# Introduction à la Programmation (IP1)
# Examen de seconde session – **Durée : 3 heures**

Université Paris-Diderot – mercredi 24 juin 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 demandées à une question précédente (même si elle est non traitée). Par exemple, la question 4 de l'exercice 1 peut être traitée même si vous n'avez pas réussi à traiter la question 2 et 3.
- Les fragments de code Java doivent être correctement indentés.
- Il est recommandé de commenter les fragments de code Java.
- Il n'est pas nécessaire de vérifier que les paramètres d'une fonction valident les hypothèses énoncées par sa spécification.
- Le barème est donné à titre indicatif. Il est donc sujet à de légères modifications.

Dans la suite, pour obtenir la taille d'un tableau `a`, on pourra écrire `a.length` ou supposer donnée une fonction `arrayLength` telle que `arrayLength(a)` est la longueur de `a`. De même, on pourra supposer l'existence d'une fonction `stringLength` telle que `stringLength(s)` retourne la longueur de la chaîne `s`.

**Exercice 1 (Énigme familiale – 4 points)**

*On souhaite utiliser un programme pour résoudre l'énigme suivante :*

> *J'ai quatre fois l'âge de mon petit-fils. Si on inverse les deux chiffres de chaque âge, le nouvel âge de mon petit-fils est égal à trois fois le mien.*

***Q1:*** *Écrire une fonction « `int inverse(int age)` » qui prend en paramètre un entier `age` compris entre* $10$ *et* $99$ *et qui retourne le nombre que l'on obtient en permutant les deux chiffres de `age`. Par exemple, `inverse (42)` retourne `24`. On rappelle que l'opération modulo, notée `%`, permet de calculer le reste de la division entière. Ainsi, par exemple,* $7$ *modulo* $3$*, noté `7 % 3` en* Java*, vaut* $1$*.*

***Q2:*** *Écrire une fonction « `boolean answer(int grandFather, int grandSon)` » qui prend en paramètre un entier `grandFather` compris entre* $10$ *et* $99$ *représentant l'âge du grand-père et un entier `grandSon` compris entre* $10$ *et* $99$ *représentant l'âge du fils. Cette fonction retourne `true` si et seulement si ces deux entiers sont une réponse valide à l'énigme décrite plus haut.*

***Q3:*** *Écrire une fonction « `int solveGrandSon (int grandFather)` » qui prend en paramètre un entier `grandFather` compris entre* $10$ *et* $99$ *représentant l'âge du grand-père et qui retourne, s'il existe, un entier `grandSon` compris entre* $10$ *et* $99$ *tel que `answer (grandFather, grandSon)` renvoie `true`. S'il n'existe pas un tel nombre, cette fonction renvoie `-1`. (Indication : Il suffit d'essayer tous les âges compris entre* $10$ *et* $99$*.)*

***Q4:*** *Écrire une fonction « `int[] solve ()` » qui retourne un tableau de taille* $2$ *qui est une solution de l'énigme. (La première case est l'âge du grand-père et la seconde case est l'âge du petit-fils. Ces âges doivent valider l'énigme.)*

□

**Exercice 2 (Guirlande lumineuse – 4 points)**

*Une guirlande lumineuse clignotante est représentée par un tableau `g` de booléens : la lampe numéro `i` est allumée si `g[i]` vaut `true` et éteinte si `g[i]` vaut `false`. Par exemple, une guirlande formées de deux lampes éteintes et d'une lampe allumée sera représentée par un tableau `g` de longueur* $3$ *tel que `g[0]` et `g[1]` valent `false`, et `g[2]` vaut `true`.*

*Dans cet exercice, on vous fait écrire des fonctions qui permettent d'animer la guirlande en suivant des motifs prédéfinis.*

***Q1:*** *Écrire une fonction « `boolean[] create (int n)` » qui prend en paramètre un entier strictement positif `n` et qui retourne un tableau représentant une guirlande de longueur `n` où toutes les lampes d'indice pair sont allumées et toutes les lampes d'indice impair sont éteintes.*

**Q2:** *Écrire une fonction « `boolean[] inverse (boolean[] g)` » qui prend en paramètre un tableau qui représente une guirlande et qui retourne une nouvelle guirlande où toutes les lampes allumées de `g` sont éteintes et toutes les lampes éteintes de `g` sont allumées. Votre fonction ne devra pas modifier le tableau `g`.*

**Q3:** *Écrire une fonction « `boolean[] repeat (int n, boolean[] p)` » qui prend en paramètre un entier positif ou nul `n` et un second tableau `p` représentant un motif. Cette fonction doit retourner une nouvelle guirlande de taille `n` répétant le motif `p` autant de fois que possible. S'il ne reste plus assez de place à la fin de la guirlande pour représenter la totalité de `p` alors on n'insérera que le début de `p`.*

*Par exemple, si `p` vaut*

`{true, false, false}`

*alors `repeat (8, p)` produit le tableau*

`{true, false, false, true, false, false, true, false}`.

**Q4:** *Écrire une fonction « `boolean[] rotate (boolean[] g)` » qui retourne une nouvelle guirlande de même longueur que `g` mais dont les lampes allumées sont décalées d'un cran à gauche par rapport à celles de `g`.*

*La lampe la plus à gauche (celle d'indice $0$) de `g` est allumée si et seulement si la dernière lampe de `rotate(g)` (celle d'indice `arrayLength(l) - 1`) est allumée.*

*Par exemple, si la guirlande `g` vaut :*

`{ true, false, false, true }`

*alors `rotate(g)` vaut :*

`{ false, false, true, true }`

□

**Exercice 3 (Isogrammes – 4 points)**

*Dans cet exercice, on étudie deux façons différentes de déterminer si un mot est un isogramme, c'est-à-dire un mot dont toutes les lettres sont distinctes.*

*On suppose donnée (vous n'avez donc pas à l'écrire) la fonction :*

`int letterRank (String word, int i)`

*qui prend en paramètre un mot `word` formé uniquement de caractères pris dans les lettres minuscules de l'alphabet et un entier positif `i` compris entre $0$ et $n-1$ (inclus) où $n$ est la longueur de `word`. Cette fonction retourne le rang de la lettre numéro `i` de `word` dans l'alphabet. (La première lettre de l'alphabet, la lettre `'a'` a pour rang $0$ et la dernière lettre de l'alphabet, la lettre `'z'` a pour rang $25$.)*

*Par exemple, `letterRank ("foobar", 3)` retourne `1` car la lettre numéro $3$ de `"foobar"` est `'b'` et c'est la lettre de rang $1$ (la seconde) dans l'alphabet.*

**Q1:** *Écrire une fonction « `int[] histogram (String word)` » qui prend en paramètre un mot `word` formé uniquement de caractères pris dans les lettres minuscules de l'alphabet et qui retourne un tableau `h` d'entiers positifs de taille $26$ tel que `h[i]` est le nombre d'occurrences de la lettre de rang `i` dans le mot `word`.*

*Par exemple, `histogram("foobar")` retourne le tableau :*

`{1,1,0,0,0,1,0,0,0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0}`

**Q2:** *À l'aide de la fonction de la question précédente, écrire une fonction « `boolean isIsogram(String word)` » qui retourne `true` si et seulement si le mot `word` est un isogramme.*

**Q3:** *Proposez une modification des fonctions `isIsogram` et `histogram` qui évite de construire totalement l'histogramme dans le cas où le mot `word` n'est pas un isogramme. Vous n'avez pas à réécrire ces fonctions mais seulement à expliquer en quoi consiste la modification.*

□

**Exercice 4 (Drapeau bicolore – 4 points)**

*Dans cet exercice, on exécute une fonction de façon à trouver une erreur dans sa conception. Cette fonction prend en paramètre un tableau de booléens et essaie de le trier en place de façon à ce que tous les booléens `false` se trouvent dans la partie gauche du tableau et que tous les booléens `true` se trouvent dans la partie droite du tableau. Par exemple, cette fonction doit transformer le tableau suivant :*

```
{ false, true, false, true, true }
```

*en :*

```
{ false, false, true, true, true }
```

*Voici le code source de cette fonction :*

```
1   public static void swap (boolean[] t, int i, int j) {
2       boolean tmp;
3       tmp = t[i];
4       t[i] = t[j];
5       t[j] = tmp;
6   }
7
8   public static void sort (boolean[] t) {
9       int left = 0;
10      int right = arrayLength (t) - 1;
11      while (left <= right) {
12          if (t[left]) {
13              swap (t, left, right);
14              right--;
15          } else {
16              left++;
17          }
18          if (! t[right]) {
19              swap (t, left, right);
20              left++;
21          } else {
22              right--;
23          }
24      }
25  }
```

***Q1:*** *En utilisant le modèle d'exécution vu en cours, donnez la séquence d'exécution de l'appel de fonction `sort(t)` où `t` vaut initialement {`true`, `false`}.*

***Q2:*** *Supposons que `t` vaut initialement {`true`, `false`, `true`}. Que vaut `t` après l'appel de fonction `sort(t)` ? (Vous pouvez donner la réponse sans la justifier.)*

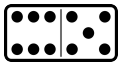
***Q3:*** *D'après les réponses aux questions précédentes, pourquoi la fonction `sort` est-elle incorrecte ?*

***Q4:*** *Quelle modification apporter à la fonction `sort` pour corriger l'erreur explicitée dans la question précédente ? (Indication : Il suffit de modifier une ligne pour corriger ce programme.)*

□

**Exercice 5 (Dominos en solitaire – 4 points)**

*Dans cet exercice, on écrit un programme qui simule quelques aspects du jeu de dominos. Un domino est formé de deux carrés contenant un nombre de points compris entre $0$ et $6$. Voici un exemple de pièce :*



*On représente une pièce de domino comme un tableau d'entiers de taille $2$ où le carré de gauche est représenté par la case d'indice $0$ et le carré de droite par la case d'indice $1$. La pièce précédente est donc représentée par le tableau `{ 6, 5 }`.*

*On dit qu'une pièce $B$ de domino est "posable à droite" d'une pièce $A$ si le nombre de points du carré de gauche de la pièce $B$ est égal au nombre de points du carré de droite de la pièce $A$.*

*Un plateau valide de jeu de domino est une séquence de pièces de domino telle que toute pièce $B$ qui suit une autre pièce $A$ dans la séquence est posable à droite de $A$. En* Java, *on représente un plateau de jeu comme un tableau de pièces, c'est-à-dire un tableau de tableau d'entiers.*

*Un joueur possède un ensemble de pièces de domino. Son objectif est de former le plus long plateau valide à l'aide de ses pièces. (Il est interdit de tourner les pièces.) En* Java, *on représente les pièces initiales d'un joueur comme un tableau de pièces (donc un tableau de tableaux d'entiers). On utilise par ailleurs un tableau de booléens de même taille pour noter les pièces déjà jouées par le joueur : ainsi, la case d'indice* `i` *de ce tableau vaut* `true` *si et seulement si la pièce d'indice* `i` *du joueur a été jouée.*

**Q1:** *Écrire une fonction «* `boolean rightValid (int[] a, int[] b)` *» qui prend en paramètre deux tableaux représentant des pièces de domino et qui retourne* `true` *si et seulement si la pièce* `b` *est posable à droite de* `a`*.*

**Q2:** *Écrire une fonction «* `boolean validSequence (int[][] board, int end)` *» qui prend en paramètre un tableau de tableaux d'entiers représentant le plateau de jeu ainsi qu'un entier* `end` *représentant l'indice dans le plateau de la dernière pièce jouée. Cette fonction retourne* `true` *si et seulement si le plateau est valide.*

**Q3:** *Écrire une fonction «* `int score (int[][] player, boolean[] played)` *» qui prend en paramètre un tableau de tableaux d'entiers représentant les pièces du joueur ainsi qu'un tableau* `played` *indiquant quelles pièces ont été jouées. Cette fonction calcule la somme des points apparaissant sur les dominos joués sur le plateau.*

**Q4:** *Écrire une fonction «* `int play (int[][] board, int[][] player)` *» qui prend en paramètre un plateau* `board` *ne contenant initialement qu'une pièce en position $0$ et l'ensemble des pièces* `player` *du joueur dont aucune n'a encore été jouée.*

*On utilisera un tableau de booléens* `played` *de la même taille que* `player` *pour se souvenir des pièces déjà jouées :* `played[i]` *retourne* `true` *si et seulement si la pièce d'indice* `i` *de* `player` *a été jouée. On utilisera aussi un entier* `end` *correspondant à l'indice dans le plateau de la dernière pièce jouée.*

*Cette fonction retourne le score du joueur qui joue en suivant la stratégie suivante :*

*(a) Calculer l'ensemble $C$ des pièces du joueur posables à droite de la dernière pièce posée sur le plateau. (Indice : cet ensemble peut être représenté par un tableau de la même taille que* `player`*.)*

*(b) Si l'ensemble $C$ est vide alors arrêter.*

*(c) Poser une pièce $a$ prise dans $C$ telle qu'il existe une autre pièce $b$ dans l'ensemble des pièces du joueur telle que $b$ est posable à droite de $a$. S'il n'existe pas une telle pièce, poser une pièce de $C$ sur le plateau et arrêter.*

*(d) Reprendre en (a).*

*(Il est fortement conseillé d'introduire des fonctions auxilaires pour répondre à cette question.)*

□