
	<p>ANNÉE UNIVERSITAIRE 2009/2010 1ÈRE SESSION D'AUTOMNE</p> <p>Parcours : CSB5 Code UE : INF353 Épreuve : Programmation 3 Date : Mardi 15 décembre 2009 Heure : 11h00 Durée : 12h30 Documents : autorisés Épreuve de Mme Irène Durand</p>	
-----------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

Le barème est donné à titre **indicatif**.

Le sujet comporte 3 pages et une annexe.

Exercice 1 (5pts)

- Écrire une fonction `expand (l v)` qui retourne une liste contenant autant de sous-listes qu'il y a d'éléments dans `l` contenant chacune uniquement des `v`. La *i*ème sous-liste a comme longueur le *i*ème élément de `l`. Exemples :

```
CL-USER> (expand '(1 0 3 2) 2)
((2) NIL (2 2 2) (2 2))
CL-USER> (expand '(1 2 3) nil)
((NIL) (NIL NIL) (NIL NIL NIL))
```

- Écrire une fonction `multi (l v)` qui retourne la liste des éléments de `l` avec une multiplicité de `v`. Exemples :

```
CL-USER> (multi '(1 2 4 3) 2)
(1 1 2 2 4 4 3 3)
CL-USER> (multi '(1 2 4 3) 0)
NIL
CL-USER> (multi '(1 2 4 3) 1)
(1 2 4 3)
```

Exercice 2 (4pts)

Écrire une fonction `maparray (funs tab)` où `funs` et `tab` sont des tableaux à une dimension de même taille. Le tableau `funs` contient des fonctions tandis que le tableau `tab` contient des valeurs telles que `funs[i]` est applicable à `tab[i]` pour tout indice `i` du tableau. La fonction `maparray (funs tab)` doit retourner un nouveau tableau `a` tel que `a[i] = f(tab[i])` avec `f = funs[i]` pour tout `i`. Exemple :

```
CL-USER> (defparameter *tab* (make-array 4 :initial-element 2))
*TAB*
CL-USER> (defparameter *funs*
  (make-array
   4
   :initial-contents (list (lambda (x) x)
                           (lambda (x) (* x x))
                           (lambda (x) (* x x x))
                           (lambda (x) (* x x x x)))))
*FUNS*
CL-USER> (maparray *funs* *tab*)
#(2 4 8 16)
```

Exercice 3 (5pts)

Soit le programme donné en Annexe.

1. Combien de créneaux a un objet de la classe `state` ?
2. Soit la session donnée ci-dessous. Compléter chacune des lignes contenant un commentaire réponse `i` par la valeur manquante.

```
CL-USER> (setf *internal-number* 0)
;; réponse 1
CL-USER> (setf *states* nil)
;; réponse 2
CL-USER> (defparameter *state1* (make-state 's1))
*STATE1*
CL-USER> *state1*
<S1>
CL-USER> (internal-number *state1*)
;; réponse 3
CL-USER> (defparameter *state2* (make-state 's2))
;; réponse 4
CL-USER> *state2*
;; réponse 5
CL-USER> (internal-number *state2*)
;; réponse 6
CL-USER> (defparameter *state3* (make-state 's1))
*STATE3*
CL-USER> *state3*
<S1>
CL-USER> (eq *state1* *state3*)
;; réponse 7
CL-USER> (internal-number *state3*)
;; réponse 8
CL-USER> (sort-states (list *state1* *state2* *state3*))
;; réponse 9
```

Exercice 4 (6pts)

Un logiciel dispose d'un ensemble de fonctionnalités implémentées par un ensemble de fonctions. Pour simplifier, on suppose que toutes ces fonctions n'ont qu'un argument.

Certaines de ces fonctionnalités sont accessibles à l'utilisateur. Elles sont alors associées à une *commande*. L'exécution d'une commande, comprend deux phases, la première consiste à demander à l'utilisateur l'argument, la deuxième à appliquer la fonction associée à l'argument obtenu.

Dans une table de hachage, mémorisée dans une variable globale `*commandes*`, on associe à chaque nom de fonctionnalité, une fonction anonyme correspondant à la commande interactive associée.

1. Écrire une macro `creer-commande (nom arg-type &body body)` qui prend en paramètre `nom` un nom de fonction et `arg-type` un couple `(nom type)`. Un appel à cette macro doit définir (avec `defun`) une fonction de nom `nom` ayant pour corps les expressions de `body` ainsi qu'une entrée dans la table `*commandes*` dont la valeur est la fonction anonyme permettant de récupérer interactivement l'argument.
2. Écrire une fonction `appel-commande (nom)` qui déclenche l'exécution de la commande associée à `nom`.

Exemples :

```
CL-USER> (setf *commandes* (make-hash-table :test #'eq))
#<HASH-TABLE :TEST EQ :COUNT 0 {1003022401}>
CL-USER> (macroexpand-1
          '(creer-commande goto-line (noligne entier)
          (format t "Exécution de (goto-line ~A)%" noligne)))
(PROGN
 (DEFUN GOTO-LINE (NOLIGNE) (FORMAT T "Exécution de (goto-line ~A)%" NOLIGNE))
 (SETF (GETHASH 'GOTO-LINE *COMMANDES*)
       (LAMBDA () (GOTO-LINE (PROGN (FORMAT T "~A : " 'ENTIER) (READ))))))
T
CL-USER> (creer-commande goto-line (noligne entier)
          (format t "Exécution de (goto-line ~A)%" noligne))
#<FUNCTION (LAMBDA ()) {1003E00B19}>
CL-USER> (creer-commande save-file (name nom-fichier)
          (format t "Exécution de (save-file ~A)%" name))
#<FUNCTION (LAMBDA ()) {1004660CB9}>
CL-USER> (appel-commande 'goto-line)
ENTIER : 10
Exécution de (goto-line 10)
NIL
CL-USER> (appel-commande 'save-file)
NOM-FICHER : toto
Exécution de (save-file TOTO)
NIL
```

FIN

Annexe

```
(defvar *internal-number* 0)
(defvar *states* nil)

(defclass abstract-state ()
  ((internal-number :initform (incf *internal-number*)
                    :reader internal-number)))

(defclass state (abstract-state)
  ((state-contents :initarg :state-contents :reader state-contents)))

(defun find-state-from-contents (contents states)
  (car (member contents states :key #'state-contents)))

(defmethod print-object ((state state) stream)
  (format stream "<~A>" (state-contents state)))

(defun make-state (state-contents)
  (let ((state (find-state-from-contents state-contents *states*)))
    (unless state
      (setf state
             (make-instance 'state :state-contents state-contents))
      (push state *states*)))
  state))

(defun sort-states (states)
  (sort (copy-list states) #'< :key #'internal-number))
```