Licence STS mention Informatique, semestre 4 — 2015–2016
Introduction à la programmation fonctionnelle — Durée : 2h
2 pages

Examen du 13 mai 2016

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 y z = if x > y then z x else z y
2)  let f2 a b = a :: (b a)
3)  let rec f3 x y = if x < 10.5 then 0 else f3 (y-1) x
```

Question 2 Donner le type de la fonction suivante et la valeur renvoyée par l'appel `f [3; 2; 1] [1; 2]`

```
let rec f x y =
  match x with
  | [] -> y
  | _ :: r -> f y r
```

Dessiner des formes simples

Le but de cet exercice est de réaliser un programme pour l'affichage d'un dessin composé de formes simples (carrés, rectangles, etc.). Pour dessiner ces formes, 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 dessins, nous définissons les types OCaml suivants :

```
type forme = Carre of int | Rectangle of int * int | Cercle of int | Point
type coordonnees = int * int
type objet = { f : forme; c : Graphics.color; xy: coordonnees  }
type dessin = objet list
```

Ainsi, une forme est soit un carré `Carre(n)`, où `n` est la longueur des côtés, un rectangle `Rectangle(w,h)` de largeur `w` et de hauteur `h`, un cercle `Cercle(r)` de rayon `r` ou un point `Point`. Les objets du dessin sont représentés par des valeurs de type `objets` qui spécifient la forme, la couleur et la position des formes à l'écran. Un dessin de type type `dessin` est alors simplement représenté par une liste de valeurs de type `objet`.

Question 3 Définir une variable `d` de type `dessin` pour représenter un dessin composé des objets suivants :

- un carré bleu de 10 pixels positionné en (10, 10)
- un rectangle rouge de largeur 20 et hauteur 30 positionné en (40,40)
- un point de couleur noir en (100, 100)
- un cercle noir de rayon 5 positionné en (150, 150)

Question 4 En utilisant les fonctions de la bibliothèque `Graphics`, écrire une fonction `affiche_forme` de type `coordonnees -> forme -> unit` tel que `affiche_forme (x,y) f` affiche à l'écran une forme `f` à la position (`x`, `y`).

Question 5 En utilisant la fonction précédente, écrire une fonction `affiche_objet` de type `objet -> unit` qui dessine un objet à l'écran.

Question 6 En utilisant la fonction précédente, écrire une fonction `dessine` de type `dessin -> unit` qui dessine une liste d'objets.

On va maintenant définir une fonction d'ordre supérieur pour appliquer des transformations géométriques aux objets. Ces transformations seront représentées par des fonctions de type `objet -> objet`.

Question 7 Écrire une fonction *récursive terminale* `applique` de type `(objet -> objet) -> dessin -> dessin` tel que `applique tr d` renvoie un nouveau dessin contenant les objets du dessin `d` transformés à l'aide de la fonction `tr`.

Question 8 Écrire une fonction `en_bleu` de type `objet -> objet` tel que `en_bleu o` renvoie un nouvel objet, identique à `o`, mais de couleur bleu.

Question 9 Écrire une fonction `translation` de type `int * int -> objet -> objet` tel que `translation (vx, vy) o` renvoie un nouvel objet, identique à `o`, mais dont la position est translatée par le vecteur de coordonnées `(vx, vy)`.

Question 10 Écrire une fonction `double` de type `objet -> objet` tel que `double o` renvoie un nouvel objet, identique à `o`, mais avec des dimensions doubles (dans le cas d'un carré, d'un rectangle ou d'un cercle).

Question 11 En utilisant les quatre fonctions précédentes, donner une expression pour transformer le dessin `d` (Question 3) de sorte que tous ses objets soient bleus, leurs coordonnées translatées par le vecteur `(10,10)` et avec des dimensions doublées.

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`

`draw_rect : int -> int -> int -> int -> unit`
: `draw_rect x y w h` dessine un rectangle de largeur `w` et de hauteur `h` dont le coin en bas à gauche est en `(x, y)`

`draw_circle : int -> int -> int -> unit`
: `draw_circle x y r` dessine un cercle de rayon `r` avec un centre en `(x, y)`

`plot : int -> int -> unit`
: `plot x y` dessine un point en `(x, y)`