `1, 2, 3, 4, 5, 6` qui le séparent du centre. Pour se déplacer sur une case chiffrée, le joueur doit faire, avec un lancé de dé, exactement la valeur indiquée sur la case. Ainsi, par exemple, pour aller sur la case 1, il doit faire 1. Pour aller sur la case 3, il doit être sur la case 2 et faire 3. Enfin, pour atteindre la case `X`, le joueur doit être sur la case 6 et faire un 6.

Exercice 6 — *Structures*

1. Déclarer un type `cheval` équivalent à `struct s_cheval`, sans définir cette structure.
2. Déclarer un type `tcase` équivalent `struct s_case`, sans définir cette structure.
3. Déclarer un type `plateau` équivalent à un pointeur sur pointeur sur `tcase`, correspondant à un plateau de jeu.

4. Définir le type `struct s_cheval` comme étant une structure contenant un `char j` correspondant à la couleur de son joueur et un pointeur `tcase* c` correspondant à la case où se trouve ce cheval.
5. Définir le type `struct s_case` comme étant une structure contenant un `int l` (la ligne de la case sur le plateau), un `int c` (la colonne de la case sur le plateau), un pointeur `cheval*` correspondant au cheval placé sur cette case, si celui-ci existe (le cheval est `NULL` sinon), un `char d` correspondant au caractère permettant d'afficher cette case en console, et, enfin, un pointeur `tcase* next` correspondant à la case qui suit cette case dans l'ordre de déplacement des chevaux. Détail important, aucune case ne pointe sur les cases chiffrées 1 car la case qui le précède possède 2 successeurs. La case à la ligne 0 et colonne 7 pointe sur la case à sa droite. La case à la ligne 7, colonne E pointe sur la case en dessous. La case à la ligne E, colonne 7 pointe sur la case à sa gauche. Et la case à la ligne 7, colonne 0, pointe sur la case au dessus.

REMARQUE IMPORTANTE : une fonction d'initialisation est fournie dans le fichier, elle est utilisée pour générer un plateau identique à celui décrit au début des règles, avec toutes les valeurs nécessaires à tous les structs, y compris les valeurs du champs `next` pour les cases et le placement des chevaux dans les coins du plateau.

Exercice 7 — *Affichage*

Écrire une fonction `display` qui prend en entrée `plateau p` de taille `15 x 15`. La fonction affiche le plateau en console de la même manière que présentée dans les règles, avec le numéro de ligne à gauche et de colonne en haut et les tirets pour séparer ces symboles du plateau. Une case sans cheval est affichée normalement, une case contenant un cheval est affichée avec la couleur du cheval à la place.

Exercice 8 — *Lancer de dé*

Écrire une fonction `die` qui ne prend aucun argument en entrée et renvoie un entier aléatoire entre 1 et 6 (inclus).

Exercice 9 — *Trouver un cheval*

Pour les deux fonctions suivantes, il vous est conseillé de compléter la fonction `trouve_bis` du fichier. Mais ce n'est pas une obligation.

1. Écrire une fonction `trouve` qui prend en entrée un `plateau p` et demande à l'utilisateur deux `char l` et `char c` (avec 2 scanf) et renvoie, s'il existe un pointeur sur le cheval situé sur la case dont la ligne est `l` et la colonne est `c`. Sinon elle renvoie NULL.

   `l` et `c` sont des noms de ligne ou de colonne, donc des chiffres ou les lettres `A, B, C, D, E`. Il faudra donc les transformer en entiers.
2. Écrire une fonction `gagne` qui prend en entrée un `plateau p` et qui renvoie un `char` correspondant au joueur qui a gagné si un de ses chevaux est au centre du plateau et le caractère `'-'` sinon.

Exercice 10 — *Déplacement*

1. Écrire une fonction `teleport` qui prend en entrée un `plateau p`, un `cheval* pch`, un `int l` et un `int c` et qui déplace le cheval pointé par `pch` sur la case du plateau située à la `l`^{e} ligne et la `c`^{e} colonne.
2. Écrire une fonction `ecurie` qui prend en entrée un `cheval ch` et renvoie un entier égal à 1 si le cheval est dans son écurie et 0 sinon.
3. Écrire une fonction `caseDeSortie` qui prend en entrée un `plateau p`, un `char j` et qui renvoie un pointeur sur `tcase` correspondant à la case où vont les chevaux du joueur `j` quand ils sortent de l'écuries.
4. Écrire une fonction `rentre` qui prend en entrée un `plateau p`, un `cheval* pch` dont le cheval pointé est supposé hors de son écurie et qui place ce dernier dans son écurie.

5. Écrire une fonction `avance` qui prend en entrée un `plateau p`, un `cheval ch` hors ou dans son écurie et un entier `int m` et qui renvoie un pointeur `tcase** p` sur au plus `m` `tcase*`. Ces `tcase*` pointent sur les cases que le cheval `ch` va rencontrer dans l'ordre de son déplacement s'il se déplace suite à un lancé de dé de valeur `m`. La case sur laquelle se trouve le cheval avant le déplacement n'est pas renvoyée dans `p`.

   Par exemple, si le cheval est sur la case ligne `A`, colonne `6` et que `m` vaut 3 alors le pointeur `p` doit contenir 3 pointeurs sur `tcase` qui pointeront respectivement sur les cases (ligne `9`, colonne `6`), (ligne `8`, colonne `6`) et (ligne `8`, colonne `5`).

   On rappelle qu'une `tcase` contient le champs `next`, utile à cette fonction.

   Il se peut que le cheval rencontre moins de `m` cases : s'il sort de l'écurie, s'il a fait un tour complet ou s'il est sur une case chiffrée, dans ce cas, les cases en trop sont remplacées par `NULL` dans `p`. Il se peut aussi que le cheval ne bouge pas, par exemple s'il est dans l'écurie et que $\texttt{m} \neq 6$. Dans ce cas `p` ne contient que des `NULL`.

6. Écrire une fonction `deplace` qui prend en entrée un `plateau p`, un `cheval* pch` et un entier `int m` et qui déplace le cheval pointé par `pch` de `m` cases. Si le cheval ne se déplace pas (car il est sur une case chiffrée différente de `m - 1` ou car il est dans l'écurie et $\texttt{m} \neq 6$), il ne se passe rien. Le cheval renvoie dans l'écurie tout cheval qu'il rattrape ou dépasse. Enfin cette fonction libère le ou les pointeurs qu'il faut libérer.

Exercice 11 — *Boucle principale*

1. Ecrire une fonction `tourdejeu` qui prend en entrée un `plateau p` et un `char j` correspondant au prochain joueur à lancer le dé. Ce joueur lance le dé avec la fonction `die` ; le résultat est affiché, puis la fonction `tourdejeu` appelle la fonction `trouve` qui doit renvoyer un cheval du joueur. Si ce cheval n'appartient pas au joueur ou si aucun cheval n'a été renvoyé, la fonction `trouve` est rappelée. Ce cheval est ensuite déplacé avec la fonction `deplace`. Si le joueur a gagné suite au déplacement, la fonction `tourdejeu` renvoie 1, sinon 0.
2. Ecrire, dans la fonction `main`, un programme qui permet de jouer au jeu des petits chevaux.