Licence STS mention Informatique, semestre 4 — 2016–2017
Introduction à la programmation fonctionnelle — Durée : 2h
3 pages

Examen du 12 mai 2017

Les notes et les transparents de cours sont les seuls documents autorisés.

Question 1 Les fonctions suivantes sont-elles bien typées ? Si oui, donner leur type, sinon préciser pourquoi.

```
1) let f1 x f = if x > 0 then f x else 5.2
2) let rec f2 a b = if a < 0.5 then a else (f2 b a) + 10
3) let f3 x y = if x < y then y :: [x] else []
```

Question 2 Donner les valeurs de `v1` et `v2` calculées de la manière suivante :

```
let rec f g l =
 match l with
 | [] -> 0
 | y :: s  -> (g y) + (f g s)

let v1 = f (fun x -> x + 2) [1;2;3]
let v2 = f (fun x -> if x = 0 then 1 else 0) [1;0;0;0;1]
```

Transformations géométriques

Le but de cet exercice est de réaliser un programme pour afficher et effectuer des transformations géométriques d'un ensemble de segments. Les transformations utilisées seront les rotations et les translations. Pour dessiner ces segments, on utilisera les fonctions de la bibliothèque `Graphics` d'OCaml (comme vu en TP). Les types des fonctions nécessaires à notre programme sont rappelés à la fin du sujet.

Pour représenter ces ensembles de segments et ces transformations, nous définissons les types OCaml suivants :

```
type point =  { px : int ; py : int ; poids : int }
type segment = { p1 : point; p2 : point }

type transformation =
  | Rotation of point * float
  | Translation of int * int

type seglist = segment list
```

Ainsi, on défnit tout d'abord un type `point` pour représenter des points pondérés du plan. Chaque point est donc un enregistrement avec trois champs : `px` et `py` contiennent les coordonnées du point et `poids` contient la pondération du point. Un segment, de type `segment`, est défini par deux points qui sont stockés dans un enregistrement avec deux champs `p1` et `p2`. Les transformations géométriques sont définies par un type algébrique `transformation` avec deux constructeurs. Les rotations sont de la forme `Rotation(p,a)` où `p` est le point (ou centre) de rotation et `a` est l'angle de rotation (en radian). Les translations sont de la forme `Translation(x,y)` où `x` et `y` contiennent les coordonnées du vecteur de translation. Enfin, le type `seglist` représente des listes de segments.

Question 3 Définir quatre variables `p1`, `p2`, `p3` et `p4` pour représenter les points de coordonnées et poids suivants :

- $(10, 10)$ de poids 5
- $(50, 10)$ de poids 3
- $(50, 20)$ de poids 20
- $(50, 60)$ de poids 10

Question 4 En utilisant les variables définies dans la question précédente, définir une variable `ls` de type `seglist` qui contient trois segments :

- un segment entre `p1` et `p2`
- un segment entre `p2` et `p3`
- un segment entre `p3` et `p4`

On définit le poids d'un segment `p` comme la somme des poids de `p.p1` et `p.p2`.

Question 5 Écrire une fonction `poids_segment`, de type `segment -> int`, qui renvoie le poids d'un segment.

On va maintenant écrire une fonction pour dessiner un segment en couleur. La couleur d'un segment sera reliée à son poids. Ainsi, la couleur d'un segment `p` sera :

- bleu, si le poids de `p` est strictement inférieur à 10 ;
- rouge, si le poids de `p` est supérieur ou égal à 10 et strictement inférieur à 30 ;
- vert, si le poids de `p` est supérieur ou égal à 30.

Question 6 Écrire une fonction `poids_segments l`, de type `seglist -> int`, qui calcule et renvoie la somme des poids des segments de la liste `l`.

Question 7 Écrire une fonction `poids_vers_color`, de type `int-> Graphics.color`, telle que `poids_vers_color x` renvoie la couleur associée au poids `x`.

Question 8 En utilisant les fonctions de la bibliothèque `Graphics`, écrire une fonction `affiche_segments`, de type `seglist -> unit` tel que `affiche_segments l` affiche à l'écran, avec la bonne couleur, tous les segments de la liste `l`.

On va maintenant programmer les transformations géométriques sur les segments.

Étant donné un point de coordonnées (x, y), on calcule les coordonnées (x', y') du point résultant de la rotation de centre (c_x, c_y) et d'angle α par les équations suivantes :

$$\begin{aligned} x' &= x \cdot \cos(\alpha) - y \cdot \sin(\alpha) + c_x \\ y' &= x \cdot \sin(\alpha) + y \cdot \cos(\alpha) + c_y \end{aligned}$$

Question 9 Écrire une fonction `rotation_point`, de type `point -> float -> point -> point`, telle que `rotation_point c a p` renvoie le point résultant de la rotation de centre `c` et d'angle `a` (supposé donné en radian).

Question 10 En utilisant la question précédente, écrire une fonction `rotation`, de type `point -> float -> seglist -> seglist`, telle que `rotation c a l` applique une rotation de centre `c` et d'angle `a` à tous les segments de `l`.

Question 11 Écrire une fonction `translation_point`, de type `point -> int -> int -> point`, telle que `translation_point p x y` renvoie le point résultant de la translation de `p` par le vecteur de coordonnées `(x,y)`.

Question 12 En utilisant la question précédente, écrire une fonction `translation`, de type `int -> int -> seglist -> seglist`, telle que `translation x y l` applique la translation de vecteur `(x,y)` à tous les segments de `l`.

Question 13 En utilisant les questions précédentes, écrire une fonction `appliquer_transformation`, de type `seglist -> transformation list -> seglist`, telle que `appliquer_transformation ls lt` applique successivement toutes les transformations de `lt` à tous les segments de `ls`.

Rappels

`set_color : color -> unit`

`set_color c` passe la couleur de dessin à la valeur `c`. Les valeurs prédéfinies de type `color` sont : `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`

`moveto : int -> int -> unit`

`moveto x y` positionne le point courant en `(x, y)`

`lineto : int -> int -> unit`

`lineto x y` dessine un segment entre le point courant et le point de coordonnées `(x, y)` et fixe le point courant en `(x, y)`.