ANNÉE UNIVERSITAIRE 2010/2011
1ÈRE SESSION D'AUTOMNE

Parcours : CSB5
Code UE : INF353
Épreuve : Programmation 3
Date : Mardi 14 décembre 2010
Heure : 11h00
Durée : 1h30
Documents : autorisés
Épreuve de Mme Irène Durand

Le barème est donné à titre **indicatif**. Le sujet comporte 2 pages.

Exercice 1 *(4pts)*

Écrire une fonction `test-predicates (objects pred-vector)` qui prend en paramètre une liste d'objets `objects` et un tableau à une dimension `pred-vector` lequel contient des prédicats applicables aux objets, et qui retourne un tableau contenant dans chaque case i la liste des objets de `objects` qui vérifient le prédicat se trouvant dans la case i de `pred-vector`.
Exemple :

```
CL-USER> (setf *pred-vector*
               (make-array
                5
                :initial-contents
                (list
                 #'zerop #'oddp #'evenp (lambda (o) (zerop (rem o 3))) #'plusp)))
#(#<FUNCTION ZEROP> #<FUNCTION ODDP> #<FUNCTION EVENP>
  #<FUNCTION (LAMBDA #) {1003580EF9}> #<FUNCTION PLUSP>)
CL-USER> (test-predicates '(-2 -1 0 1 2 3 0) *pred-vector*)
#((0 0) (3 1 -1) (0 2 0 -2) (0 3 0) (3 2 1))
CL-USER>
```

Exercice 2 *(7pts)*

Dans un système d'information géographique, on souhaite pouvoir déterminer si un *point* se trouve dans une *zone* donnée. Il y a deux grands types de zones : les *zones fonctionnelles* représentées pas leur fonction caractéristique et les *zones élémentaires* représentées par certaines de leurs caractéristiques géométriques. Parmi les zones élémentaires, on distingue

- les *zones circulaires* représentées par leur centre et leur rayon ;
- les *zones polygonales* représentées par la liste des points correspondant aux coins du polygone.

1. Définir l'ensemble des classes correspondant aux objets décrits ci-dessus.
2. Dessiner la hiérarchie des classes.
3. Définir l'opération `point-in-zone`.
4. Implémenter l'opération `point-in-zone` pour les zones fonctionnelles.
5. Implémenter l'opération `point-in-zone` pour les zones circulaires.

Exercice 3 *(2pts)*

Soit le programme ci-dessous :

```
(defparameter *i* 0)
(let ((*i* 5)
      (j 2))
  (defun f (x)
    (+ x *i* j)))
```

Que retournent les expressions suivantes :

1. `(f 3)`
2. `(let ((*i* 5)) (f 3))`

Exercice 4 *(7pts)*

Soit la macro `push-ntimes` définie ci-dessous :

```
(defmacro push-ntimes (e l &optional (n 1))
  `(progn
     ,@(loop
         repeat n
         collect `(push ,e ,l))))
```

1. Compléter le scénario suivant

```
CL-USER> (defparameter *l* '(1 2 3 4))
                                          ;; Réponse 1
CL-USER> (macroexpand-1 '(push-ntimes 0 *l* 3))
                                          ;; Réponse 2

CL-USER> (push-ntimes 0 *l* 3)
                                          ;; Réponse 3
CL-USER> *l*
                                          ;; Réponse 4
CL-USER> (defparameter *x* 5)
                                          ;; Réponse 5
CL-USER> (macroexpand-1
           '(push-ntimes (incf *x*) *l* 3))
                                          ;; Réponse 6

CL-USER> (push-ntimes (incf *x*) *l* 3)
                                          ;; Réponse 7
```

2. Comment expliquer que `*x*` ait été incrémenté plusieurs fois ? Comment s'appelle ce phénomène ?
3. Corriger la macro de manière à ce que le scénario se poursuive de la manière suivante :

```
CL-USER> *l*
(8 7 6 0 0 0 1 2 3 4)
CL-USER> *x*
8
CL-USER> (push-ntimes (incf *x*) *l* 3)
(9 9 9 8 7 6 0 0 0 1 2 3 4)
```

FIN