« Processus d'évaluation » DEUG MIAS — UPMC

queinnec

Octobre 2001 — janvier 2002

Exercice 1 – Mon beau sapin

Le fichier[1], nommé `sapin.scm`, qui suit contient 13 fonctions permettant de dessiner des sapins. Le but de cet exercice est de vous entrainer à lire et à modifier des programmes écrits en Scheme. On peut bien sûr utiliser le système Scheme lui-même pour mieux comprendre certaines fonctions.

```
;;; Id : sapin.scm,v 1.62001/10/0408 : 52 : 41 titouExp
;;; La version de <christian.queinnec@lip6.fr> des sapins de Titou Durand.

;;; rectangle-plein: Coordonnée * Coordonnée * Coordonnée * Coordonnée -> Image
;;; avec Coordonnée = Nombre/entre -1 et 1/
;;; (rectangle-plein x1 y1 x2 y2) construit une image contenant un
;;; rectangle noir dont le coin haut-gauche est en x1,y1 et le coin
;;; bas-droit est en x2,y2.

(define (rectangle-plein x1 y1 x2 y2)
  (overlay (filled-triangle x1 y1 x2 y2 x1 y2)
           (filled-triangle x1 y1 x2 y1 x2 y2) ) )

;;; quart-de-tour-a-gauche: Image -> Image
;;; (quart-de-tour-a-gauche image) construit une image semblable à son argument
;;; pivotée d'un quart de tour vers la gauche.

(define (quart-de-tour-a-gauche image)
  (quarter-turn-right
   (quarter-turn-right
    (quarter-turn-right image) ) ) )

;;; Les images de base sont construites par ces fonctions:

;;; triangle: -> Image
;;; (triangle) construit un triangle isocèle rectangle de côté 50 pixels.

(define (triangle)
  (let ((t (filled-triangle -1 -1 -1 1 1 -1)))
    (resize-image t 50 50) ) )

;;; rectangle-noir: -> Image
;;; (rectangle-noir) construit un rectangle noir (deux fois plus haut que
;;; large) ou, plus précisément, de 25*50 pixels.

(define (rectangle-noir)
  (let ((r (rectangle-plein -1 -1 1 1)))
    (resize-image r 25 50) ) )
```

[1] `http://www.infop6.jussieu.fr/deug/2002/mias/mias-a/public/sapin.scm`

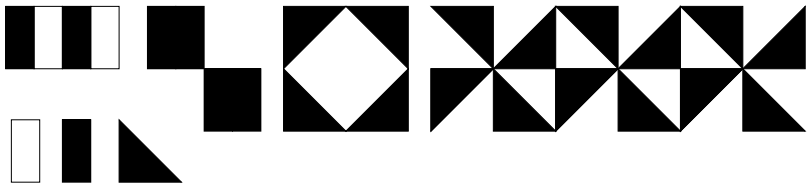
```
;;;rectangle-blanc: -> Image
;;;(rectangle-blanc) construit un rectangle blanc (deux fois plus haut
;;;que large) ou, plus précisément, de 25*50 pixels.

(define (rectangle-blanc)
  (let ((r (line 1 1 1 1)))
    (resize-image r 25 50) ) )

;;;collage-horizontal: LISTE[Image]/non vide/ -> Image
;;;(collage-horizontal dessins) concatène horizontalement une liste
;;;d'images en une seule image.

(define (collage-horizontal dessins)
  (quart-de-tour-a-gauche
   (collage-vertical
    (map quarter-turn-right dessins) ) ) )

;;;collage-vertical: LISTE[Image]/non vide/ -> Image
;;;(collage-vertical dessins) concatène verticalement une liste
;;;d'images en une seule image.

(define (collage-vertical dessins)
  (if (pair? (cdr dessins))
      (stack (car dessins)
             (collage-vertical (cdr dessins)) )
      (car dessins) ) )

;;;demi-tronc: nat -> LISTE[Image]
;;;(demi-tronc n) construit un demi-tronc bordé de blanc à droite (pour
;;;un sapin de hauteur n).

(define (demi-tronc n)
  (cons (rectangle-noir) (collage-multiple (- n 1) (rectangle-blanc))) )

;;;demi-etage-droit: nat * nat -> LISTE[Image]
;;;(demi-etage-droit i n) construit le i-ème étage d'un sapin de hauteur n.

(define (demi-etage-droit i n)
  (append (collage-multiple i (rectangle-noir))
          (list (triangle))
          (collage-multiple (- n i 2) (rectangle-blanc)) ) )

;;;collage-multiple: nat * Image -> LISTE[Image]
;;;(collage-multiple i dessin) construit une liste de i fois le même dessin.

(define (collage-multiple i dessin)
  (if (> i 0)
      (cons dessin (collage-multiple (- i 1) dessin))
      (list) ) )

;;;demi-ramure: nat * nat -> LISTE[LISTE[Image]]
;;;(demi-ramure i n) construit la liste de tous les étages d'un
;;;demi-sapin, chaque étage étant constitué d'une liste d'images
;;;élémentaires.

(define (demi-ramure i n)
  (if (< i (- n 1))
```

```
      (cons (demi-etage-droit i n)
            (demi-ramure (+ i 1) n) )
      (list (demi-tronc n)) ) )

;;;mon-beau-sapin: nat/>=2/ -> Image
;;;(mon-beau-sapin n) construit une image représentant un sapin de
;;;hauteur n (tronc compris). Le tronc mesure 1 et il y a n-1 étages de
;;;ramure.  Chaque étage est décalé par rapport au précédent. Le
;;;sapin n'est composé que de deux images: un bout de tronc et un
;;;petit triangle.

(define (mon-beau-sapin n)
  ;;NOTA: on construit d'abord la moitié droite du sapin.
  (let* ((dessins-demi-sapin (demi-ramure 0 n))
         (demi-sapin (collage-vertical
                      (map collage-horizontal dessins-demi-sapin)) )
    (collage-horizontal
     (list (mirror-image demi-sapin) demi-sapin) ) ) )

(write-image-as-epsf (mon-beau-sapin 5) "sapin.epsf")
;;;fin de sapin.scm
```

Question 1: Construire un sapin de hauteur 6.

Question 2: Étudiez les sept premières fonctions (c'est-à-dire `rectangle-plein`, `quart-de-tour-a-gauche`, `triangle`, `rectangle-noir` et `rectangle-blanc` enfin `collage-horizontal` et `collage-vertical`) et, pour chacune d'entre elles, donnez une expression la mettant en œuvre. Il n'est pas interdit d'avoir des soucis esthétiques !

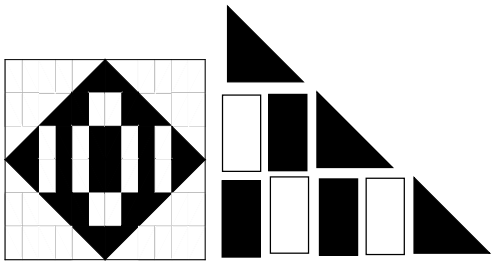
Question 3: Écrire les expressions construisant les valeurs suivantes (il y a de nombreuses solutions possibles), (les trois figures de base apparaissent en bas à gauche du dessin) :



Question 4: Quel est le type de la variable `dessins-demi-sapin` dans la fonction `mon-beau-sapin` ? Quel est le type de la variable `demi-sapin` dans cette même fonction ?

Question 5: Écrire une unique fonction, nommée `beau-sapin`, contenant tout ce qui est nécessaire pour dessiner un sapin. Aucune des fonctions utilitaires du fichier donné ci-avant ne doit être visible.

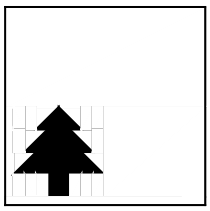
Question 6: Écrire une fonction, nommée `losange`, sur les mêmes bases que la fonction `mon-beau-sapin`, prenant un entier *n* et construisant un un carré échancré inscrit dans un cadre de hauteur 100*n* pixels. Ainsi l'expression `(losange 3)` mène-t'elle à la figure suivante (où l'on a identifié une méthode de tracé) :



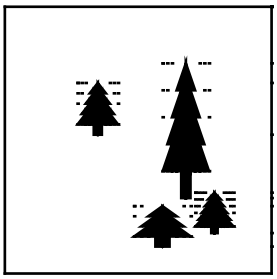
Question 7: Réaliser une fonction, nommée `image-reduite`, prenant une image (de taille quelconque) et les coordonnées de deux points et insérant cette image dans une nouvelle image carrée de 100 pixels de côté avec comme coin bas gauche le premier point et comme coin haut droit le second point. Ainsi :

```
(reduire-image (mon-beau-sapin 4) -0.9 -0.9 0 0)
```

conduit à l'image :



Question 8: On veut maintenant construire une forêt de sapin comme le montre l'image suivante.



Écrire une fonction, nommée `foret-de-sapin`, prenant un entier naturel et construisant une forêt contenant ce nombre de sapins. Les sapins seront placés aléatoirement, on utilisera, pour cela, la fonction `random` dont voici la spécification :

```
;;; random: nat/inférieur à 2 puissance 31/ -> nat
;;; (random max) rend un entier aléatoire compris entre 0 inclus et
;;; max exclus. La probabilité est uniforme.
```