UNIVERSITÉ PARIS 7 DENIS DIDEROT - IF1 - EXAMEN - 20 DÉCEMBRE 2012 - 08^{15}-11^{15}

Aucun document. Aucune machine. Le barème est indicatif. Les six exercices sont indépendants. Une question peut toujours être traitée en utilisant les précédentes (traitées ou non). Les morceaux de code Java devront être clairement présentés, indentés et commentés.

Exercice 1. (3 points) Le but est de construire des entiers miroirs d'eux-mêmes via un procédé récursif.

1. On définit la fonction récursive `speg` comme suit :

```
  static int speg(int n, int m){
2       if (n==0) return m;
        return speg(n/10,10*m+n%10);
4       }
```

   Expliquer ce que produit l'évaluation de `speg(123,456)`.
2. Préciser quel appel de la fonction `speg` permet de définir une fonction `miroir` qui prend en argument en entier `n` et renvoie l'entier miroir obtenu en renversant l'ordre des chiffres de `n`.
3. Écrire une fonction `estMiroir` qui teste si l'entier en argument est miroir de lui-même.
4. Écrire une fonction **récursive** `persistance` qui prend en argument un entier `n` et qui renvoie l'entier miroir de lui-même calculé de la façon suivante : si l'entier `n` n'est pas miroir de lui-même, on calcule la somme de `n` et de son miroir, puis on recommence si nécessaire avec l'entier obtenu.
   Expliquer ce que produit l'évaluation de `persistance(78)`.

Exercice 2. (4 points) On considère un jeu où deux adversaires (ici la machine et l'utilisateur) choisissent puis présentent simultanément une figure parmi cinq possibles (codées `'C'`, `'F'`, `'L'`, `'P'`, `'S'`), les règles de supériorité entre ces figures étant : Ciseau coupe Feuille, qui recouvre Pierre, qui casse Ciseau, qui décapite Lézard, qui empoisonne Spock, qui vaporise Pierre, qui écrase Lézard, qui mange Feuille, qui réfute Spock, qui tord Ciseau.

1. Écrire une fonction `alea` qui renvoie le caractère codant une des cinq figures choisie aléatoirement.
2. Écrire une fonction `zuper` qui prend en arguments deux caractères codant des figures et renvoie `+1` si la première figure est supérieure à la seconde, `-1` si elle lui est inférieure et `0` en cas d'égalité.
3. Écrire une fonction `tour` qui choisit une figure `c1` aléatoirement, demande à l'utilisateur de choisir une figure `c2`, affiche la figure `c1` et renvoie `+1` si `c1` est supérieure à `c2`, `-1` si elle lui est inférieure et `0` sinon.
4. Le vainqueur d'un match est le premier qui gagne cinq tours. Écrire une fonction `match`.
   En donner un exemple d'exécution.
5. Proposer une manière de truquer discrètement ce jeu.

Exercice 3. (2,5 points) L'unité de base du calendrier maya est le *kin*, soit un jour. Les unités supérieures sont le *winal* (soit 20 kins), le *tun* (soit 18 winals), le *katun* (soit 20 tuns) et le *baktun* (soit 20 katuns).

1. Écrire une fonction `estDureeMaya` qui prend en argument un tableau d'entier et teste s'il correspond à une durée dans le calendrier maya valide : il doit porter les quantités entières (positives) de chacune des cinq unités, les quantités de kins, de tuns et de katuns doivent être inférieures à 19 et celle de winals inférieure à 17.
2. Écrire les fonctions `maya2Jours` et `jours2Maya` qui convertissent une durée dans le calendrier maya supposée valide (sous forme de tableau d'entiers) en nombre de jours et réciproquement.
3. Cette fin d'année correspond à la fin d'un cycle de 13 baktuns. Donner une expression qui utilise une des deux fonctions précédentes et dont la valeur serait une estimation de l'année du début de ce cycle.

Exercice 4. (2,5 points) Fractran est un langage de programmation imaginé par John Conway, dans lequel un programme `p` est une liste de fractions positives et s'exécute en mettant à jour un argument entier `n` comme suit :
– pour la fraction `f` la plus à gauche dans `p` pour laquelle le produit $n \times f$ est un entier, `n` est remplacé par $n \times f$,
– cette règle est répétée jusqu'à ce qu'aucune fraction de `p` ne produise un entier par multiplication par `n`.
Écrire une fonction `fractran` qui prend en arguments un entier `n` et un tableau `t` de longueur paire d'entiers strictement positifs (un numérateur, un dénominateur, un numérateur, un dénominateur, etc) et renvoie le résultat de l'exécution du programme en Fractran codé par `t` sur l'argument `n`.

Exercice 5. (5,5 points) Un tableau de n entiers est une *permutation* s'il contient les entiers $0, 1, \ldots, n-2, n-1$, dans un ordre quelconque.

Par exemple, | 2 | 0 | 3 | 1 | est une permutation de longueur 4, mais | 2 | 4 | 3 | 1 | ou | 2 | 1 | 0 | 1 | n'en sont pas.

Dans cet exercice, on suppose disposer, sans avoir à la définir, d'une fonction `estPerm` qui prend en argument un tableau d'entiers et teste s'il s'agit d'une permutation. Si `tab` est une permutation et si `i, j` sont deux entiers avec $0 \leqslant$ `i` $<$ `j` $<$ `tab.length`, on dit que `tab` admet une *inversion en* `(i, j)` si on a `tab[i]` $>$ `tab[j]`.

Par exemple, | 2 | 0 | 3 | 1 | a une inversion en $(0,1)$, une autre en $(0,3)$ et une dernière en $(2,3)$, soit 3 inversions en tout.

Si `tab` est une permutation de longueur `n`, son *tableau d'inversions* est le tableau d'entiers de longueur `n` dont l'élément d'indice `i` contient le nombre d'inversions en `(i, j)` de `tab` pour `i` $+ 1 \leqslant$ `j` $\leqslant$ `n` $- 1$.

Par exemple, le tableau d'inversions de | 2 | 0 | 3 | 1 | est | 2 | 0 | 1 | 0 |.

1. Donner le nombre total d'inversions de la permutation | 3 | 1 | 2 | 0 | et son tableau d'inversions.
2. Écrire une fonction `maxInversions` qui prend en argument un entier `n` et renvoie la permutation de longueur `n` qui a le plus grand nombre d'inversions, parmi toutes les permutations de longueur `n`.
3. Écrire une fonction `nbInversions` qui prend en argument un tableau `tab` d'entiers et renvoie le nombre d'inversions de `tab` si `tab` est une permutation et `-1` sinon.
4. Écrire une fonction `tableauInversions` qui prend en argument un tableau `tab` d'entiers et renvoie le tableau d'inversions de `tab` si `tab` est une permutation et `null` sinon.
5. Écrire une fonction `estInv` qui prend en argument un tableau d'entiers et teste si c'est le tableau d'inversions d'une permutation.
6. Écrire une fonction `inv2Perm` qui prend en argument un tableau `tab` d'entiers et renvoie la permutation dont `tab` est le tableau d'inversions si `tab` est le tableau d'inversions d'une permutation et `null` sinon.

Exercice 6. (5,5 points) Dans un entrepôt (divisé en cases carrées et codé par un tableau `grille` bidimensionnel d'entiers), un magasinier (dont la position `pos` sur la grille est codée par un tableau de deux coordonnées entières) doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions (chacune codée par un tableau de deux coordonnées dont l'une est nulle et l'autre est ± 1) et pousser (mais pas tirer) une seule caisse à la fois. Dans le tableau `grille`, une case cible est indiquée par `-1`, une case libre par `0`, une case cible contenant une caisse par `2`, une case contenant une caisse par `3` et un mur (case infranchissable) par `4`.

1. Écrire une fonction `sontCorrectes` qui prend en arguments une grille et une position et teste si l'entrepôt codé est rectangulaire, contient uniquement des cases libres, des caisses, des cases cibles et des cases infranchissables, contient autant de caisses (sur des cases non-cibles) et que de cases cibles (sans caisse) et si le magasinier se trouve à l'intérieur de l'entrepôt ailleurs que sur une caisse ou un mur.
2. Écrire une fonction `string` qui prend en arguments une grille et une position (supposées correctes) et renvoie la chaîne de caractères représentant l'entrepôt avec `'u'` pour une case cible, `'.'` pour une case libre, `'o'` pour une case cible contenant une caisse, `'n'` pour une case contenant une caisse, `'x'` pour un mur, `'A'` pour le magasinier sur une case libre et `'B'` pour le magasinier sur une case cible.
3. Écrire une fonction `estComplete` qui prend en argument une grille (supposée correcte) et teste si les caisses sont toutes rangées dans les cases cibles.
4. Écrire une fonction `direction` qui prend en argument un caractère et, s'il est `'e'`, `'z'`, `'a'` ou `'s'`, renvoie la direction associée (est, nord, ouest, sud) sous forme d'un tableau de deux coordonnées dont l'une est nulle et l'autre est ± 1 et renvoie `null` sinon.
5. Écrire une fonction `coupPossible` qui prend en arguments une grille, une position (supposées correctes) et un caractère (parmi `'e'`, `'z'`, `'a'`, `'s'`) et teste si le magasinier peut prendre la direction donnée ; auquel cas, la grille et la position seront mises à jour.
6. En déduire une fonction `jouer` qui met en œuvre ce jeu.
7. Proposer un (ou plusieurs) moyen(s) de prévenir certaines situations de blocage.

```
$ java Sokoban
...xxx
ux.n.x
..nx..
.ou...
x.A.xx  dir? a

...xxx
ux.n.x
..nx..
.ou...
xA..xx  dir? z

...xxx
ux.n.x
.nnx..
.Bu...
x...xx  dir? a

...xxx
ux.n.x
.nnx..
Auu...
x...xx  dir? z

...xxx
ux.n.x
Annx..
.uu...
x...xx  dir? z

...xxx
Bx.n.x
.nnx..
.uu...
x...xx  dir? z

A..xxx
ux.n.x
.nnx..
.uu...
x...xx  dir? e

.A.xxx
ux.n.x
.nnx..
.uu...
x...xx  dir? w
Interruption!
$
```