NFP 119 : Programmation Fonctionnelle

Première session – Février 2018

Durée : 3h

Il vous est recommandé de rédiger de façon concise et précise, tout en écrivant lisiblement. Il sera tenu compte de la présentation dans la note de la copie. Tous les documents sont autorisés. Les calculatrice, ordinateur et téléphone portable sont interdits.

Exercice 1

Complétez le tableau suivant :

Motif	Valeur comparée	Réussite	Liaisons
'a'	'b'		
a	'b'		
(a,b)	(1,2)		
x : : _	[1 ; 2 ; 3]		
	[[1 ; 2] ; [3]]	oui	x = [1 ; 2]
_ : : x : : y	[1 ; 2 ; 3 ; 4]		
(x, y, _)	(3, 4)		
	('a', "abc", true)		str = 'abc'
{id = n}	{id = 1 ; alias = "bob"}		
	{id = 1 ; alias = "bob"}	oui	i = 1, a = "bob"

Lorsqu'il s'agit de compléter la colonne « **Motif** » plusieurs réponses sont en général possibles.

Exercice 2

On cherche a créer un inventaire d'équipement informatique pour une entreprise. Un équipement est définit par un genre (ordinateur de bureau, portable, tablette, switch, routeur, ...), un nom de modèle, une marque, une date de mise en service, ainsi qu'un numéro de série.

Considérez la définition de type suivante :

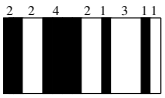
```
type equip = { genre: string; modele: string; marque: string;
               dateMiseService: int*int*int; numSerie: int }
```

1. Écrire, en utilisant le filtrage, une fonction qui permet de retourner les équipements d'une marque donnée ;
2. Écrire une fonction qui trie par dates croissantes de mise en services une liste d'équipements ;
3. Écrire une fonction qui retourne la liste des équipements mises en services avant une date donnée `t`.

Exercice 3

On veut écrire un programme permettant de manipuler des codes barres. Un code barre est une suite de ligne verticales noires ou blanches plus ou moins épaisses. Par exemple :

Les largeurs possibles des barres sont des multiples de la largeur minimale. Dans l'exemple les 2 dernières lignes noires sont de largeur minimale l, la première est de largeur $2 \times l$ et la deuxième de largeur $4 \times l$. Il y a également des barres blanches de largeur l, $2 \times l$ et $3 \times l$, comme illusté ici :



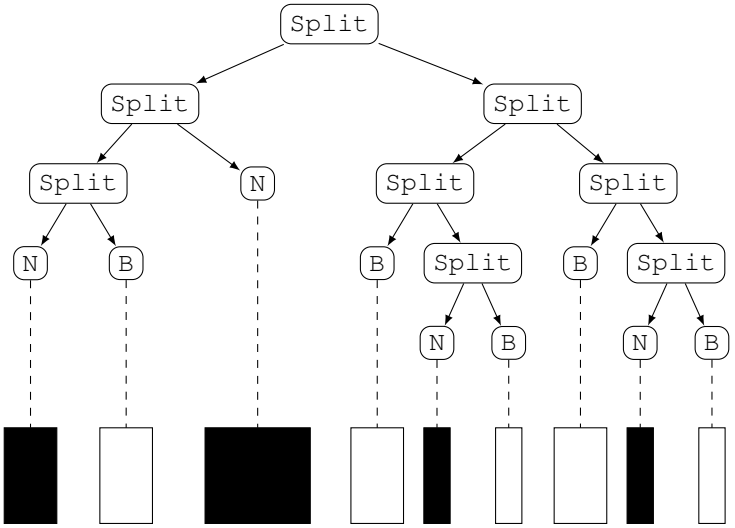
On représente les codes barres à l'aide du type suivant :

```
(* Des arbres qui representent des codes barres *)
type codeBarre =
  | B (* Barre Blanche *)
  | N (* Barre Noire *)
  | Split of codeBarre*codeBarre
```

Un code barre est soit une barre noire (`N`), soit une barre blanche (`B`) soit la concaténation de 2 codes barres (`Split(c1,c2)` ce constructeur est récursif). Le code barre ci-dessus pourra être représenté de la façon suivante :

```
Split(
  Split(Split(N,B),N),
  Split(Split(B,
              Split(N,B)),
        Split(B,
              Split(N,B))))
```

Comme le montre la figure ci-dessous. Notez que les `N` et `B` représentent des barres de largeurs différentes en fonction de leur profondeur dans l'arbre. Notez également qu'il arrive que des barres de même couleur soient voisines.



1. Dessinez l'arbre et le code barre (comme dans la figure ci-dessus) correspondant aux expressions ci-dessous :

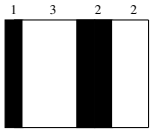
   (a)

```
Split(
  Split(Split(N,N), B),
  Split(
    Split(B, N),
    Split(Split(N,B), B)
  )
)
```

   (b)

```
Split(
  Split(
    Split(B, Split(N,B)),
    B
  ),
  B
)
```

2. Écrivez l'expression de type `codeBarre` correspondant au code barre suivant :



3. Écrivez la fonction `prof: codeBarre -> int` qui retourne la profondeur d'un arbre de type `codeBarre`.

4. Écrivez la fonction `to_string_h: codeBarre -> int -> string` telle que `to_string_h c n` retourne la chaîne de caractères représentant un code barre `c` dont la profondeur est `n`. `n` permet de savoir la largeur des `N` et `B` à la profondeur actuelle.

   On représentera une barre blanche par une succession de caractère espace `"␣"` et les barre noire par une succession de caractères `"x"`.

   Par exemple, `to_string_h (Split(B,Split(N,B)))` retournera la chaîne `"␣x␣"` composée de 2 espaces, un x et un espace. De même le code barre de la question 2 ci-dessus s'affichera comme suit : `"x␣xx␣"`.

   Note : la fonction standard `String.make` pourra être utilisée pour créer des chaînes de caractères : `String.make n c` retourne une chaîne de caractères composée de `n` copies du caractère `c`. Par exemple `String.make 'x' 4` retourne `"xxxx"`.

5. En utilisant les deux fonctions `prof` et `to_string_h` écrivez la fonction

   ```
   to_string: codeBarre -> string
   ```

   qui retourne la chaîne de caractères représentant un code barre.

6. Un QRCode est une version 2D des codes barres. On place des carré blancs ou noire dans le plan au lieu de placer des barre sur une droite. Proposez une nouveau type ocaml pour représenter les QRCode de façon similaire au type `codeBarre`. Indice : un carré se divise naturellement en 4 carrés plus petits.