UNIVERSITÉ BORDEAUX 1 Sciences Technologies DEVUIP Service Scolarité	ANNÉE UNIVERSITAIRE 2013/2014 1ÈRE SESSION D'AUTOMNE **Parcours :** IN501 **Code UE :** IN5011 **Épreuve :** Programmation 3 **Date :** Lundi 16 décembre 2013 **Heure :** 08h30 **Durée :** 1h30 Documents : autorisés Épreuve de Mme Irène Durand	Département DL Licence

Le barème est donné à titre **indicatif**. Le sujet comporte 4 pages.

Exercice 1 *(6pts)*

Soit la fonction `cartesian-product (l1 l2)` qui construit la liste des couples d'éléments (e_1, e_2) tels que $e_1 \in l_1$ et $e_2 \in l_2$ en respectant l'ordre visible sur les exemples. Exemples :

```
CL-USER> (cartesian-product '() '())
NIL
CL-USER> (cartesian-product '(1 2) '(a b c))
((1 A) (1 B) (1 C) (2 A) (2 B) (2 C))
```

1. Écrire une version **itérative** de cette fonction.
2. Écrire une version **récursive** de cette fonction. On n'hésitera pas à faire une fonction auxiliaire.
3. La fonction récursive est-elle récursive terminale ?

Exercice 2 *(8pts)*

Un jeu se joue à deux joueurs (numérotés 1 et 2) sur une grille carrée représentée par un tableau à deux dimensions. La longueur des côtés de la grille est donnée par la variable `*size*`.

```
CL-USER> (defvar *size* 5)
*SIZE*
```

Au départ la grille est vide (chaque case du tableau a la valeur `0`). Par la suite, les cases pourront être occupées par au plus un joueur, la case du tableau contient alors le numéro du joueur.

1. Écrire une fonction `make-empty-grid ()` qui crée une grille vide. Exemple :

   ```
   CL-USER> (setf *grid* (make-empty-grid))
   #2A((0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0))
   ```

Pour mieux comprendre les exemples, on suppose déjà écrite une fonction `print-grid (grid)` qui imprime la grille de manière plus lisible avec les numéros de ligne et de colonne. Exemple :

```
CL-USER> (setf (aref *grid* 0 0) 1)
1
CL-USER> (print-grid *grid*)
   | 0 | 1 | 2 | 3 | 4 |
-------------------------
 0 | 1 |   |   |   |   |
-------------------------
 1 |   |   |   |   |   |
-------------------------
 2 |   |   |   |   |   |
-------------------------
 3 |   |   |   |   |   |
-------------------------
 4 |   |   |   |   |   |
-------------------------
NIL
```

Soit le prédicat `square-full-p (grid x y)` qui retourne `NIL` si la case de coordonnées (x, y) [1] est vide, le numéro du joueur qui occupe la case sinon. Exemples :

```
CL-USER> *grid*
#2A((1 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0) (0 0 0 0 0))
CL-USER> (square-full-p *grid* 1 1)
NIL
CL-USER> (square-full-p *grid* 0 0)
1
```

2. Implémenter le prédicat `square-full-p`.

On utilise des instances de la classe `direction` pour représenter une direction de déplacement sur la grille.

```
(defclass direction ()
  ((dir-x :initarg :dir-x :type function :reader dir-x)
   (dir-y :initarg :dir-y :type function :reader dir-y)))
```

Les valeurs des créneaux `dir-x` et `dir-y` sont des fonctions appartenant à la liste `*1directions*` définie par

```
(defvar *1directions* (list #'identity #'1+ #'1-))
```

Un object de la classe direction permet de se déplacer d'une case dans la grille, horizontalement, verticalement ou diagonalement. Si on est dans la case (x, y), la case suivante est obtenue en appliquant respectivement les fonctions contenues dans les créneaux `dir-x` et `dir-y` à x et y.
Soit la fonction `make-direction (dir-x dir-y)` qui retourne une instance de la classe `direction` dont les créneaux `dir-x` et `dir-y` ont les valeurs passées en paramètre. Exemple :

1. x est le numéro de ligne et y le numéro de colonne.

```
CL-USER> (make-direction #'1+ #'identity)
[#<FUNCTION 1+>,#<FUNCTION IDENTITY>]
```

3. Implémenter la fonction `make-direction`.

Soit l'opération booléenne

```
(defgeneric homogeneous-dir-p (grid x y direction)
  (:documentation
   "returns T if all squares from square (x,y) to the side
    in the direction DIRECTION have the same value"))
```

qui retourne `T` si toutes les cases comprises entre la case (x, y) et le bord de la grille en suivant la direction indiquée ont toutes la même valeur. Exemples :

```
CL-USER> (print-grid *grid*)
   | 0 | 1 | 2 | 3 | 4 |
-------------------------
 0 | 1 | 1 | 1 |   |   |
-------------------------
 1 |   | 1 |   |   |   |
-------------------------
 2 |   |   | 2 |   |   |
-------------------------
 3 |   | 2 |   |   |   |
-------------------------
 4 | 2 |   |   |   |   |
-------------------------
NIL
CL-USER> (homogeneous-dir-p *grid* 2 2 (make-direction #'1- #'1+))
NIL
CL-USER> (homogeneous-dir-p *grid* 2 2 (make-direction #'1+ #'1-))
T
CL-USER> (homogeneous-dir-p *grid* 0 2 (make-direction #'1- #'identity))
T
CL-USER> (homogeneous-dir-p *grid* 0 2 (make-direction #'1+ #'identity))
NIL
CL-USER> (homogeneous-dir-p *grid* 0 4 (make-direction #'1+ #'identity))
T
```

4. Implémenter l'opération booléenne `homogeneous-dir-p`
5. Sans toucher au code existant, proposer une solution pour que les créneaux d'une instance de la classe `direction` soient toujours corrects, c'est-à-dire vérifient les deux conditions suivantes :
   (a) leur valeur appartient à la liste `*1direction*`,
   (b) on n'a pas les deux créneaux avec la valeur `#'identity`.

Exercice 3 *(6pts)*

On souhaite une macro `setq2 (variable1 variable2 expression)` permettant d'affecter la valeur de l'expression `expression` aux deux variables `variable1` et `variable2`. Des exemples sont présentés Figure 1. Un étudiant a proposé la solution suivante :

```
(defmacro setq2 (variable1 variable2 expression)
  `(progn
     (setq ,variable1 ,expression)
     (setq ,variable2 ,expression)))
```

1. En utilisant la version de l'étudiant, compléter le scénario suivant :

```
CL-USER> (list *x* *y*)
(0 1)
CL-USER> (macroexpand-1 '(setq2 *x* *y* 3))
                                ;; Réponse A

CL-USER> (setq2 *x* *y* 3)
                                ;; Réponse B
CL-USER> (list *x* *y*)
                                ;; Réponse C
CL-USER> (setf *x* 0)
                                ;; Réponse D
CL-USER> (macroexpand-1 '(setq2 *x* *y* (incf *y*)))
                                ;; Réponse E

CL-USER> (setq2 *x* *y* (incf *y*))
                                ;; Réponse F
CL-USER> (list *x* *y*)
                                ;; Réponse G
```

2. Quel est le problème avec la solution de l'étudiant ?
3. Modifier la macro de manière à ce qu'on obtienne le résultat escompté, présenté Figure 1.

FIN

```
CL-USER> (list *x* *y*)
(0 1)
CL-USER> (setq2 *x* *y* 3)
3
CL-USER> (list *x* *y*)
(3 3)
CL-USER> (setf *x* 0)
0
CL-USER> (setq2 *x* *y* (incf *y*))
4
CL-USER> (list *x* *y*)
(4 4)
```

FIGURE 1 – Exemples pour la macro `setq2`