UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

t.d.

Programmation fonctionnelle

informatique / licence 2

1. Écrire le code en langage Scheme correspondant à l'expression $3x+5y-3$.

2. Écrire le code en langage Scheme d'une expression équivalente à l'assertion « n n'est divisible ni par 3 ni par 5 ».

3. On donne ci-dessous un code pour la fonction factorielle ainsi qu'une version très naïve du calcul des termes de la suite de Fibonacci.

```
1 (define (factorielle n)
2   (if (< n 2) 1
3     (* n (factorielle (- n 1)))))
```

```
1 (define (fibonacci n)
2   (if (< n 2) n
3     (+ (fibonacci (- n 2))
4        (fibonacci (- n 1)))))
```

On réécrit maintenant les expressions selon le modèle suivant :

`(factorielle 4)` devient `(* 4 (* 3 (* 2 1)))`

Selon le même modèle, réécrire `(fibonacci 5)`.

4. Analyser la fonction « mystère » suivante, prenant un argument strictement positif ; en donner la spécification dans les termes les plus clairs possibles.

```
1 (define (mystere n)
2   (if (< n 2) 0
3     (+ 1 (mystere (quotient n 2)))))
```

5. Analyser la fonction « mystère » suivante, prenant un argument strictement positif ; en donner la spécification dans les termes les plus clairs possibles.

```
1 (define (mystere2 n)
2   (if (zero? n) 0
3     (+ (modulo n 10) (mystere2 (quotient n 10)))))
```

6. Écrire le code d'une fonction récursive `hamming-weight` donnant le poids de Hamming d'un entier positif, c'est-à-dire le nombre de bits de valeur 1 dans son écriture binaire. Par exemple, le poids de Hamming de 23 est 4.

7. Écrire le code d'une fonction récursive `collatz-length` prenant en argument un entier n strictement positif et renvoyant le nombre d'étapes nécessaires pour arriver à la valeur 1 en répétant le processus suivant :

$$\begin{cases} n \leftarrow n/2 & \text{si } n \text{ est pair} \\ n \leftarrow 3n+1 & \text{si } n \text{ est impair} \end{cases}$$

jusqu'à ce que l'on arrive à $n=1$. (Par exemple, en partant de $n=7$, on passe par les termes successifs 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2 et 1, soit 16 étapes en tout.)

8. Écrire le code d'une fonction récursive `isqrt` calculant la « racine carrée entière » d'un entier, c'est-à-dire la partie entière de la racine carrée, *par dichotomie* : on encadre initialement la racine carrée de n entre 0 et n, puis on prend le milieu de l'intervalle, on le met au carré, on teste s'il est trop petit ou trop grand, etc.