Introduction à la Programmation (IP1)
Examen – **Durée : 3 heures**

Université Paris-Diderot – Jeudi 17 décembre 2015

- Aucun document ni aucune machine ne sont autorisés. Les téléphones doivent être éteints et rangés.
- Les exercices sont tous indépendants.
- Une réponse peut utiliser les fonctions ou les procédures attendues à une question précédente (même si elle est non traitée).
- Les fragments de code Java doivent être correctement indentés.
- Les questions annotées par le mot **Amphi** portent sur des sujets vus en cours d'amphi.

Exercice 1 (Lapins et renards)

On souhaite modéliser un écosystème où cohabitent des lapins et des renards. On note R_n le nombre de lapins pour l'année n et F_n le nombre de renards pour l'année n.

Le nombre de lapins pour l'année $n+1$ est proportionnel à ce nombre pour l'année n mais aussi au nombre de renards pouvant manger ces lapins lors de l'année n :

$$R_{n+1} = max(0, 2 * R_n - F_n)$$

Quant aux renards, leur nombre pour l'année $n+1$ dépend du nombre de lapins à manger et de leur nombre pour l'année n :

$$F_{n+1} = (F_n * F_n + F_n/2) * R_n/(max(1, F_n))$$

Dans les formules précédentes, l'opérateur / correspond à la division entière. Dans les questions suivantes, on pourra utiliser la fonction « `public static int max (int x, int y)` » qui attend deux entiers `x` et `y` et retourne le maximum de ces deux entiers.

Q1: *Écrire une fonction « `public static int rabbits (int pastRabbits, int pastFoxes)` » qui attend un entier `pastRabbits` représentant R_n, un entier `pastFoxes` représentant F_n et qui retourne R_{n+1}. Par exemple, « `rabbits (2, 1)` » retourne 3 tandis que « `rabbits (1, 3)` » retourne 0.*

Q2: *Écrire une fonction « `public static int foxes (int pastRabbits, int pastFoxes)` » qui attend un entier `pastRabbits` représentant R_n, un entier `pastFoxes` représentant F_n et qui retourne F_{n+1}. Par exemple, « `foxes (2, 1)` » retourne 2 tandis que « `foxes (1, 3)` » retourne 3.*

Q3: *En utilisant les fonctions issues des deux questions précédentes, écrire une fonction « `public static int simulateRabbits (int initialRabbits, int initialFoxes, int n)` » qui attend un entier `initialRabbits` représentant R_0, un entier `initialFoxes` représentant F_0, un entier positif `n` et qui retourne la taille de la population de lapins de l'écosystème après `n` années, c'est-à-dire la valeur R_n. On introduira quatre variables entières pour représenter R_n, R_{n+1}, F_n et F_{n+1} à chaque itération du calcul. Par exemple, « `simulateRabbits(2, 1, 2)` » retourne 4 tandis que « `simulateRabbits(1, 3, 2)` » retourne 0.*

□

Exercice 2 (Fractions)

Une fraction est un couple de nombres entiers positifs (p, q) tels que $q \neq 0$. Dans la suite, on utilisera la notation $\frac{p}{q}$ pour parler du couple (p, q). On représentera un tel couple par un tableau d'entiers de taille 2. On rappelle que n est divisible par k si le reste de la division entière de n par k vaut 0.

Q1: *Le résultat de la multiplication de deux fractions $\frac{p}{q}$ et $\frac{p'}{q'}$ est la fraction $\frac{p*p'}{q*q'}$. Écrire une fonction « `public static int[] multiply (int[] f1, int[] f2)` » qui attend deux tableaux d'entiers représentant les fractions `f1` et `f2` et qui retourne un nouveau tableau représentant la fraction résultat de la multiplication de `f1` et `f2`. Si `f1 = { 2, 3 }` et `f2 = { 3, 5 }` alors `multiply (f1, f2)` retourne un tableau qui vaut `{ 6, 15 }`.*

Q2: *La simplification d'une fraction $\frac{p}{q}$ par un nombre entier naturel strictement positif k est obtenue en divisant tant que c'est possible p et q par k, c'est-à-dire tant que p et q sont tous les deux divisibles par k. Écrire une procédure « `public static void simplify (int[] f, int k)` » qui attend un tableau d'entiers représentant une fraction `f` ainsi qu'un entier naturel strictement positif `k` et qui modifie la fraction `f` en la simplifiant par `k`. Si `f = { 24, 20 }` alors après l'exécution de `simplify (f, 2)`, `f` vaut `{ 6, 5 }`.*

Q3: ***(Amphi)*** *Le résultat de l'addition de deux fractions $\frac{p}{q}$ et $\frac{p'}{q'}$ est la fraction $\frac{p*q'+p'*q}{q*q'}$. Lorsque $q = q'$, on peut simplifier directement la fraction en calculant $\frac{p+p'}{q}$. Le programme suivant est une tentative d'implémentation de l'addition de deux fractions et son utilisation sur les fractions $\frac{1}{2}$ et $\frac{1}{3}$ et sur les fractions $\frac{1}{2}$ et $\frac{3}{2}$.*

```
1  public class Fraction {
2    public static int[] addition (int[] f1, int[] f2) {
3      int[] r = { 0, 0 };
4      r[1] = f1[1] * f2[1];
5      if (f1[1] == f2[1]) {
6          r[0] = f1[0] + f2[0];
7      } else {
8          r[0] = f1[0] * f2[1] + f2[0] * f1[1];
9      }
10     return r;
11   }
12   public static void main (String[] args) {
13     int[] a = { 1, 2 };
14     int[] b = { 1, 3 };
15     int[] c = { 3, 2 };
16     int[] ab = { 0, 0 };
17     int[] ac = { 0, 0 };
18     ab = addition (a, b);
19     ac = addition (a, c);
20   }
21 }
```

À l'aide du modèle d'exécution vu en cours d'amphi, donnez la trace d'exécution de ce programme.

Q4: ***(Amphi)*** *Pourquoi la fonction `addition` précédente est-elle incorrecte ? Justifiez votre réponse en donnant un contre-exemple et proposez une correction.*

□

Exercice 3 (Images en noir et blanc)

Une image faite de m lignes de n colonnes contenant des points noirs ou blancs est représentée à l'aide d'un tableau `i` de m tableaux de n booléens tel que `i[r][c] = true` si le point à la ligne r et à la colonne c est noir et `i[r][c] = false` si le point à la ligne r et à la colonne c est blanc.

Q1: *Écrire une fonction « `public static boolean[][] inverse (int m, int n, boolean[][] i)` » qui attend deux entiers strictement positifs `m` et `n`, une image `i` de `m` lignes de `n` colonnes, et qui produit une image `j`*

telle que « `j[r][c] == ! i[r][c]` » pour tout `r` compris entre `0` et `m - 1` et pour tout `c` compris entre `0` et `n - 1`.

Q2: *Écrire une procédure « `public static void rect (int m, int n, boolean[][] i, int r0, int c0, int h, int w)` » qui attend deux entiers strictement positifs `m` et `n`, une image `i` de `m` lignes de `n` colonnes, un entier `r0` tel que `0 <= r0 < m`, un entier `c0` tel que `0 <= c0 < n`, un entier `h` tel que `r0 + h <= m`, un entier `w` tel que `c0 + w <= n`. Cette procédure dessine dans `i` un rectangle à la position `(r, c)` de hauteur `h` et de largeur `w` : elle modifie `i` en mettant en noir tous les points de coordonnées `(r, c)` telles que `r0 <= r < r0 + h` et `c0 <= c < c0 + w`.*

Q3: *Écrire une procédure « `public static void flip (int m, int n, boolean[][] i)` » qui attend deux entiers strictement positifs `m` et `n`, une image `i` de `m` lignes de `n` colonnes, et qui modifie `i` en inversant l'ordre de ses lignes. Ainsi, la première ligne devient la dernière ligne, la seconde devient l'avant-dernière ligne, etc...*

□

Exercice 4 (Scrabble)

Dans cet exercice, nous supposerons que les mots sont des chaînes de caractères non vides formées uniquement des lettres contenues dans le tableau de chaîne de caractères `letters` suivant :

```
String[] letters = { "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L",
    "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z" };
```

Dans les questions suivantes, vous pourrez utiliser la fonction « `public static String charAt(String a, int n)` » qui renvoie la lettre d'indice `n` de `a` sous la forme d'une chaîne de caractères. (Les indices débutent à 0.) Vous pourrez aussi utiliser la fonction « `public static boolean stringEqual(String a, String b)` » qui retourne `true` si et seulement si `a` est la même chaîne que `b`.

Q1: *Écrire une fonction « `public static int code (String[] letters, String c)` » qui attend le tableau des lettres défini plus haut ainsi qu'une chaîne `c` et qui retourne la position de `c` dans le tableau `letters` si `c` apparaît dans `letters` et qui retourne `-1` sinon. Par exemple, « `code (letters, "E")` » retourne `4` tandis que « `code (letters, "YODA")` » retourne `-1`.*

Q2: *Écrire une fonction « `public static int points (String[] letters, int[] lp, String w)` » qui attend le tableau `letters` des lettres défini plus haut, un tableau d'entiers `lp` tel que `lp[i]` est le nombre de points rapportés par la `i`-ème lettre de `letters` et une chaîne `w` dont toutes les lettres sont dans `letters`. Cette fonction retourne la somme des points obtenus grâce aux lettres du mot `w`. Par exemple, si `lp[4] = 1` et `lp[19] = 3`, alors « `points (letters, lp, "ETE")` » retourne `1 + 3 + 1 = 5`.*

Q3: *Écrire une fonction « `public static int pointsWithBonus (String[] letters, int[] lp, int[] bonus, String w)` » qui attend les mêmes arguments que la fonction précédente ainsi qu'un tableau d'entiers représentant un bonus associé à chaque lettre du mot `w`. (`bonus[i]` est le bonus de la lettre `i` de `w`.) Cette fonction retourne la somme des points obtenus grâce aux lettres du mot `w` en prenant en compte les bonus codés comme suit :*

- *Si `bonus[i] = 1`, il n'y a pas de bonus.*
- *Si `bonus[i] = 2`, la lettre numéro `i` de `w` compte double.*
- *Si `bonus[i] = 3`, la lettre numéro `i` de `w` compte triple.*
- *Si `bonus[i] = 4`, le mot compte double.*
- *Si `bonus[i] = 5`, le mot compte triple.*

Enfin, si le mot `w` est de longueur 7 alors il s'agit d'un scrabble : on rajoute 50 points au score final. Par exemple, si `lp[4] = 1`, `lp[19] = 3`, `bonus[0] = 3`, `bonus[1] = 5`, `bonus[2] = 4`, alors « `pointsWithBonus (letters, lp, bonus, "ETE")` » retourne `(1 * 3 + 3 + 1) * 3 * 2 + 0 = 42`.

Ainsi, pour calculer les points, on commence par calculer la somme des points des lettres en prenant en compte les bonus liés à la lettre (si la lettre compte double ou triple). Le score obtenu est alors doublé et/ou triplé en fonction des bonus de la forme "le mot compte double" et "le mot compte triple". (On peut cumuler plusieurs doublement et triplement.) Enfin, on rajoute 50 points si le mot est de longueur 7.

FIGURE 1 – Plateau de Scrabble

Le Scrabble est un jeu de lettres qui se déroule sur un plateau comme décrit par la figure 1. Nous utilisons un tableau de tableaux de chaînes de caractères pour représenter ce plateau. Pour dire qu'une case ne contient pas de lettre, on lui affecte la chaîne `"."`*. Par exemple, si* `board` *représente le plateau de la figure 1, alors on a* `board[0][14] = "E"` *car on trouve la lettre* `E` *à la position* `A15`*. On a aussi* `board[2][1] = "."` *car il n'y a pas de lettre à la position* `C2`*.*

Les cases grisées de la figure 1 représentent des bonus. Ces bonus sont décrits par un tableau de tableaux d'entiers `boardbonus` *en suivant le même codage que celui de la question 3. Par exemple, si la case* `H4` *contient le bonus « lettre compte triple » alors* `boardbonus[7][3] = 3`*.*

Chaque joueur possède 7 lettres et doit essayer de les placer sur le plateau pour former des mots. Comme pour les mots-croisés, les mots peuvent partager des lettres. Ainsi, quand un joueur forme un mot, il peut utiliser ses propres lettres mais aussi celles déjà posées sur le plateau. Par exemple, si le joueur possède les lettres `{ C, D , Y, J, A, V, Z }`*, il peut poser les lettres* `J` *en* `E1`*,* `V` *en* `E3` *et* `A` *en* `E4` *pour former le mot* `JAVA`*. On utilisera un tableau de chaînes de caractères de longueur 7 pour représenter l'ensemble des lettres du joueur. Pour retirer une lettre de cet ensemble, on remplacera cette lettre par* `"."` *dans le tableau.*

Q4: *Écrire une fonction « `public static boolean makeHorizontalWord (int m, int n, String[][] board, String w, int r, int c, String[] p)` » qui attend deux entiers positifs* `m` *et* `n`*, un plateau* `board` *de* `m` *lignes de* `n` *colonnes, le mot* `w` *que le joueur souhaite placer horizontalement sur le plateau, le numéro* `r` *de la ligne et le numéro* `c` *de la colonne où il souhaite placer la première lettre, ainsi qu'un tableau de chaînes de caractères* `p` *représentant ses lettres. Cette fonction modifie le plateau en y plaçant les lettres du joueur pour former le mot* `w` *si c'est possible. Cette fonction modifie aussi* `p` *en y retirant les lettres placées sur le plateau. Enfin, si on a pu placer les lettres sur le plateau, cette fonction retourne* `true`*, sinon elle retourne* `false`*. Il pourra être utile d'introduire des fonctions et procédures auxiliaires.*

Q5: *Écrire une fonction « `public static boolean makeVerticalWord (int m, int n, String[][] board, String w, int r, int c, String[] p)` » dont les paramètres et le comportement sont similaires à ceux de la fonction de la question précédente mais qui place le mot verticalement plutôt qu'horizontalement.*

Q6: *Le rôle de l'arbitre est de vérifier qu'une proposition faite par un joueur est valide, ce qui signifie :*

(a) que cette proposition est constituée d'un mot, d'une position et d'une direction ;

(b) que l'on peut placer ce mot sur le plateau ;

(c) que le mot est un mot du dictionnaire.

Si ces trois critères sont vérifiés alors les lettres du joueur sont placées sur le plateau et l'arbitre calcule les points du joueur à l'aide de la fonction `pointsWithBonus` *de la question précédente. Après avoir déterminé ses arguments, proposez une fonction qui implémente un arbitre de Scrabble. Il pourra être utile d'introduire des fonctions et procédures auxiliaires.*

□