Institut National
Universitaire
Champollion

L3 Info
19 Octobre 2021

PROGRAMMATION FONCTIONNELLE CAML

Durée : 2h Aucun document autorisé

Toutes vos fonctions devront être accompagnées de leur type et d'un jeu de tests représentatifs.

1. Sur les listes : 5 points

1. Écrire une fonction `nbOcc : 'a * 'a list -> int` qui compte le nombre d'occurrences (= apparitions) d'un élément x dans une liste ℓ.

   ```
   nbOcc : 'a * 'a list -> int
   ```

2. Écrire une fonction récursive `repet : int * 'a -> 'a list` qui à un entier n et un objet a de type quelconque associe la liste constituée de n répétitions de a. On pourra supposer sans vérification que n est positif.

   ```
   repet(7,3) ; ;
   - : int list = [3 ; 3 ; 3 ; 3 ; 3 ; 3 ; 3]

   repet(7,'a') ; ;
   - : char list = ['a' ; 'a' ; 'a' ; 'a' ; 'a' ; 'a' ; 'a']
   ```

3. Écrire une fonction récursive `nPremiers : int * 'a list -> 'a list` qui à un entier n et une liste ℓ, associe la liste constituée des n premiers éléments de ℓ.

   ```
   nPremiers(4,[2 ;3 ;4 ;5 ;6 ;7 ;8]) ; ;
   - : int list = [2 ; 3 ; 4 ; 5]

   nPremiers(4,[2 ;3 ;4]) ; ;
   Exception non rattrapée : Failure "la liste est trop courte"
   ```

4. En déduire une fonction récursive `tranche : int * int * 'a list -> 'a list` qui à deux entiers n et m et une liste ℓ, associe la liste constituée des éléments de ℓ dont l'indice est compris entre n et m (on indexe les éléments de ℓ à partir de 1).

   ```
   tranche(2,4,[1 ;2 ;3 ;4 ;5 ;6 ;7]) ; ;
   - : int list = [2 ; 3 ; 4]

   tranche(2,6,[1 ;2 ;3 ;4 ;5 ;6 ;7]) ; ;
   - : int list = [2 ; 3 ; 4 ; 5 ; 6]

   tranche(2,9,[1 ;2 ;3 ;4 ;5 ;6 ;7]) ; ;
   Exception non rattrapée : Failure "la liste est trop courte"
   ```

2. Dans la cour de récréation : 3 points

Nous définissons le type `chifoumi` de la façon suivante :

```
type chifoumi = Pierre | Feuille | Ciseaux ;;
```

1. Ecrire une fonction `quiGagne : chifoumi -> chifoumi` qui pour chaque coup (Pierre, Feuille ou Ciseaux) renvoie le coup gagnant.

```
quiGagne(Pierre) ;;
-  : chifoumi = Feuille

quiGagne(Feuille) ;;
-  : chifoumi = Ciseaux

quiGagne (Ciseaux) ;;
-  : chifoumi = Pierre
```

2. En déduire une fonction `duel : chifoumi * chifoumi -> chifoumi` qui renvoie le coup gagnant d'un duel de Chifoumi.

```
duel (Feuille, Ciseaux) ;;
(*-  : chifoumi = Ciseaux*)

duel (Feuille, Feuille) ;;
(*#Exception non rattrapée : Failure "Egalite !"*)
```

3. Arbre binaire de recherche : 12 points

Nous nous intéressons à des opérations de recherche et de tri sur des arbres binaires dits de recherche, d'entiers du type suivant :

```
type arbre_binaire =
Vide
| Noeud of int * arbre_binaire * arbre_binaire ;;
```

dont les éléments sont des entiers et dont les nœuds vérifient la propriété suivante : **pour chaque nœud, les éléments du sous-arbre gauche sont inférieurs ou égaux à l'entier du nœud et les éléments du sous-arbre droit sont supérieurs ou égaux à l'entier du nœud.**

Nous testerons nos fonctions sur les deux exemples suivants, qui sont des arbres valides (on remarquera que le second possède 2 fois l'entier 5) :

```
let abr1 = Noeud( 8,
Noeud (3, Noeud(2,Vide,Vide), Noeud(6,Vide, Vide)),
Noeud (19, Vide, Vide)) ;;

let abr2 = Noeud( 5,
Noeud (3, Noeud(2,Vide,Vide), Noeud(5,Vide, Vide)),
Noeud (7, Vide, Noeud(8,Vide,Vide))) ;;
```

Première fonction sur les arbres

1. Écrire une fonction `taille : arbre_binaire -> int` qui renvoie le nombre d'entiers présents dans un arbre.

```
taille abr1 ;;
- : int = 5

taille abr2 ;;
- : bool = 6
```

Recherche dans un arbre binaire de recherche

2. Écrire une fonction `recherche : int * arbre_binaire -> bool` qui teste si un entier appartient à un arbre binaire.

```
recherche (6, abr1) ;;
- : bool = true
recherche (12, abr2) ;;
- : bool = false
```

Tri d'une liste grâce à un arbre binaire de recherche

3. Écrire une fonction `insertion : int* arbre_binaire -> arbre_binaire` qui ajoute un entier à un arbre binaire en respectant la propriété de l'arbre binaire.

   Rappel Structures de Données : Si on souhaite ajouter l'entier x, on parcourt l'arbre comme si on cherchait l'entier x et lorsque l'on arrive sur un nœud qui n'a pas de fils gauche et dont l'entier est supérieur ou égal à x, on place x en fils gauche. De la même façon, lorsque l'on arrive sur un nœud qui n'a pas de fils droit et dont l'entier est inférieur ou égal à x, on place x en fils droit.

```
insertion 4 abr1 ;;
- : arbre_binaire =
Noeud( 8,
Noeud (3, Noeud(2,Vide,Vide), Noeud(6,Noeud(4,Vide, Vide, Vide)),
Noeud (19, Vide, Vide))

insertion 3 abr2 ;;
- : arbre_binaire =
Noeud( 5,
Noeud (3, Noeud(2,Vide,Noeud(3,Vide,Vide)), Noeud(5,Vide, Vide)),
Noeud (7, Vide, Noeud(8,Vide,Vide)))
```

4. En déduire une fonction `list_to_arbre : int list -> arbre_binaire` qui prend une liste d'entiers et renvoie l'arbre binaire de recherche associé.

```
list_to_arbre [8 ;3 ;2 ;19 ;6] ; ;
- : arbre_binaire =
Noeud (8,
Noeud (3, Noeud (2, Vide, Vide), Noeud (6, Vide, Vide)),
Noeud (19, Vide, Vide))
```

5. Écrire une fonction `parcours_infixe : arbre_binaire -> int list` qui convertit un arbre en liste, en faisant un parcours infixe : d'abord le sous-arbre gauche, ensuite la racine puis le sous-arbre droit.

```
parcours_infixe abr1 ;;
- : int list = [2 ; 3 ; 6 ; 8 ; 19]

parcours_infixe abr2 ;;
- : int list = [2 ; 3 ; 5 ; 5 ; 7 ; 8]
```

6. En déduire une fonction `tri : int list -> int list` permettant de trier une liste grâce à un arbre binaire de recherche.

```
tri [8 ;3 ;2 ;19 ;6] ;;
- : int list = [2 ; 3 ; 6 ; 8 ; 19]
```

Questions Superman

Pour ces deux dernières questions, vous pouvez écrire une ou plusieurs fonction(s) auxiliaire(s).

7. Écrire une fonction `a_doublons : arbre_binaire -> bool` qui détermine si un arbre binaire de recherche a des doublons.

```
a_doublons abr1 ;;
- : bool = false
a_doublons abr2 ;;
- : bool = false
```

8. Écrire une fonction `est_trie : arbre_binaire -> bool` qui vérifie si un arbre binaire est bien trié.

```
est_trie abr1 ;;
- : bool = true
```