Institut National
Universitaire
Champollion

Contrôle continu n°1

L1 DLMI
7 novembre 2019

PROGRAMMATION FONCTIONNELLE CAML
Durée : 1h30 - Aucun document autorisé

1. Conversion secondes en heures (expressions élémentaires et définitions locales)

Écrire une fonction `secEnHeures : int -> int * int * int` qui prend un nombre de secondes s et le convertit en heures, minutes et secondes. Par exemple, 1000 secondes correspond à 16 minutes et 40 secondes. On utilisera obligatoirement au moins définition locale.

```
#secEnHeures(1000) ;;
- : int * int * int = 0, 16, 40
#secEnHeures(100000) ;;
- : int * int * int = 27, 46, 40
```

2. Opérations booléennes (booléens et filtrage)

Écrire les fonctions booléennes suivantes, obligatoirement par filtrage ou par composition et sans utiliser les fonctions prédéfinies *not*, *or* et & de caml :

1. `non : bool -> bool` qui à un booléen b associe sa négation logique,
2. `et : bool * bool -> bool` qui vaut vrai ssi a et b sont vrais,
3. `ou : bool * bool -> bool` qui vaut faux ssi a et b sont faux,
4. `nand : bool * bool -> bool` qui vaut vrai ssi $et(a,b)$ est faux.

3. Bouge ! (chaînes de caractères et filtrage)

On dispose d'un robot dont la position courante (dans le plan cartésien) est (x,y) où x et y sont des entiers. On souhaite lui donner l'ordre de bouger de 5 unités dans une des 4 directions représentée par une des chaînes de caractère "nord", "sud", "est" ou "ouest". Écrire une fonction `bouge : string * int * int -> int * int` qui au triplet (direction,x,y) associe la nouvelle position (x',y') du robot. On utilisera un filtrage (non exhaustif) des 4 directions autorisées.

```
bouge("ouest",15,45) ;;
- : int * int = 10, 45
```

```
bouge("nord",15,45) ;;
- : int * int = 15, 50
```

4. Puissance de 2 (Récursivité élémentaire sur les entiers)

Écrire une fonction `div2 : int -> int` qui à un entier n associe le nombre de fois que n est divisible par 2 (précisément c'est la puissance de 2 dans la décomposition en facteurs premiers de n).
Par exemple, comme $24 = 2^3 * 3$, que $10 = 2 * 5$ et que 7 n'est pas divisible par 2 :

```
#div2(24) ; ;
- : int = 3
```

```
#div2(10) ; ;
- : int = 1
```

```
#div2(7) ; ;
- : int = 0
```

5. Nombres parfaits (récursivité)

On rappelle que d est un diviseur de n ssi le reste de la division entière de n par d est nul. Bien sûr 1 divise tous les entiers et n divise toujours n. On appelle diviseur strict de n les diviseurs de n contenus entre 1 et $n-1$.
Par exemple les diviseurs stricts de 12 sont 1,2,3,4 et 6. La somme des diviseurs stricts de 12 est donc 1+2+3+4+6=16.

1. Écrire une fonction récursive `somAux : int * int -> int` qui au couple d'entiers (n, p), associe la somme des diviseurs de n compris entre 1 et d. On utilisera obligatoirement un filtrage.

   ```
   #somAux(12,8) ; ;
   - : int = 16
   ```

2. En déduire une fonction `somDiv : int -> int` qui calcule la somme des diviseurs stricts d'un entier n..

   ```
   #somDiv(36) ; ;
   - : int = 55
   #somDiv(28) ; ;
   - : int = 28
   ```

3. Un entier n est dit parfait s'il est égal à la somme de ses diviseurs stricts. D'après les exemples précédents, 28 est un nombre parfait mais pas 36.
   Écrire une fonction booléenne `parfait : int -> bool` qui détermine si un nombre entier n est parfait.

   ```
   parfait(28) ; ;
   - : bool = true
   #parfait(36) ; ;
   - : bool = false
   ```