

Programmation Fonctionnelle : TP1

Université de Tours

Département informatique de Blois

Fonctions, conditions et structures de données

*
* *

Première partie

Dans cette première partie, on s'intéresse au problème classique en algorithmique de la vérification de la validité d'une date donnée. Ce problème est un bon exemple de la puissance du paradigme fonctionnel car il met en jeu différentes fonctions simples opérant ensemble sur un problème plus complexe.

Principe de la décomposition de problèmes

On considère une fonction `dateValide` de paramètres d'entrée : un jour j , un mois m et une année a et correspondant à la spécification suivante :

$$\text{dateValide} : \begin{cases} \text{int} \times \text{int} \times \text{int} & \rightarrow \{true, false\} \\ j, m, a & \mapsto \begin{cases} true & \text{si le format de date } j/m/a \text{ est valide} \\ false & \text{sinon} \end{cases} \end{cases}$$

Par exemple, pour les applications de cette fonction aux arguments suivants, on doit avoir les réponses :

- `dateValide 18 11 1999 :- true`
- `dateValide 18 13 1999 :- false` (Le mois numéro 13 n'existe pas)
- `dateValide 31 11 1999 :- false` (31 n'est pas un jour valide pour le mois de Novembre)
- `dateValide 29 2 1996 :- true`
- `dateValide 29 2 1999 :- false` (Le 29 Février n'existe pas pour l'année 1999)

Description du problème

1. Décrire en français les différentes conditions permettant de vérifier qu'une date au format $j/m/a$ est valide.
2. Écrire la spécification et le code d'une fonction `in_range n n0 n1` qui teste si $n \in \llbracket n_0, n_1 \rrbracket$, c'est-à-dire si un nombre n appartient à l'intervalle de nombres entre n_0 et n_1 . On supposera que $n_0 < n_1$.

Exemple : `in_range 3 (-1) 9 :- true`

`in_range 0 5 7 :- false`

3. Écrire la spécification et le code d'une fonction `nbJoursMois m a` qui retourne le nombre de jours du mois m pour l'année a .

On pourra s'aider de la fonction `bissextile` :

$$\begin{cases} \text{int} & \rightarrow \{true, false\} \\ a & \mapsto \begin{cases} true & \text{si } a \text{ est bissextile} \\ false & \text{sinon} \end{cases} \end{cases}$$

```
let bissextile a = (a mod 400 = 0) || ((a mod 4 = 0) && (a mod 100 <> 0));;
```

Exemple : `nbJoursMois 7 1995 :- 31`

`nbJoursMois 2 1996 :- 29`

`nbJoursMois 2 1999 :- 28`

4. Écrire le code de la fonction `dateValide` en vous aidant des fonctions précédentes.

À l'aide du type `date`

On souhaite implémenter les fonctions précédentes mais à l'aide d'une structure de données définissant une date.

1. Créer un type `date` composé des trois nombres j, m et a de la forme $j/m/a$.
2. Écrire la spécification et le code de la fonction `dateValide_type d` qui retourne vrai si et seulement si la date d possède un format valide. On pourra ré-utiliser la fonction `dateValide` implémenter précédemment.
3. Écrire la spécification et le code d'une fonction `lendemain d` qui retourne la date du lendemain de la date d passée en paramètre. On pourra utiliser les fonction implémenter précédemment.

Exemple : `lendemain {j=31; m=1; a=2019} :- {j = 1; m = 2; a = 2019}`

`lendemain {j=6; m=6; a=2016} :- {j = 7; m = 6; a = 2016}`

`lendemain {j=31; m=12; a=2018} :- {j = 1; m = 1; a = 2019}`

Deuxième partie

Partiel 2019 – En avant la musique !

Le but de cet exercice est de concevoir un programme en Ocaml permettant de représenter des partitions et données musicales très simples.



Johann Pachelbel – *Canon en Ré Majeur* (mes. 17 et 18)

On précise qu'aucune connaissance musicale n'est requise pour la suite de l'exercice.

1. On rappelle que les notes du discours musical peuvent être représentées selon l'ensemble $N = \{Do, Ré, Mi, Fa, Sol, La, Si\}$.

Créer un type `note` permettant de représenter les différentes notes de l'ensemble N .

2. Chaque note possède une fréquence bien déterminée. On note f_0 la fréquence d'une note donnée dans le tableau ci-dessous :

Fréquence f_0 (en Hz)	32.70	36.71	41.20	43.65	49	55	61.74
Note	Do	Ré	Mi	Fa	Sol	La	Si

Écrire la spécification et le code d'une fonction `f_0 n` qui prend en paramètre une note n et retourne sa fréquence f_0 .

3. Au sein du discours musical, une **hauteur** représente un couple constitué d'une **note** n ainsi qu'une octave $k \in \mathbb{N}$ (`int`). Par exemple une hauteur peut-être décrite par le couple $(La, 3)$.

(a) Créer un type **hauteur** permettant de représenter la structure décrite précédemment.





(b) La fréquence f d'une hauteur est donnée par la formule suivante :

$$f \approx f_0 \times 2^k$$

Écrire la spécification et le code d'une fonction `frequence h` qui, selon une hauteur h retourne sa fréquence f .


On pourra utiliser l'opérateur `**` : $\begin{cases} \text{float} \times \text{float} & \rightarrow \text{float} \\ a, b & \mapsto a ** b = a^b \end{cases}$. On donne aussi la commande (`float_of_int k`) qui permet de convertir un entier k en sa version flottante.

4. La **durée** d'une note, déclarée en unité de temps, est également très importante car c'est elle qui est chargée de la représentation temporelle de la musique.

Durée	Nombre d'unités de temps
Croche 	1/2
Noire 	1
Blanche 	2
Ronde 	4

On définit la structure **duree** permettant de représenter les unités de temps telles que décrites dans le tableau ci-dessus.

```
type duree = Croche | Noire | Blanche | Ronde;;
```

Il est également possible de créer des durées de temps fractionnaires plus complexes, notamment au moyen du point de prolongation noté  (ici pour une croche) au sein du discours musical. La durée est alors dite *pointée* et est allongée de sa moitié.

(a) Modifier le type **duree** défini pour tenir compte des durées pointées. On pourra utiliser la notion de type récursif.

(b) Écrire la spécification et le code d'une fonction `unit_temps d` qui prend en paramètre une durée de note d et retourne son nombre d'unités de temps.

5. On définit le type **symbole** suivant permettant de représenter un élément du discours musical :

```
type symbole = Note of hauteur * duree | Silence of duree;;
```

On peut ainsi, par exemple, définir un symbole musical s tel qu'un $(Ré, 5)$ noir :

```
let s = Note ({n=Re; k=5}, Noire);;
```

On définit également les types **tempo** et **partition** tels que :

```
type tempo = {d:duree; nb:int};;
type partition = {symboles : symbole list; tau : tempo};;
```

On souhaite ramener nos durées en unités de temps à une durée en millisecondes. Pour cela, on va utiliser la notion de *tempo*. Le tempo τ représente le nombre de symboles nb d'une durée de note d devant être joués en une minute, c'est-à-dire :

$$60 \text{ sec} = \text{unit_temps}(\tau.d) \times \tau.nb$$

Par exemple, un tempo $\tau = (120, \text{Noire})$ signifie que l'on doit jouer 120 noires par minute, soit une noire toutes les demi-secondes (ou 500 millisecondes).

- (a) Écrire la spécification et le code d'une fonction `ms tau d` qui calcule le temps en millisecondes d'une durée de note d suivant le tempo τ .

Par exemple, pour $\tau = (\text{Blanche}, 120)$ et $d = \text{Noire}$, la fonction `ms` retournera 250.

- (b) Écrire la spécification et le code d'une fonction `temps_melodie p` qui calcule le temps d'une partition en millisecondes.

- (c) Écrire le code permettant de représenter les sept premiers symboles du *Canon en Ré Majeur*, c'est-à-dire, une partition $[s_1; s_2; s_3; s_4; s_5; s_6; s_7]$ telle que :

$s_1 = ((\text{Ré}, 5) \text{ Croche})$; $s_2 = ((\text{Ré}, 4) \text{ Croche})$; $s_3 = ((\text{Do}, 4) \text{ Noire})$; $s_4 = (\text{Silence Croche})$;
 $s_5 = ((\text{Si}, 3) \text{ Croche})$; $s_6 = ((\text{Ré}, 4) \text{ Croche})$; $s_7 = ((\text{Ré}, 4) \text{ Noire pointée})$

avec un tempo $\tau = (\text{Noire}, 68)$.