# J1MI2013: Algorithmes et Programmes: feuille 3

### Itérer avec la boucle while

## Travaux dirigés

**Exercice 1.** Division euclidienne

Soient `a` un entier positif et `b` un entier strictement positif. Effectuer la division euclidienne (division entière) de `a` par `b`, c'est déterminer l'unique couple d'entiers $(q, r)$ tels que $a = bq + r$ et $0 \leq r < b$.

1. Écrire la partie de code (ou bloc d'instructions), comportant une boucle `while`, qui permet de calculer la division euclidienne en utilisant l'algorithme par soustractions successives.

   Faire tourner l'algorithme pour $a = 13$ et $b = 3$ ; pour $a = 2$ et $b = 7$.

2. En `C` l'instruction `return` ne permet pas de renvoyer plus d'une valeur à la fois. Écrire donc deux fonctions `int reste(int a, int b)` et `int quotient(int a, int b)` qui, grâce à l'algorithme précédant, calculent et renvoient respectivement le reste et le quotient de la division entière de `a` par `b`.

**Exercice 2.** On rappelle que, dans une base `b` :

- le reste de la division de `n` par `b` est égal à la valeur du dernier chiffre de `n` (celui situé le plus à droite)
- le quotient de la division entière de `n` par `b` est égal à la valeur représentée par le nombre `n` privé de son dernier chiffre.

Écrire une fonction `int sommeChiffres(int n)` qui calcule et renvoie la somme des chiffres décimaux d'un nombre entier `n` passé en paramètre[1].

**Exercice 3.** Soit la fonction `mystere` suivante :

```
1 int mystere(int n, int x){
2   while ((n/x > x) && (x!=0)){
3     n = n/10;
4     x--;
5   }
6   return n;
7 }
```

- Quelle est la valeur renvoyée par `mystere(100,6)` ?
- Quelle est la valeur renvoyée par `mystere(1000,3)` ?

Corriger la condition d'itération afin qu'elle ne produise jamais d'erreur.

1. En `C` l'opérateur `%` de modulo permet d'obtenir le reste, l'opérateur `/` de division permet d'obtenir le quotient entier si les deux opérandes sont des nombres entiers.

**Exercice 4.** On a écrit la fonction qui calcule $2^n$ en utilisant la boucle `for` :

```
1 int puissanceDe2(int n){
2   int p=1;
3   for(int i=1; i<=n; i++){
4     p = p*2;
5   }
6   return p;
7 }
```

1. Donner une deuxième version de ce programme en utilisant l'instruction `while` au lieu de `for`.
2. Il existe en C une troisième structure de boucle, le `do-while` dans laquelle la condition d'arrêt est testée après chaque passage dans le corps de la boucle, qui s'exécute donc au moins une fois. La syntaxe du `do-while` est :

```
do {
  instructions ;
} while (expression) ;
```

   Le bloc d'instructions est exécuté, puis l'expression est évaluée. Si elle est vraie on exécute à nouveau le bloc d'instructions. La boucle se termine quand l'expression devient fausse.

   Donner une troisième version de ce programme en utilisant l'instruction `do-while`.
3. Vos trois versions de la fonction fonctionnent-elles si `n` est négatif ou nul ?

**Exercice complémentaire 1.** Multiplication alexandrine

Pour multiplier deux entiers naturels $x$ et $y$, on répète, tant que $y$ n'est pas nul, la double opération suivante : *multiplier* $x$ *par 2 et diviser* $y$ *par 2 (quotient entier)*. Le résultat est égal à la somme des multiples de $x$ correspondant à des quotients de $y$ impairs (le couple initial compris).
Cela revient à déterminer la suite :

- $(x_0, y_0) = (x, y)$
- $(x_{n+1}, y_{n+1}) = (2x_n, \frac{y_n}{2})$ (où le quotient est un quotient entier)

Le produit $xy$ est égal à la somme des $x_n$ pour lesquels $y_n$ est impair.

1. Faire tourner l'algorithme pour calculer $7 \times 9$ (donner le tableau des valeurs successives de $n$, $x_n$ et $y_n$).
2. Écrire une fonction `multiplicationAlexandrine` implémentant cet algorithme.
   Déterminer, en fonction de $x$ et de $y$, le nombre d'opérations élémentaires effectuées (multiplications par 2, divisions par 2, comparaisons, additions).
3. Sur le modèle de la multiplication alexandrine, imaginer un algorithme permettant de calculer $x^n$.