

TD 6 : Codage de l'information

Conversion décimal - binaire

1. Écrire un programme qui convertisse un nombre décimal en binaire.

On donne l'algorithme suivant :

```

Algorithme dec2bin
{ conversion décimal → binaire pour des entiers  $\geq 0$  }
variables
    nombre : entier
    binaire : tableau de [1..N] entiers
début
    lire(nombre)
    binaire.ajout(nombre modulo 2) { ajout en fin de tableau }
    tantque nombre  $\geq 2$  faire
        nombre := nombre / 2
        binaire.ajout(nombre modulo 2)
    inverse(binaire) { inverse les éléments d'une liste }
    afficher(binaire)
fin
  
```

2. Montrer que l'algorithme se termine pour des entiers ≥ 0 . *car le tant que est ≥ 2 donc cela va le faire se diviser jusqu'à 1.*
3. Montrer que l'assertion $0 \leq \text{binaire}[i] \leq \text{nombre}$ avant, pendant et après la boucle tant que est toujours vraie (invariant de boucle).

Tout au début du programme, on placera toujours ces 2 lignes :

```
#!/usr/bin/env python3
# coding: utf-8
```

4. Indiquer l'utilité de la 2ème ligne.
5. Donner la conversion de 1843 en binaire. *$(11100110011)_2 = 1843$*
6. Convertir -2 et conclure. *sa donne 0*

Pour éviter que le programme « crash » sur une erreur de valeur d'entrée (ValueError), une façon de gérer cette erreur est d'intercepter le message d'erreur et de le traiter correctement à l'aide d'un assert.

La syntaxe est la suivante :

```
try:
    # bloc d'instructions à exécuter
except ValueError:
    # afficher « Erreur de valeur »
```

7. Vérifier la cohérence et la stabilité du programme avec quelques tests : 1, -1, 0, 1.1, -1.0, a, abc

Défi du Big Boss : On veut créer une fonction `add_bin` qui additionne deux nombres binaires de 4 bits.

```
def add_bin(a:list, b:list) -> list:
```