**IUT de Montpellier** **TD1 : bases de la récursivité**
**R3.02 Développement efficace**

# 1 Exercices de base sur les entiers

**Exercice 1. Factorielle**
Ecrire un algorithme récursif `int factorielle(int n)` qui pour tout $n \in \mathbb{N}^*$ calcule $n!$.

**Exercice 2. Pair**
Ecrire un algorithme récursif `boolean pair(int n)` qui pour tout $n \in \mathbb{N}$ retourne vrai ssi $n$ est pair.

**Exercice 3. Somme impairs**
Ecrire un algorithme récursif `int sommeImpairs(int n)` qui pour tout $n \geq 1$, $n$ impair, calcule $1+3+5+\cdots+n$.

**Exercice 4. Puissance**
Ecrire un algorithme récursif `int puiss(int x, int n)` qui pour tout entier $x$ et pour tout $n \in \mathbb{N}$ calcule $x^n$. Combien de multiplications fait votre algorithme ? Nous écrirons une autre version de cet algorithme de type "diviser pour régner" plus tard, qui fait beaucoup moins de multiplications.

# 2 Exercices de base sur les tableaux

**Exercice 5. Nombre d'occurences**

**Question 5.1.**
Ecrire un algorithme récursif `int nbOccAux(int x, int []t, int i)` qui pour tout entier $x$, tableau $t$ non vide et pour tout $i$ tel que $0 \leq i < t.length$ calcule le nombre d'occurences de $x$ dans le sous tableau $t[i..(t.length - 1)]$.

**Question 5.2.**
En déduire un algorithme (non récursif .. mais sans boucle !) `int nbOcc(int x, int []t)` qui calcule le nombre d'occurences de $x$ dans $t$. Indication : appelez nbOccAux.

**Question 5.3.**
On remarque que pour l'instant nbOccAux ne supporte pas les tableaux vides (car il faudrait donner $i$ tel que $0 \leq i < 0$). On va donc changer la spécification de nbOccAux. Ecrire un algorithme `int nbOccAux2(int x, int []t, int i)` qui pour tout entier $x$, tableau $t$ et pour tout $i$ tel que $0 \leq i \leq t.length$ calcule le nombre d'occurences de $x$ dans le sous tableau $t[i..(t.length - 1)]$. Conclusion : nbOccAux2 supporte maintenant les tableaux vides, et on s'aperçoit que le cas de base était plus court, on est gagnant des deux côtés ! Il faudra donc s'habituer à ces spécifications où les indices ont le droit de sortir du tableau.

**Exercice 6. Palindrome**
En s'inspirant de la démarche de l'exercice précédent (c'est à dire en écrivant un estPalindromeAux(.....) **pour lequel vous indiquerez votre spécification**, écrire un algorithme `boolean estPalindrome(char []t)` qui détermine si $t$ est un palindrome. Par exemple, $t = ['a','b','c','b','a']$ est un palindrome, $t = ['a','b','b','a']$ est un palindrome, mais $t = ['a','b','c','e','a']$ n'est pas un palindrome.

**Exercice 7. Croissants**
En s'inspirant de la démarche de l'exercice précédent, écrire un algorithme `boolean estCroissant(int []t)` qui détermine si les éléments de $t$ sont rangés par ordre croissant.

# 3 Exercices plus difficiles

**Exercice 8. Pavage avec des dominos**
On considère une grille de 2 cases de haut et $n \geq 1$ cases de large, ainsi que des dominos de taille $2 \times 1$ (que l'on peut placer horizontalement ou verticalement sur le damier). On appelle **paver** une grille le fait de disposer des dominos pour

couvrir toutes les cases, sans laisser de trous, sans que deux dominos se chevauchent, et sans qu'un domino soit à moitié sur la grille et à moitié dehors.

**Question 8.1.**
Dessinez les 3 façons de paver une grille de 2 cases de hauteur 3 cases de largeur (placer tous les dominos verticalement ne donne donc qu'un pavage, peut importe leur ordre).

**Question 8.2.**
Ecrire un algorithme récursif `int f(int n)` qui pour tout entier $n \geq 1$ calcule le nombre de façons de paver une grille de 2 cases de haut et de $n$ cases de large. Par exemple, pour $n = 3$, $f(3)$ doit retouner 3.

**Question 8.3.**
Et $f$ .. vous la reconnaissez ?

**Exercice 9. Hanoï**
Le but de cet exercice est d'écrire le jeu des tours de Hanoï, célèbre chez les informaticiens[1] Dans ce jeu on considère 3 poteaux (dénommés "1" (à gauche), "2" (au milieu), et "3" (à droite)), ainsi que $N$ disques de diamètres deux à deux distincts. Les disques sont troués en leur centre, de telle sorte que l'on puisse les enfiler sur les poteaux. Dans la situation initiale, les $N$ disques sont sur le poteau gauche, et rangés "en pyramide" : c'est à dire de telle sorte que les plus petits disques sont au dessus (voir Figure 1).

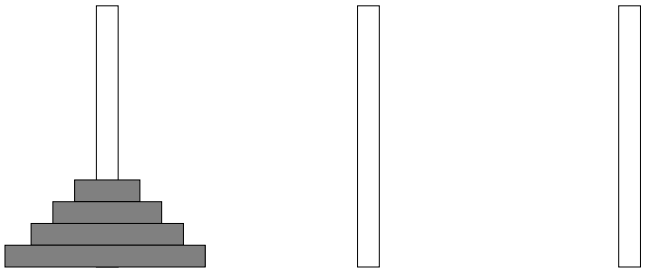

Figure 1: Exemple avec $N = 4$.

Le but du jeu est de déplacer cette pyramide sur le poteau de droite, en sachant qu'un coup légal consiste à

- choisir un poteau de départ, et prendre le disque du dessus
- choisir un poteau d'arrivée, et déposer le disque au sommet
- s'assurer que sur chaque poteau les disques sont rangés par *ordre décroissant*, c'est à dire avec les plus petits disques au sommet (autrement dit, un disque ne peut être placé que sur un disque de plus grande taille).

Par exemple, la succession de coups "1 -> 2", "1 -> 3", "2 -> 3" est légale, alors que la succession de coups "1 -> 2", "1 -> 2" ne l'est pas. On souhaite écrire une méthode `void resoudre(int n)` qui pour tout $n \geq 1$, affiche une liste de coups qui en partant de l'état initial du jeu (les $n$ disques sur le poteau 1) mène à l'état final (tous les disques sur le poteau 3).

**Question 9.1.**
Pourquoi est-ce que écrire "directement" la méthode `void resoudre(int n)` récursivement semble difficile ?

**Question 9.2.**
On va donc écrire d'abord la méthode `void resoudreAux(int n, int i, int k, int j)` qui pour tout $n \geq 1$ affiche une liste de coups permettant, étant donné un jeu de Hanoï avec $n$ disques initialement sur le poteau $i$, de déplacer ces $n$ disques sur le poteau $j$ en se servant si besoin du poteau intérmediaire $k$ (avec $i$, $j$, $k$ dans $\{1, 2, 3\}$, deux à deux distincts).
Par exemple, resoudreAux(2,2,3,1) affiche:
"2 -> 3"
"2 -> 1"
"3 -> 1"
Remarquez le fait suivant contre-intuitif. La méthode `resoudreAux` est plus générale que `resoudre` (puisque l'on peut choisir ses poteaux de départ/arrivée), et pourtant on parvient à l'écrire plus facilement que `resoudre`. Pourquoi ? Car les appels récursifs sont eux-aussi plus généraux, et nous permettent donc de faire plus de choses.

[1] En particulier pour la découverte de la récurrence!

**Question 9.3.**
En déduire la méthode `void resoudre(int n)`

**Question 9.4.**
Soit $u_n$ le nombre de coups qu'utilise resoudre(n) pour résoudre Hanoï à $n$ disques. Que vaut $u_n$ par rapport à $u_{n-1}$ ? Une fois votre suite définie par récurrence, déterminez la valeur de $u_n$.

# 4 Exercices bonus

**Exercice 10. PGCD**
Nous allons écrire l'algorithme récursif d'Euclide pour calculer le PGCD de deux entiers naturels.

**Question 10.1.**
Soient $a$ et $b$ dans $\mathbb{N}$, avec $b > 0$. Soient $q$ et $r$ dans $\mathbb{N}$ tels que $a = bq + r$, avec $0 \leq r < b$. Montrer que pour tout $x$, ($x$ divise $a$ et $x$ divise $b$) $\Leftrightarrow$ ($x$ divise $b$ et $x$ divise $r$).

**Question 10.2.**
Soient $a$ et $b$ dans $\mathbb{N}$, avec $b > 0$. Montrer que $PGCD(a, b) = PGCD(b, r)$.

**Question 10.3.**
En déduire un algorithme récursif `int PGCD(int a,int b)` qui pour tout $a$ et $b$ dans $\mathbb{N}$ calcule le pgcd de $a$ et $b$.