Université Bordeaux 1, Sciences et technologies
UFR Informatique et Mathématiques

Maîtrise d'informatique
Session de janvier 2002

Techniques et Fondement de la Programmation (TFP)
(partie "techniques")
Le 25 janvier 2002

Durée : trois heures (les deux parties)

Tous documents autorisés sauf photocopies illégales de livres. Votre voisin n'est pas un document.

Rappel : Soyez bref et clair ! Personne n'est impressionné par la quantité !

Bon courage !

Exercice 1. (7p)

Identifier les problèmes du programme suivant (5p), et écrire une version améliorée (2p) :

```
; Name: all-fs
; Arguments: a list l and a function fn
; Returns: a list of all members of l
;          for which fn returns a value > n
(defun all-fs (l f n)
  (let ((retval nil))
  (labels ((fn (l)
             (if (not (null l))
                 (if (> (funcall f (car l)) n)
                     (progn
                       (append retval (list (car l)))
                       (fn (cdr l)))
                     (fn (cdr l))))))
    (fn l))
    retval))
```

Exercice 2. (6p)

Dans une application graphique, on souhaite capturer différents types d'évènements (disons `key-press-event` et `button-press-event`) sur différents types de widgets (disons `text-widget` et `canvas-widget`).

On souhaite que l'arborescence des classes soit extensible à d'autres types de widgets et d'évènements. De plus, on souhaite vérifier à chaque réception d'un évènement que les coordonnées de l'évènement représentent un point à l'intérieur du widget (on suppose que le widget est rectangulaire).

Modéliser les classes, les fonctions génériques et les méthodes de cette application (écrire uniquement les en-têtes et non le code même).

Exercice 3. (7p)

Dans un éditeur de dessins, on souhaite maintenir la notion de *point* implicite de manière à ce que les commandes de dessins tracent à partir de ce point. De plus, on a besoin d'une commande pour déplacer le point implicite sans dessiner. On souhaite pouvoir écrire des expressions de ce type :

```
(let ((x 10)
      (y 20)
      (w 40)
      (h 20))
  (with-implicit-point
    (moveto startx starty)
    (lineto x (+ y h))
    (lineto (+ x w) (+ y h))
    (lineto (+ x w) y)
    (lineto x y)))
```

Cette expression dessine un rectangle avec un coin dans le point (10,20) et avec dimensions 40 et 20. La fonction `moveto` déplace le point (initialement à (0,0)). La fonction `lineto` trace un segment de droite entre la valeur actuelle du point implicite et le point donné par ses arguments, puis déplace le point implicite.

Écrire une macro `with-implicit-point` permettant des expressions de ce type. Il est important que `moveto` et `lineto` soient des *fonctions* normales définies uniquement dans le corps de la macro, et que le point ne soit pas accessible explicitement par des fonctions autre que `moveto` et `lineto`. On suppose l'existence d'une fonction (X11 par exemple) `draw-line` avec quatre paramètres : `x1`, `y1`, `x2` et `y2`.