Institut National Universitaire Champollion

Contrôle continu n°1

L3 Info
8 décembre 2020

PROGRAMMATION FONCTIONNELLE CAML

Durée : 2h Aucun document autorisé

1. Listes : Gloire aux deuxièmes ! À la recherche du deuxième plus grand élément d'une liste

On souhaite écrire des fonctions calculant le plus grand élément d'une liste puis le deuxième plus grand élément d'une liste.

1. Écrire une fonction `maxi : 'a list -> 'a` qui retourne le plus grand élément de la liste. On pourra écrire une fonction de calcul du maximum de deux entiers ou utiliser la fonctionnelle prédéfinie `max`. On lèvera l'exception `ListeVide` si la liste est vide.

```
maxi [3 ;2 ;5 ;2 ;7 ;1 ;9 ;4 ;3 ;2 ;5] ; ;
-  : int = 9
```

2. Écrire une fonction `elimine : 'a * 'a list -> 'a list` qui élimine la première occurrence d'un élément dans une liste, sachant qu'il est présent dans la liste. Utiliser si possible un filtrage gardé (when...) pour écrire cette fonction.

```
elimine(9, [3 ;2 ;5 ;2 ;7 ;1 ;9 ;4 ;3 ;2 ;5]) ; ;
-  : int list = [3 ; 2 ; 5 ; 2 ; 7 ; 1 ; 4 ; 3 ; 2 ; 5]
```

3. En composant les fonctions précédentes, écrire une fonction `deuxieme : 'a list -> 'a` qui retourne le deuxième plus grand élément de la liste. On lèvera l'exception `TropCourte` si la liste est tropCourte.

```
deuxieme [3 ;2 ;5 ;2 ;7 ;1 ;9 ;4 ;3 ;2 ;5] ; ;
-  : int = 7
```

2. Listes : deux fonctions récursives

1. Écrire une fonction `applique : ('a -> 'b) * 'a list -> 'b list` qui applique une fonction à tous les éléments d'une liste.

```
applique ((fun n->n*n),[1 ;2 ;3 ;4 ;5]) ; ;
-  : int list = [1 ; 4 ; 9 ; 16 ; 25]
```

2. Écrire une fonction `applatir : 'a list list -> 'a list` qui calcule la liste de tous les éléments d'une liste de listes. On comprends mieux sur un exemple...

```
applatir ([[1 ;3] ;[] ;[2] ;[6 ;4 ;2] ;[1 ;5]]) ; ;
-  : int list = [1 ; 3 ; 2 ; 6 ; 4 ; 2 ; 1 ; 5]
```

3. types sommes et produits : gestion du personnel de l'entreprise

Le but de cet exercice est d'écrire des fonctions permettant la gestion des informations familiales des membres du personnel d'une entreprise (pour mieux préparer le sapin de Noël). L'entreprise distingue donc ses employés célibataires et ceux qui ont une famille. Une famille est caractérisée par son nom (une chaîne de caractère), la liste des âges du membre du personnel et de son éventuel conjoint (une liste d'un ou deux entiers) et enfin la liste des âges de ses enfants (une liste d'entiers, éventuellement vide). On peut donc la représenter en Caml par le type produit suivant :

```
type famille = {Nom : string ; AgesParents : int list ; AgesEnfants : int list} ; ;
```

Un célibataire est simplement défini par un couple constitué de son nom et de son âge. Un foeyr peut alors être représenté par le type somme :

```
type foyer = Celib of string*int
           | Famille of famille ; ;
```

Le personnel de l'entreprise est enfin représenté par une liste de foyers : Nous considérerons par exemple l'entreprise suivante :

```
let enterprise = [Famille {Nom = "Dupont" ; AgesParents = [22 ; 25] ; AgesEnfants = [3]} ;
Celib ("Durand",18) ; Famille {Nom="Vador" ;AgesParents=[54] ;AgesEnfants=[26]} ;
Famille {Nom = "Martin" ; AgesParents = [32 ; 30] ; AgesEnfants = [2 ; 4 ; 6]}] ; ;
```

Écrire les fonctions suivantes, permettant la gestion des membres du personnel :

1. `long` calculant la longueur d'une liste.
2. `nbIndiv : foyer -> int` qui donne le nombre de personnes dans un foyer.

```
nbIndiv(Celib ("Durand",18)) ; ;
-  : int = 1
nbIndiv(Famille {Nom = "Martin" ; AgesParents = [32 ; 30] ; AgesEnfants = [2 ; 4 ; 6]}
-  : int = 5
```

3. `nbCelib : foyer list -> int` calculant le nombre de célibataires parmi les membres du personnel.

```
#nbCelib(enterprise) ; ;
-  : int = 1
```

4. `nbEnfants : foyer list -> int` permettant de compter le nombre total d'enfants des membres du personnel

```
#nbEnfants(enterprise) ; ;
- : int = 5
```

5. `naissance : string * foyer list -> foyer list` permettant la mise à jour de la liste du personnel lors de la naissance d'un enfant dans une famille de nom donné. On lèvera l'exception pasTrouvé si l'employé n'est pas dans l'entreprise.

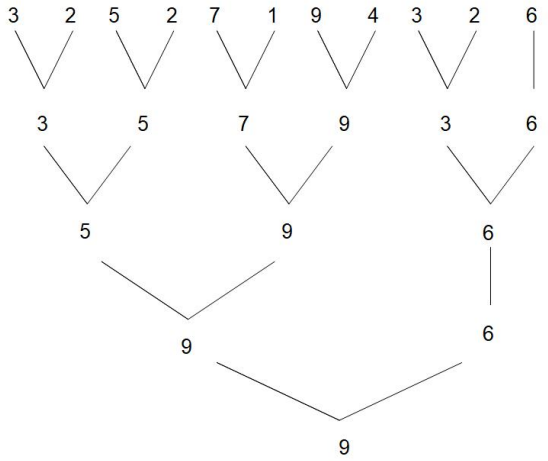
```
naissance ("Martin",enterprise) ; ;
- : foyer list = [Famille {Nom = "Dupont" ; AgesParents = [22 ; 25] ;
AgesEnfants = [3]} ; Celib ("Durand", 18) ; Famille {Nom = "Vador" ;
AgesParents = [54] ; AgesEnfants = [26]} ; Famille {Nom = "Martin" ;
AgesParents = [32 ; 30] ; AgesEnfants = [0 ; 2 ; 4 ; 6]}]

naissance ("Durand",enterprise) ; ;
- : foyer list =
 [Famille {Nom = "Dupont" ; AgesParents = [22 ; 25] ; AgesEnfants = [3]} ;
  Famille {Nom = "Durand" ; AgesParents = [18] ; AgesEnfants = [0]} ;
  Famille {Nom = "Vador" ; AgesParents = [54] ; AgesEnfants = [26]} ;
  Famille {Nom = "Martin" ; AgesParents = [32 ; 30] ; AgesEnfants = [2 ; 4 ; 6]}]
```

6. Tous les ans, les enfants et leurs parents prennent un an de plus ! Écrire une fonction `miseAJour : foyer list -> foyer list`, ajoutant 1 an à l'âge de tous les enfants et tous les parents. (On pourra commencer par écrire une fonction auxiliaire ajoutant 1 à tous les éléments d'une liste d'entiers)

```
#miseAjour(enterprise) ; ;
- : foyer list =
 [Famille {Nom = "Dupont" ; AgesParents = [23 ; 26] ; AgesEnfants = [4]} ;
  Celib ("Durand", 19) ;
  Famille {Nom = "Vador" ; AgesParents = [55] ; AgesEnfants = [27]} ;
  Famille {Nom = "Martin" ; AgesParents = [33 ; 31] ; AgesEnfants = [3 ; 5 ; 7]}]
```

4. Duels et tournois : un type somme récursif

Pour calculer le maximum d'une liste, on peut utiliser une méthode calquée sur le principe des tournois par duels successifs. Par exemple, on peut calculer le maximum (9) de la liste `[3 ;2 ;5 ;2 ;7 ;1 ;9 ;4 ;3 ;2 ;5]` à l'aide du tournoi suivant :



Le but de cet exercice est de transformer une liste d'entiers en un tournoi, qui n'est autre qu'un arbre binaire.
On définit le type somme récursif :

```
type tournoi =   joueur of int
               | partie of int*tournoi*tournoi ; ;
```

Dans le cas récursif, une partie est constituée de deux sous-tournois et du numéro du vainqueur

1. Écrire une fonction `initialise : int list -> tournoi list` qui transforme une liste d'entiers en liste de tournois élémentaires réduits aux joueurs

   ```
   let init=initialise([3 ;2 ;5 ;2 ;7 ;1 ;9 ;4 ;3 ;2 ;6]) ; ;
   init  : tournoi list =
    [joueur 3 ; joueur 2 ; joueur 5 ; joueur 2 ; joueur 7 ; joueur 1 ; joueur 9 ;
     joueur 4 ; joueur 3 ; joueur 2 ; joueur 6]
   ```

2. Écrire une fonction `vainqueur : tournoi -> int` qui à un tournoi associe son vainqueur.

   ```
   vainqueur (joueur 5) ; ;
   -  : int = 5
   ```

   ```
   vainqueur (partie (3, joueur 3, joueur 2)) ; ;
   -  : int = 3
   ```

3. Écrire une fonction `jeu : tournoi * tournoi -> tournoi` qui organise une partie entre les vainqueurs de deux tournois.

   ```
   jeu (joueur 4,partie (5, joueur 5, joueur 2)) ; ;
   -  : tournoi = partie (5, joueur 4, partie (5, joueur 5, joueur 2))
   ```

4. Écrire une fonction `ronde : tournoi list -> tournoi list` qui organise des parties 2 par 2 entre les éléments d'une liste de tournois.

```
let ronde1=ronde(init) ; ;
ronde1  : tournoi list =
 [partie (3, joueur 3, joueur 2) ; partie (5, joueur 5, joueur 2) ;
  partie (7, joueur 7, joueur 1) ; partie (9, joueur 9, joueur 4) ;
  partie (3, joueur 3, joueur 2) ; joueur 6]

let ronde2=ronde(ronde1) ; ;
ronde2  : tournoi list =
 [partie (5, partie (3, joueur 3, joueur 2), partie (5, joueur 5, joueur 2)) ;
  partie (9, partie (7, joueur 7, joueur 1), partie (9, joueur 9, joueur 4)) ;
  partie (6, partie (3, joueur 3, joueur 2), joueur 6)]
```

5. Écrire une fonction `leTournoi  : tournoi list -> tournoi` qui enchaîne les rondes jusqu'à ce qu'il n'y ait plus qu'un seul vainqueur.

```
leTournoi(init) ; ;
-  : tournoi =
 partie  (9,   partie    (9,     partie      (5, partie (3, joueur 3, joueur 2), par
 partie      (9, partie (7, joueur 7, joueur 1), partie (9, joueur 9, joueur 4))),
 partie (6, partie (3, joueur 3, joueur 2), joueur 6))
```

6. Écrire enfin une fonction `leChampion  : int list -> int`, utilisant les fonctions précédentes qui initialise, met en oeuvre le tournoi et retourne son vainqueur.

```
leChampion([3 ;2 ;5 ;2 ;7 ;1 ;9 ;4 ;3 ;2 ;6]) ; ;
-  : int = 9
```