



## TME2 – Fonctions (suite) – Listes

**Exercice 2.1** (*Représentation binaire inversée d'un entier* – Extrait revisité de l'examen de Janvier 2023).

La représentation binaire *inversée* d'un entier naturel  $n$  est la séquence de bits  $b_0b_1\cdots b_N$  telle que  $n = \sum_{i=0}^N b_i 2^i$  ( $b_N$  est le bit de poids fort et  $b_0$  est le bit de poids faible). Notez que  $b_0 = n \bmod 2$ , et que  $(b_1 \cdots b_N)$  représente l'entier  $n / 2$ . Par exemple, 110101 est la représentation binaire inversée de l'entier 43 puisque :

$$43 = (1 \times 2^0) + (1 \times 2^1) + (0 \times 2^2) + (1 \times 2^3) + (0 \times 2^4) + (1 \times 2^5) = 1 + 2 + 0 + 8 + 0 + 32$$

On encode une représentation binaire inversée d'un entier par une liste d'entiers ne contenant que des 0 et des 1. Par exemple, en binaire inversé, 2 peut être représenté par la liste  $[0; 1]$  et 43 peut être représenté par la liste  $[1; 1; 0; 1; 0; 1]$ . En représentation binaire inversée, les bits de poids faibles sont en début de liste (c'est-à-dire à gauche).

On remarquera que si  $l$  est une liste représentant un entier  $i$ , alors la liste  $1 @ [0]$  obtenue en ajoutant 0 à la fin de la liste  $l$  représente le même entier  $i$  : en effet, ajouter un 0 en position de poids fort ne change pas l'entier représenté (car cela correspond à l'ajout d'un 0 non significatif). Il existe donc plusieurs représentations binaires inversées d'un même entier. Par exemple, les listes  $[0; 1; 1]$ ,  $[0; 1; 1; 0]$  et  $[0; 1; 1; 0; 0]$  représentent toutes l'entier 6.

1. Définir une fonction de signature `bin_to_int (i : int list) : int` calculant un entier à partir de sa représentation binaire inversée. On suppose ici que la liste  $i$  ne contient que des entiers égaux à 0 ou 1 et la fonction lèvera une exception si la liste  $i$  est vide.

```
# bin_to_int [];;                                # bin_to_int [0; 1; 1];;
Exception: Invalid_argument "empty list".      - : int = 6
# bin_to_int [1];;                               # bin_to_int [1; 1; 0; 1; 0; 1];;
- : int = 1                                     - : int = 43
```

2. Définir une fonction de signature `int_to_bin (i : int) : int list` calculant la représentation binaire inversée la plus courte d'un entier naturel  $i$ .

```
# int_to_bin 0;;                                # int_to_bin 6;;
- : int list = [0]                             - : int list = [0; 1; 1]
# int_to_bin 1;;                                # int_to_bin 43;;
- : int list = [1]                             - : int list = [1; 1; 0; 1; 0; 1]
```

3. Définir une fonction de signature `comp_bin (l : int list) (n : int) : int list` qui construit la liste obtenue en ajoutant à la fin de la liste  $l$  les entiers 0 pour obtenir une liste de longueur  $n$ . Si  $n$  est strictement inférieur à la longueur de  $l$ , alors la fonction `comp_bin` lève l'exception `Invalid_argument`; si  $n$  est égal à la longueur de  $l$ , alors la fonction `comp_bin` retourne la liste  $l$ .

```
# comp_bin [1; 0] 2;;                          # comp_bin [1; 0] 4;;
- : int list = [1; 0]                          - : int list = [1; 0; 0; 0]
# comp_bin [0; 1; 1] 2;;
Exception: Invalid_argument "comp_bin".
```

**Exercice 2.2** (Crible d'Eratosthène).

Le crible d'Eratosthène permet d'obtenir la liste des premiers nombres premiers<sup>1</sup>. Cette liste est construite de manière incrémentale : le  $i$ -ième élément est le plus petit entier supérieur au  $(i-1)$ -ième élément non divisible par tous les éléments qui le précèdent dans la liste. Le premier élément de cette liste est 2.

1. Rappelons qu'un nombre est premier s'il n'est divisible que par lui-même et par 1. D'autre part, 1 n'est pas un nombre premier.

1. Définir une fonction de signature `genere_list (n : int) : int list` qui construit la liste des entiers consécutifs de 2 à  $n$  ( $[2; 3; \dots; n]$ ) si  $n \geq 2$  et retourne la liste vide sinon. On pourra utiliser la fonction `range_inter` définie dans l'exercice ?? du TD2.

```
# (genere_list 1);;      # (genere_list 4);;
- : int list = []      - : int list = [2; 3; 4]
```

2. Définir une fonction de signature `elimine (l : int list) (n : int) : int list` qui construit la liste des éléments de  $l$  privée des multiples de  $n$  (c-à-d privée des éléments que  $n$  divise).

```
# (elimine [1; 2; 3; 4; 5; 6] 3);;
- : int list = [1; 2; 4; 5]
```

3. Définir une fonction de signature `ecreme (l : int list) : int list` qui construit la liste des éléments de  $l$  en ne conservant que les entiers ayant la propriété de n'être multiple d'aucun des entiers qui les précèdent dans  $l$ .

```
# (ecreme [2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12]);;
- : int list = [2; 3; 5; 7; 11]
```

Il suffit ici d'enlever de la liste  $[3; 4; 5; 6; 7; 8; 9; 10; 11; 12]$  tous les multiples de 2 pour obtenir la liste  $[3; 5; 7; 9; 11]$  puis d'enlever tous les multiples de 3 de la liste  $[5; 7; 9; 11]$  pour obtenir la liste  $[5; 7; 11]$  puis d'enlever tous les multiples de 5 de la liste  $[7; 11]$  ... et ainsi de suite.

4. Définir une fonction de signature `crible (n : int) : int list` qui construit la liste des nombres premiers inférieurs ou égaux à  $n$ .

```
# (crible 1);;          # (crible 20);;
- : int list = []      - : int list = [2; 3; 5; 7; 11; 13; 17; 19]
```