TD 17 : Entiers naturels (2)

```
chiffres = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Exercice 1 (Conversion vers une base B ($1 < B \leq 62$))

L'algorithme du cours a montré comment réaliser la conversion vers une base $B \leq 10$ d'un entier strictement positif. L'objectif de cet exercice est de généraliser cet algorithme pour les bases B dans l'intervalle $2 \leq B \leq 62$.

- Rappeler l'algorithme de conversion vers la base B pour $2 \leq B \leq 10$.
- Les symboles supplémentaires utilisés lorsque $B \geq 10$ sont les lettres minuscules, puis les lettres majuscules, ce qui permet d'avoir 62 symboles différents. Modifier le programme de conversion précédent pour qu'il accepte une base B quelconque dans l'intervalle $2 \leq B \leq 62$. Pourquoi utiliser la variable `chiffres` ?
- Proposer une version récursive de cet algorithme.

Certains opérateurs travaillent, non sur les valeurs fournies, mais directement sur les représentations binaires de ces valeurs : ce sont les opérateurs *bitwise*. Les opérateurs de décalage de bits vers la gauche (resp. la droite) sont « (resp. »).
`1 « 4` renvoie $(10000)_2 = (2^4)_{10} = (16)_{10}$: cela revient donc à multiplier `1` par 2^4.
De même, `23 » 2` donne `5` car $(23)_{10} = (10111)_2$ et $(5)_{10} = (101)_2$: `23//2**2=5`.
Une documentation succincte est fournie avec cet énoncé.

Exercice 2 (Produit binaire)

Soit $(n, m) \in \mathbb{N}^2$. La décomposition en base 2 de `n` s'écrit $n = (b_p b_{p-1} \ldots b_1 b_0)_2$.

1. Exprimer le produit nm en fonction de m et des b_k.
2. En déduire un algorithme pour effectuer le produit de 2 nombres représentés en binaire, uniquement à l'aide d'additions, de décalages et de test sur un bit.
3. Traduire cet algorithme en une fonction `multiplie(n,m)` qui retourne le produit de 2 entiers non signés.

Exercice 3 au dos

Exercice 3 (Code Gray)

Quand on augmente un nombre binaire, comme passer de 1 à 2 en binaire (on passe de 01 à 10), on est amené à modifier 2 bits simultanément, ce qui peut poser des problèmes de stabilité en électronique et des difficultés pour vérifier les changements effectués. On peut alors utiliser un code dit *Gray* qui permet de ne modifier qu'un seul bit quand on augmente d'une unité un nombre (comme par exemple un compteur, un capteur de rotation...). Ainsi le passage de 1 à 2 en code Gray correspond au passage en binaire du code 01 à 11.

On passe du code binaire de `n` à son code Gray avec l'algorithme suivant : lire le mot binaire représentant `n` depuis la gauche, inverser chaque bit dont le bit précédant (à gauche) est égal à 1.

On passe du code Gray de `n` au code binaire en laissant le bit de poids fort inchangé, puis à partir de là, pour chaque bit suivant vers la droite, on réalise un `ou exclusif` avec le bit que l'on vient de trouver.

- Donner le code Gray des entiers de 0 à 15, ainsi que leur représentation binaire.
- Ecrire une fonction `bin2gray(n)` qui renvoie le code Gray de `n`.
  `list(bin(n)[2:])` renvoie un tableau contenant le code binaire de `n`.
- Ecrire une fonction `gray2bin(g)` qui renvoie la représentation binaire de `g`.
- Reprendre les deux questions précédentes en utilisant les opérateurs sur les bits, appelés *opérateurs bitwise* (cf doc jointe). On commencera par chercher ce que représente un ou exclusif entre la représentation binaire de `n` et la représentation binaire de `n//2`.