TD1 – Premières fonctions

Exercice 1.1 (Définition et application de fonctions).

1. Définir une fonction de signature `double (x : int) : int` qui calcule l'entier $2x$ à partir de l'entier x.

```
# (double 3);;
- : int = 6
```

2. En utilisant la fonction `double`, définir une fonction de signature

   `somme_2double (x : int) : int`

   qui calcule $2x + 4x$ à partir de l'entier x.

```
# (somme_2double 3);;
- : int = 18
```

   Dans le corps de la fonction `somme_2double`, combien de fois est calculé `(double x)` ? Est-il possible de définir une version de cette fonction ou `(double x)` n'est calculé qu'une unique fois ?

3. Définir une fonction de signature `make_even (x : int) : int` qui étant donné un entier x retourne x si x est pair et retourne $2x$ sinon.

```
# (make_even 6);;          # (make_even 5);;
- : int = 6                - : int = 10
```

   On pourra utiliser la fonction prédéfinie `mod` qui calcule le reste de la division entière de deux entiers :

```
# 5 mod 2;;                # 4 mod 2;;
- : int = 1                - : int = 0
```

Exercice 1.2 (Expressions conditionnelles / expressions booléennes).

1. Montrer que les expressions :
   (a) `if a then true else (f a)`
   (b) `if a then a else (f a)`
   (c) `a || (f a)`

   ont les mêmes valeurs pour tout booléen `a` et toute fonction booléenne (unaire) `f`.

2. Montrer que les expressions :
   (a) `if a then (f a) else false`
   (b) `if a then (f a) else a`
   (c) `a && (f a)`

   ont les mêmes valeurs pour tout booléen `a` et toute fonction booléenne (unaire) `f`.

Exercice 1.3 (Somme des premiers entiers naturels impairs).

1. Définir une fonction de signature `sum_impairs (n : int) : int` qui calcule la somme des n premiers entiers naturels impairs $(1 + 3 + \cdots + (2n - 1))$.

```
# (sum_impairs  4);;
- : int = 16
```

2. Définir une fonction de signature `sum_impairs_inf (n : int) : int` qui calcule la somme de tous les entiers naturels impairs strictement inférieurs à n.

```
# (sum_impairs_inf 1);;    # (sum_impairs_inf 8);;    # (sum_impairs_inf 9);;
- : int = 0                - : int = 16               - : int = 16
```

3. Combien de tests de parité sont effectués lors de l'exécution de la fonction `sum_impairs_inf` de la question précédente sur un entier `n` ? Proposer une définition de cette fonction où ce test n'est plus effectué (*indication* : utiliser la fonction `sum_impairs`).

Exercice 1.4 (Suite récurrente)**.**

Soit $(u_n)_{n\in\mathbb{N}}$ la suite récurrente définie par : $\begin{cases} u_0 = 2 \\ u_{n+1} = 3u_n + 4 \end{cases}$

1. Définir une fonction de signature `u_n (n : int) : int` qui calcule le n-ième terme u_n de la suite.

```
# (u_n  0);;         # (u_n  3);;
- : int = 2          - : int = 106
```

2. En utilisant la fonction `u_n`, définir une fonction de signature `sum_un (n : int) : int` qui calcule la somme $\sum_{i=0}^{n-1} u_i$ des n premiers termes de la suite.

```
# (sum_un 0);;       # (sum_un 1);;       # (sum_un 3);;
- : int = 0          - : int = 2          - : int = 46
```

Exercice 1.5 (Représentation binaire d'un entier naturel)**.**

On rappelle que la représentation binaire d'un entier naturel n est la séquence de bits $b_{n-1}\cdots b_1 b_0$ telle que $n = \sum_{i=0}^{n-1} b_i 2^i$ (b_{n-1} est le bit de poids fort et b_0 est le bit de poids faible). Par exemple 10011 est la représentation binaire de l'entier 19 puisque :

$$19 = (1 \times 2^0) + (1 \times 2^1) + (0 \times 2^2) + (0 \times 2^3) + (1 \times 2^4) = 1 + 2 + 0 + 0 + 16$$

1. Définir une fonction de signature `nb_un (n : int) : int` qui calcule le nombre de 1 contenus dans la représentation binaire d'un entier n.

```
# (nb_un 19);;
- : int = 3
```

Indication. Le bit b_0 le plus à droite vaut 1 si n est impair et 0 sinon et l'entier naturel représenté par les bits $b_{n-1}\cdots b_1$ est l'entier k tel que $n = (2 \times k) + b_0$. Par exemple, $19 = (2 \times 9) + 1$ et la représentation binaire de 9 est 1001.

2. Définir une fonction de signature `nb_bits (n : int) : int` qui calcule le nombre de bits minimum contenus dans la représentation binaire d'un entier n.

```
# (nb_bits 19);;
- : int = 5
```

Indication. Combien de fois faut-il diviser 19 par 2 pour obtenir un quotient nul ?

3. Définir une fonction de signature `nb_max (n : int) : int` qui calcule le plus grand entier naturel que l'on peut représenter avec $n > 0$ bits.

```
# (nb_max 3);;
- : int = 7
```