université de BORDEAUX Collège Sciences et Technologies DEVUIP Service Scolarité	ANNÉE UNIVERSITAIRE 2014/2015 2ÈME SESSION D'AUTOMNE **Parcours :** IN501 et IM500 **Code UE :** IN5011 **Épreuve :** Programmation 3 **Date :** juin 2015 **Heure :** 08h30 **Durée :** 1h30 Documents : autorisés Épreuve de Mme Irène Durand	Département D L Licence

Le barème est donné à titre **indicatif**.
Le sujet comporte 2 pages plus 3 pages annexes.

Exercice 1 *(2pts)*

Soit la fonction `iota (n)` qui s'applique à un entier `n`, et qui retourne une liste de longueur `n` qui contient les entiers allant de 0 à $n-1$. Exemples :

```
CL-USER> (iota 0)
NIL
CL-USER> (iota 1)
(0)
CL-USER> (iota 10)
(0 1 2 3 4 5 6 7 8 9)
```

1. Donner une implémentation de la fonction `iota`.

Exercice 2 *(3pts)*

Soit la fonction `rmap (q l fun)` qui s'applique à un entier `q`, une liste `l` et une fonction `fun` unaire pouvant s'appliquer à chacun des éléments de `l`. La fonction `rmap` applique la fonction `fun` aux `q` premiers éléments de la liste en les parcourant en sens inverse. Cette fonction ne construit rien et retourne `NIL`. Si la longueur de la liste est inférieure à `q`, la fonction `fun` est appliquée à tous les éléments en allant du dernier au premier. Exemples :

```
CL-USER> (rmap 5 (iota 10) #'prin1)
43210
NIL
CL-USER> (rmap 15 (iota 10) #'prin1)
9876543210
NIL
```

2. Donner une implémentation de la fonction `rmap`.

Exercice 3 *(2pts)*

Soit la fonction `gensyms (n)` qui retourne une liste de `n` nouveaux symboles obtenus par des appels à la fonction Common Lisp `gensym ()`. Exemple :

```
CL-USER> (gensyms 3)
(#:G997 #:G998 #:G999)
```

3. Donner une implémentation de la fonction `gensyms`.

Exercice 4 *(2pts)*
Soit la fonction `binding (q s1 s2)` qui s'applique à un entier `q` et deux symboles `s1` et `s2` et qui retourne une liste destinée à être un élément d'un environnement `let` et qui contient le symbole `s2` suivi de l'appel `(nthcdr q s1)`. Exemple :

```
CL-USER> (binding 5 'a 'b)
(B (NTHCDR 5 A))
```

4. Donner une implémentation de la fonction `binding`.

Exercice 5 *(2pts)*
Soit la fonction `bindings (q symbols)` qui s'applique à un entier `q` et à une liste de symboles `symbols (s1 ...sk)` et qui retourne la liste

```
((S2 (NTHCDR Q S1))
 (S3 (NTHCDR Q S2))
 ...
 (Sk (NTHCDR Q Sk-1)))
```

Exemple :

```
CL-USER> (bindings 5 '(a b c e))
((B (NTHCDR 5 A)) (C (NTHCDR 5 B)) (E (NTHCDR 5 C)))
```

5. En utilisant la fonction `binding (q s1 s2)` de l'exercice précédent, implémenter la fonction `bindings (q symbols)`.

Exercice 6 *(3pts)*
Soit la macro suivante :

```
(defmacro divide-and-rmap (list n fun)
  (let* ((gensyms (gensyms (1+ n)))
         (quotient (gensym)))
    `(let* ((,quotient (floor (length ,list) ,n))
            (,(car gensyms) ,list)
            ,@(bindings quotient gensyms))
       ,@(mapcar (lambda (gensym)
                   `(rmap ,quotient ,gensym ,fun))
                 (reverse gensyms)))))
```

.

6. Évaluer l'expression suivante : `(macroexpand-1 '(divide-and-rmap *l* 3 #'prin1))`
7. En supposant que `*l*` contienne `(0 1 2 3 4 5 6 7 8 9)`,
   que donne l'appel `(divide-and-rmap *l* 3 #'prin1)` ?

Exercice 7 *(6pts)*
Soit le code donné en Annexe page 3.

8. Dessiner la hiérarchie des classes.
9. Compléter le scénario de la Figure 1 donnée page 5.

FIN

Annexe

```
(defclass abstract-enumerator () ())

(defgeneric init-enumerator (enumerator)
  (:documentation "reinitializes and returns ENUMERATOR"))

(defgeneric next-element-p (enumerator)
  (:documentation "T if there is a next element"))

(defgeneric next-element (enumerator)
  (:documentation "the next element"))

(defgeneric call-enumerator (enumerator)
  (:documentation
   "returns as first value the next object produced by ENUMERATOR
    if it exists NIL otherwise
    as second value T if object was produced"))

(defmethod call-enumerator ((e abstract-enumerator))
  (if (next-element-p e)
      (values (next-element e) t)
      (values nil nil)))

(defmethod init-enumerator ((e abstract-enumerator))
  e)

(defclass list-enumerator (abstract-enumerator)
  ((initial-list :initarg :initial-list :reader initial-list)
   (current-list :initarg :current-list :accessor current-list))
  (:documentation "enumerators of the elements of a list"))

(defmethod next-element-p ((e list-enumerator))
  (not (endp (current-list e))))

(defmethod next-element ((e list-enumerator))
  (pop (current-list e)))

(defun make-list-enumerator (l)
  (make-instance 'list-enumerator :initial-list l :current-list l))

(defmethod init-enumerator :after ((e list-enumerator))
  (setf (current-list e) (initial-list e)))

(defclass fun-mixin ()
  ((fun :initarg :fun :reader fun)))
```

```
(defclass inductive-enumerator (abstract-enumerator fun-mixin)
  ((initial-value :initarg :initial-value :accessor initial-value)
   (current-value :initarg :current-value :accessor current-value)))

(defmethod next-element-p ((e inductive-enumerator))
  t)

(defmethod next-element ((e inductive-enumerator))
  (current-value e))

(defmethod next-element :after ((e inductive-enumerator))
  (setf (current-value e)
        (funcall
         (fun e)
         (current-value e))))

(defmethod init-enumerator :after ((e inductive-enumerator))
  (setf (current-value e) (initial-value e)))

(defun make-inductive-enumerator (initial-value fun)
  (make-instance 'inductive-enumerator
                 :fun fun
                 :initial-value initial-value
                 :current-value initial-value))
```

Scénario

```
CL-USER> (defparameter *e* (make-list-enumerator '(3 4)))
                            ;; Réponse A
CL-USER> (call-enumerator *e*)
                            ;; Réponse B

CL-USER> (call-enumerator *e*)
                            ;; Réponse C

CL-USER> (call-enumerator *e*)
                            ;; Réponse D

CL-USER> (init-enumerator *e*)
#<LIST-ENUMERATOR {1020F31473}>
CL-USER> (call-enumerator *e*)
                            ;; Réponse E

CL-USER> (defparameter *i* (make-inductive-enumerator 5 (lambda (x) (* 2 x))))
                            ;; Réponse F
CL-USER> (call-enumerator *i*)
                            ;; Réponse G

CL-USER> (call-enumerator *i*)
                            ;; Réponse H

CL-USER> (call-enumerator *i*)
                            ;; Réponse I

CL-USER> (init-enumerator *i*)
#<INDUCTIVE-ENUMERATOR {1022126373}>
CL-USER> (call-enumerator *i*)
                            ;; Réponse J
```

FIGURE 1 – Scénario à compléter