

Programmation Fonctionnelle : Partiel

Université de Tours

Département informatique de Blois

Partiel 2019 - *Durée : 1h30 (Documents autorisés)*

*
* *

Le code devra être rédigé en Ocaml et comporter les spécifications fonctionnelles nécessaires. Tout code ne compilant pas correctement sera considéré comme invalide.

En avant la musique ! 45min ~ 9 pts : 0.5 + 0.5 + (1+1) + (1+1) + (1+1.5+1.5)

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 noire) au sein du discours musical. La durée est alors dite *pointée* et est allongée de sa moitié.

- Modifier le type **duree** défini pour tenir compte des durées pointées.
 - Écrire la spécification et le code d'une fonction **unit_temps d** qui prend en paramètre une durée *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 définit également les types **tempo** et **partition** tels que :

```
type tempo = {d:duree; k: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 *k* d'une durée *d* devant être joués en une minute, c'est-à-dire :

$$60s = \text{unit_temps}(\tau.d) \times \tau.k$$

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).

- Écrire la spécification et le code d'une fonction **ms tau d** qui calcule le temps en millisecondes d'une durée *d* suivant le tempo τ .
 Par exemple, pour $\tau = (\text{Blanche}, 120)$ et *d* = Noire, *ms* retournera 250.
- Écrire la spécification et le code d'une fonction **temps_melodie p** qui calcule le temps d'une partition en millisecondes.
- Écrire le code permettant de représenter les sept premiers symboles du *Canon en Ré Majeur*, c'est-à-dire, une partition $[s_1; s_2; s_2; s_3; s_4; s_5; s_6; s_7]$ 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 $\tau = (\text{Noire}, 68)$.

Étude de complexité : Le tri à bulles 45min ~ 11 pts : (0.5+1) + (0.5+1+1) + 1 + (1+1.5) + (2+1.5)

On s'intéresse ici à l'étude, d'un point de vue empirique, de la complexité d'une nouvelle méthode de tri d'une liste, le *tri à bulles*.

On précise ici que, pour simplifier la modélisation, on considère qu'une liste l de n éléments est représentée telle que :

$$l = [x_0; x_1; \dots; x_{n-1}]$$

1. Pour commencer, on va générer des grandes listes.
 - (a) Écrire le code d'une fonction `grandeListeDecroiss n` qui, pour un entier n , crée une liste l décroissante de la forme $l = [n; n-1; \dots; 3; 2; 1]$.
 - (b) Écrire le code d'une fonction `grandeListeCroiss n` qui, pour un entier n , crée une liste l croissante de la forme $l = [1; 2; 3; \dots; n-1; n]$.
2. La permutation d'éléments est une opération essentielle pour le tri des listes. Les prochaines questions visent à implémenter cette opération :
 - (a) Écrire la spécification et le code d'une fonction `get i l` qui, étant donné une liste l retourne l'élément x_i .
 - (b) Écrire la spécification et le code d'une fonction `replace e i l` qui, étant donné une liste l remplace l'élément x_i par l'élément e .
 - (c) Écrire la spécification et le code d'une fonction `permute i j l` qui, étant donné une liste l d'éléments, permute les éléments x_i et x_j de la liste.
Exemple : L'appel `permute [3;4;5;2;8;0] 2 4` retournera la liste `[3;4;8;2;5;0]`.
3. On propose la fonction suivante `grandeListeAlea n` qui génère une liste de taille n d'éléments, sans doublon, et mélangée :

```
let grandeListeAlea n =
  let rec melange l =
    let l_melange = permute l 0 (Random.int (List.length l)) in
    match l_melange with
    | [] -> []
    | h::t -> h::(melange t) in
  melange (grandeListeDecroiss n);;
```

À votre avis, et sans le démontrer, quelle est la complexité de la fonction `grandeListeAlea n`? Expliquez l'intuition de votre réponse.

On rappelle les différentes classes de complexité et leur relation d'inclusion : $O(\log(n)) \subseteq O(n) \subseteq O(n \log(n)) \subseteq O(n^2) \subseteq O(a^n) \subseteq O(n!)$.

4. On rappelle les formules de la moyenne μ_l et l'écart-type σ_l des éléments d'une liste de flottants l :

$$\mu_l = \frac{1}{n} \sum_{i=0}^{n-1} x_i \quad \text{et} \quad \sigma_l = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (x_i - \mu_l)^2}$$

On accordera un bonus de 2 pts dans le cas où le code des fonctions suivantes est écrit selon la version *récursive terminale*.

- (a) Écrire le code d'une fonction `mean l` qui calcule la moyenne μ_l d'une liste l de flottants.
- (b) Écrire le code d'une fonction `sd l` qui calcule l'écart-type σ_l d'une liste l de flottants. On pourra utiliser la fonction `sqrt` :
- $$\text{sqrt} : \begin{cases} \text{float} & \rightarrow \text{float} \\ a & \mapsto \sqrt{a} \end{cases}.$$

5. On considère le code suivant qui réalise un tri à bulles sur une liste d'entrée l :

```
let rec tri_a_bulles l =
  let rec _tri_a_bulles = function
    | h :: h2 :: t when h > h2 -> begin
      match _tri_a_bulles (h :: t) with
      | [] -> h2 :: h :: t
      | t2 -> h2 :: t2
    end
    | h :: h2 :: t -> begin
      match _tri_a_bulles (h2 :: t) with
      | [] -> []
      | t2 -> h :: t2
    end
    | _ -> []
  in
  match _tri_a_bulles l with
  | [] -> l
  | l -> tri_a_bulles l;;
```

- (a) On propose le code suivant afin de mesurer le temps d'exécution du tri à bulles sur une liste l placée en paramètre, pour se faire, on déclare :

$$\text{chronometre} : \begin{cases} \text{List} < a' > & \mapsto \text{float} \\ l & \mapsto \text{Temps d'exécution du tri à bulles pour } l \end{cases}$$

```
let chronometre l =
  let deb = Sys.time() in
  let l_triee = tri_a_bulles l in
  (Sys.time() -. deb);;
```

On considère également la fonction `mesures` permettant d'effectuer k mesures du temps d'exécution du tri à bulles sur l :

$$\text{mesures} : \begin{cases} \text{int} \times \text{List} < a' > & \mapsto \text{float} \times \text{float} \\ k, l & \mapsto (\mu, \sigma) \end{cases}$$

Où μ et σ sont respectivement le temps moyen et l'écart-type d'exécution du tri à bulles de la liste l pour une liste de k mesures.

```
let mesures k l =
  let rec make_mesures k l = match (k,l) with
    (0, l) -> []
    | (k, l) -> (chronometre l)::(make_mesures (k-1) l)
  in
```

```
(moyenne (make_mesures k 1), sd(make_mesures k 1));;
```

Remplissez le tableau suivant du temps de calcul du tri à bulles pour une liste de taille n à l'aide de la fonction `mesures k 1`. On fixera ici $k = 3$.

Si la fonction `sd` ou `mean` n'a pas été réalisée, on utilisera la fonction `chronometre`. Sinon, on utilisera la fonction `mesures`.

(Les résultats seront donnés à 10^{-3} près).

l		Taille n				
		1000	5000	10.000	15.000	20.000
<code>grandeListeDecroiss</code>	Temps moyen μ					
	Écart-type σ					
<code>grandeListeCroiss</code>	Temps moyen μ					
	Écart-type σ					
<code>grandeListeAlea</code>	Temps moyen μ					
	Écart-type σ					

- (b) À votre avis, et sans le démontrer, quelle est la complexité de calcul du tri à bulles lorsque la liste est décroissante (pire des cas)? Lorsque la liste est croissante (meilleur des cas)? Lorsque la liste est aléatoire? Expliquez l'intuition de votre réponse.

Travail complémentaire facultatif à rendre le 15 Mars (23h55) $\sim (2+1)$ pts

- Sous un outil de tableur Excel, tracer les courbes représentant le temps d'exécution $t = f(n)$ du tri en fonction de la taille n de la liste. On utilisera :
 - Une couleur par type de liste (croissante, décroissante, aléatoire),
 - Différents styles de trait (continu, pointillé...) pour distinguer la taille des listes.
- Quelles conclusions en tirez-vous?