# Travaux Pratiques d'algorithmique n°2

**Cours d'Informatique de Deuxième Année**

*—L2.1—*

---

## Récursivité

L'objet de cette séance est d'introduire la notion d'algorithmes récursifs.

---

▸ **Exercice 1.** ***(Critères de divisibilité)***

- *On veut calculer la somme des chiffres d'un entier positif. Par exemple, la somme des chiffres formant* 3281 *est* 14.

  *Écrire une fonction* `SommeChiffre` *prenant en argument un entier* `n` *et qui calcule récursivement la somme des chiffres qui le forment.*

- *Le critère de divisibilité suivant est bien connu : un entier est divisible par* 3 *si la somme des chiffres qui le forment est elle-même divisible par* 3*. Autrement dit :*

$$n \equiv 0 \pmod 3 \;\Leftrightarrow\; \sum_{i=0}^{k} c_i \;\equiv\; 0 \pmod 3$$

  *où $c_i$ est le $i$ème chiffre de $n$ en partant de la droite.*

  *On remarquera que la somme des chiffres d'un entier supérieur à* 10 *est toujours strictement plus petite que cet entier. Cette simplification (passer d'un entier à la somme de ses chiffres) fait naître un algorithme récursif.*

  *Un entier $n$ est-il divisible par* 3 *? Si l'entier est strictement plus petit que* 10 *: facile à déterminer ; sinon on se pose la même question sur la somme de ses chiffres.*

  *Effectuer le calcul complet pour* 3281 *et* 1847394*.*

  *Écrire une fonction* `MultDe3` *récursive réalisant cet algorithme ( donc sans utiliser de tests comme n%3==0).*

- *Effectuer le même travail pour la divisibilité par* 11 *:*

$$n \equiv 0 \pmod{11} \;\Leftrightarrow\; \sum_{i=0}^{k} (-1)^i c_i \;\equiv\; 0 \pmod{11}$$

*On utilisera une nouvelle fonction* `SommeAlternée` *pour laquelle on remarquera que :*

$$\begin{aligned} SommeAlternee(17328) &= 8-2+3-7+1 \\ &= 8-(2-(3-(7-(1)))) \end{aligned}$$

▸ **Exercice 2.** ***(Décomposition d'un entier)***
*Décomposer un entier* $n(n >= 0)$*, dans la base* $b$ $(2 \leq$ `b`$)$*, c'est trouver la suite* $c_p, c_{p-1}, \ldots, c_0$ *avec* $0 \leq c_i < b, 0 \leq i \leq p$ *telle que:*
$n = c_p b^p + c_{p-1} b^{p-1} + \ldots + c_0$*. Écrire une fonction de prototype* `void AffDecomp(int n, int b)` *qui affiche la décomposition de l'entier* `n` *dans la base* `b` *(*`n`$\geq 0$, $2 \leq$ `b`$\leq 10$*). Par exemple, pour* `n` $= 172$ *et* `b` $= 10$*, l'affichage sera* 1 7 2 *; et, pour* `n` $= 172$ *et* `b` $= 8$*, l'affichage sera* 2 5 4*. Pour cela, vous utiliserez un algorithme récursif.*

▸ **Exercice 3.** ***(Plus Grand Commun Diviseur)***
*On souhaite calculer le PGCD de deux entiers* $a$ *et* $b$*. On suppose tout d'abord* $a \geq b \geq 0$ *et on suppose que* $a$ *et* $b$ *ne sont pas tous les deux nuls. La méthode proposée est l'algorithme d'Euclide (*$\approx 300$ *AV. J.C) ;*

$$PGCD(a,b) = \begin{cases} a & \text{si } b = 0, \\ PGCD(b, a \bmod b) & \text{sinon.} \end{cases}$$

1. *Calculer à la main* $PGCD(142, 12)$*.*
2. *Que se passe-t-il si on applique l'algorithme aux entiers* 12 *et* 142 *(de façon générale, aux entiers* $a$ *et* $b$ *avec* $a < b$*) ? Pourquoi ?*
3. *Implémenter cet algorithme de façon récursive. On supposera* $a \geq 0$ *et* $b \geq 0$ *(on convient que* $PGCD(0,0)$ *est égal à zéro).*
4. *En déduire une fonction qui calcule le PGCD de deux entiers* $a$ *et* $b$ *quelconques.*

▸ **Exercice 4.** ***(Nombres de Fibonacci)***
*Les nombres de Fibonacci sont définis par la relation :*

$$\begin{cases} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \quad \text{pour } n \geq 2 \end{cases}$$

1. *Écrire une fonction C récursive,* `FiboR`*, qui calcule le nième nombre de Fibonacci.*
2. *Vérifier (par votre programme) que le nombre d'appels à la fonction* `FiboR` *pour calculer* $F_n$ *est exactement égal à* $2 * F_{n+1} - 1$*.*
3. *Écrire une fonction C itérative qui calcule le nième nombre de Fibonacci. Combien y a-t-il de passage dans la boucle?*
4. *Soient* $F_{k+1}$ *et* $F_k$ *deux nombres successifs de la suite de Fibonacci. Que vaut* $PGCD(F_{k+1}, F_k)$ *? Combien d'appels à la fonction sont nécessaires pour calculer* $PGCD(F_{k+1}, F_k)$ *?*