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

Université Paris-Diderot – Jeudi 18 décembre 2014

- 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). Par exemple, dans l'exercice 1, si vous ne réussissez pas à programmer la fonction `achille`, vous pouvez tout de même l'utiliser pour répondre à la question 3 de ce même exercice.
- Les fragments de code Java doivent être correctement indentés.
- Le barème est donné à titre indicatif. Il est donc sujet à de légères modifications.

Exercice 1 (Achille victorieux – 4 points)

Nézon d'Allée[1] *raconte cette histoire : « Un jour, Achille a disputé une course avec une tortue. Achille était le coureur le plus rapide de Grèce et la tortue était plutôt connue pour sa lenteur. En effet, on sait maintenant qu'en* 1 *seconde, Achille parcourait* 1000 *centimètres tandis que la tortue n'en parcourait que* 3*. On décida donc de donner une avance de* 10000 *centimètres à la tortue. Mais, ce ne fut pas suffisant... »*

Q1: *Écrire une fonction «* `int achille (int s)` *» qui attend un entier positif ou nul* `s` *et qui renvoie la distance (en centimètres) entre Achille et son point de départ après* `s` *seconde(s) de course.*

Q2: *Écrire une fonction «* `int turtle (int s)` *» qui attend un entier positif ou nul* `s` *et qui renvoie la distance (en centimètres) entre la tortue et le point de départ d'Achille après* `s` *seconde(s) de course.*

Q3: *En utilisant une boucle* `while` *et les deux fonctions* `achille` *et* `turtle` *des questions précédentes, écrire une fonction «* `int achilleScore ()` *» qui renvoie le nombre de secondes nécessaires à Achille pour dépasser (strictement) la tortue.*

□

Exercice 2 (Histogramme – 4 points)

Q1: *Écrire une fonction «* `int count (int n, int[] t)` *» qui attend un entier* `n`*, un tableau d'entiers* `t` *et qui renvoie le nombre de fois où* `n` *apparaît dans* `t`*. Par exemple, si* `t` *vaut* `{ 1, 0, 3, 1, 5 }`*,* `count (1, t)` *doit renvoyer* `2`*.*

Q2: *Écrire une fonction «* `int[] histogram (int n, int[] t)` *» qui attend un entier positif* `n`*, un tableau* `t` *d'entiers compris entre* `0` *et* `n − 1` *et qui renvoie un tableau* `h` *où* `h[i]` *contient le nombre de fois où* `i` *apparaît dans* `t`*. Comme les valeurs sont entre* `0` *et* `n − 1`*, la taille de* `h` *est* `n`*.*
Par exemple, `histogram (4, {1, 0, 1, 2, 1, 2})` *renvoie* `{1, 3, 2, 0}` *car dans le tableau en entrée,* `0` *apparaît* `1` *fois,* `1` *apparaît* `3` *fois,* `2` *apparaît* `2` *fois et* `3` *n'apparaît pas.*
(Remarque : Vous n'avez pas à vérifier que les entiers contenus dans `t` *sont compris entre* `0` *et* `n − 1`*.)*

Q3: *Écrire une procédure «* `void printHistogram (int[] h)` *qui affiche l'histogramme représenté par* `h` *à l'aide d'étoiles. Pour chaque indice* `i` *du tableau, on affichera le nombre entier* `i` *suivi d'un espace puis d'un nombre de caractères* `*` *égal à* `h[i]`*. Par exemple, pour* `h` *valant* `{ 1, 3, 2, 0 }`*, l'appel de procédure* `printHistogram (h)` *donne :*

```
0 *
1 ***
2 **
3
```

(On pourra utiliser la procédure « `void printString (String s)` *» qui affiche la chaîne de caractères* `s`*, la fonction «* `string intToString (int n)` *» qui convertit un entier en une chaîne de caractères ainsi que la fonction «* `int intArrayLength (int[] a)` *» qui renvoie la longueur du tableau* `a`*.)*

1. Son cousin Zénon d'Élée, qui a lui réellement existé, raconte une histoire où la tortue gagne la course contre toute attente.

Q4: *Écrire un programme* Histogram *qui attend en argument une séquence d'entiers compris entre* 0 *(inclus) et* 3 *(inclus) et qui affiche l'histogramme de ces valeurs. Si les entiers passés en argument ne sont pas compris entre* 0 *et* 3*, le programme affichera* Erreur*. Par exemple, si on exécute le programme ainsi*

```
java Histogram 1 0 1 2 1 2
```

alors on obtient l'affichage décrit dans la question précédente. Tandis que :

```
java Histogram 1 9 3 0 1
```

produit l'affichage :

```
Erreur
```

(On pourra utiliser la procédure « void printString (String s) » qui affiche la chaîne de caractères s*, la fonction « int stringToInt (String s) » qui convertit une chaîne de caractères en un entier, ainsi que la fonction « int intArrayLength (int [] t) qui renvoie la longueur du tableau d'entiers* t*.)*

□

Exercice 3 (Carré très magique – 4 points)

On rappelle que si s *est une valeur de type* int [][] *alors* s *peut être vue comme un tableau bidimensionnel où* s[i][0]*,* s[i][1]*, ...,* s[i][n − 1] *sont les éléments de la ligne* i *(en supposant qu'elle est de longueur* n*) et* s[0][i]*,* s[1][i]*, ...,* s[n − 1][i] *sont les éléments de la colonne* i *(en supposant qu'il y a* n *lignes de longueur au moins* i *dans le tableau* s*).*

Un ***carré très magique*** *d'ordre* n *est un tableau d'entiers à* n *lignes et* n *colonnes tel que la somme des entiers de chaque ligne et de chaque colonne vaut* n^2*.*

Le carré de gauche suivant est un carré très magique d'ordre 3 tandis que celui de droite n'en est pas un.

3	3	3
1	5	3
5	1	3

3	3	3
1	1	3
1	1	3

Q1: *Pour chaque tableau bidimensionnel (a), (b) et (c) ci-dessous, indiquez s'il s'agit d'un carré très magique. Ne pas justifier.*

3	3	3	1	
1	5	3	1	(a)
5	1	3	0	

1	2	6	
2	6	1	(b)
6	0	2	

1	1	7	
3	6	0	(c)
5	2	2	

Q2: *Écrire une fonction « boolean isSquare (int [][] s) » qui attend un tableau de tableaux d'entiers* s *et qui renvoie* true *si* s *est un carré – c'est-à-dire un tableau de* n *tableaux de longueur* n *– et* false *sinon.*

(On pourra utiliser « int intArrayArrayLength (int [][] s) » qui renvoie la longueur du tableau de tableaux d'entiers s *et « int intArrayLength (int [] a) » qui renvoie la longueur du tableau* a*.)*

Q3: *Écrire une fonction « boolean checkRow (int i, int [][] s, int x) » qui attend un entier* i *qui est un numéro de ligne de* s*, un tableau de tableaux d'entiers* s *qui est un carré, et un entier* x*. Cette fonction renvoie* true *si la somme des éléments de la ligne* i *de* s *vaut* x *et* false *sinon.*

Q4: *Écrire une fonction « boolean checkColumn (int i, int [][] s, int x) » qui attend un entier* i *qui est un numéro de colonne de* s*, un tableau de tableaux d'entiers* s *qui est un carré, et un entier* x*. Cette fonction renvoie* true *si la somme des éléments de la colonne* i *de* s *vaut* x *et* false *sinon.*

Q5: *Écrire une fonction « boolean isVeryMagicSquare (int [][] s) » qui attend un tableau de tableaux d'entiers* s*, et qui renvoie* true *si* s *est un carré* ***et*** *qu'il est très magique d'ordre* n*, où* n *est le côté du carré, et* false *sinon.*

□

Exercice 4 (Un échange incorrect – 4 points)

```java
public class IntSwap {
    public static void swap (int[] t, int i, int j) {
        t[i] = t[i] + t[j];
        t[j] = t[i] - t[j];
        t[i] = t[i] - t[j];
    }
    public static void main (String[] args) {
        int[] t = new int[2];
        t[0] = 1;
        t[1] = 2;
        swap (t, 0, 1);
        swap (t, 1, 1);
    }
}
```

Q1: *Donnez la séquence d'exécution du programme* IntSwap *en utilisant le modèle d'exécution vu en cours.*

Q2: *Pourquoi la spécification suivante n'est-elle pas réalisée par l'implémentation de la procédure* swap *?*

Entrées : i *et* j *sont dans les bornes du tableau* t.
Sortie : *Les contenus des cases* i *et* j *du tableau* t *sont échangés.*
Si i = j, *le contenu de la case* i *(qui est aussi la case* j*) ne change pas.*

Q3: *Donnez une version modifiée de la procédure* swap *qui respecte la spécification de la question précédente.*

□

Exercice 5 (La hotte du père Noël – 4 points)

Contrairement aux idées reçues, même si le père Noël se déplace plus vite que la lumière, il ne peut pas mettre une quantité infinie de cadeaux dans sa hotte. En effet, celle-ci ne supporte que 42*kg de cadeaux. Il doit donc faire des allers-retours entre son usine de jouets et les différentes cheminées pour régulièrement remplir (le plus possible) sa hotte.*

Fort heureusement, l'usine du père Noël est très bien organisée à l'aide d'immenses bacs à jouets numérotés. Les jouets pesant 1*kg se trouvent dans le bac numéro* 1*, ceux pesant* 2*kg se trouvent dans le bac numéro* 2*, . . . et ceux pesant* 9*kg se trouvent dans le bac numéro* 9*. On n'a jamais vu un jouet pesant plus de* 9*kg et le poids minimal d'un jouet est* 1*kg.*

Le père Noël souhaite programmer un remplissage automatique de sa hotte. On suppose donc que les bacs à jouets de son usine sont représentés par un tableau d'entiers de taille 10 *dans lequel la case numéro* i *contient le nombre de jouets restant dans le bac numéro* i*. La hotte du père Noël est représentée par un tableau d'entiers de taille* 10 *suivant le même modèle : dans la case numéro* i*, on trouve le nombre de jouets de poids* i *kg contenus dans la hotte. Dans les deux cas, la case numéro* 0 *n'est pas utilisée.*

La méthode ancestrale de remplissage de hotte est la suivante : tant que c'est possible, on rajoute dans la hotte un jouet le plus lourd possible. Nous allons essayer de traduire cette méthode ancestrale en un programme.

Q1: *Écrire une procédure « void* pickToy (int [] factory, int p) *» qui attend un tableau* factory *représentant les bacs à jouets de l'usine du père Noël et un entier* p *compris entre* 1 *et* 9*. Cette fonction met à jour le contenu des cases de* factory *pour prendre en compte le fait que l'on a retiré un jouet de poids* p *kg du bac numéro* p *de l'usine. (On suppose qu'il existe un tel jouet.)*

Q2: *Écrire une procédure « void* putToy (int [] bag, int p) *» qui attend un tableau* bag *représentant la hotte du père Noël et un entier* p *compris entre* 1 *et* 9*. Cette fonction met à jour le contenu des cases de* bag *pour prendre en compte le fait que l'on a ajouté un jouet de poids* p *kg dans la hotte.*

Q3: *Écrire une fonction « int* whichToy (int [] factory, int c) *» qui attend un tableau* factory *représentant les bacs à jouets de l'usine du père Noël et un entier* c *positif ou nul représentant le poids actuel de la hotte du père Noël. Cette fonction renvoie le poids* p *du jouet le plus lourd qui est dans un bac de l'usine et qui est tel que* p + c <= 42*. Si aucun jouet ne remplit ces critères alors la fonction renvoie* 0*.*

Q4: *Écrire une procédure « void* fill (int [] factory, int [] bag) *» qui attend un tableau* factory *représentant les bacs à jouets de l'usine du père Noël et un tableau* bag *représentant la hotte vide du père Noël. Cette fonction met à jour les tableaux* factory *et* bag *en suivant la méthode ancestrale du père Noël.*

Q5: *La méthode du père Noël n'est pas optimale ! Trouvez un tableau* factory *représentant une configuration des bacs à jouets de l'usine du père Noël pour lequel la méthode ancestrale échoue à mettre* 42*kg de jouets à partir de la hotte vide alors que c'est possible.*

□