ANNÉE UNIVERSITAIRE 2009/2010
2IÈME SESSION DE PRINTEMPS

Parcours : CSB6
Code UE : INF356
Épreuve : Projet de Programmation 3
Date : Jeudi 24 juin 2010
Heure : 8h30
Durée : 1h30
Documents : autorisés
Épreuve de Mme Irène Durand

Le barème est donné à titre **indicatif**. Le sujet comporte 6 pages dont 3 d'annexe.

Exercice 1 *(2pts)*

L'archive `my-system.tgz` d'un système Lisp se trouve dans le répertoire courant à partir duquel `Emacs` et `Slime` ont été lancés. Soit la session `Slime` suivante :

```
CL-USER> (asdf-install::install "my-system.tgz")
Install where?
1) System-wide install:
   System in /usr/lib/sbcl/site-systems/
   Files in /usr/lib/sbcl/site/
2) Personal installation:
   System in /usr/labri/idurand/.sbcl/systems/
   Files in /usr/labri/idurand/.sbcl/site/
 --> 2
Installing /usr/labri/idurand/.sbcl/site/my-system.tgz in /usr/labri/idurand/.sbcl/site/,/usr/labr
my-system/
my-system/my-system.asd
my-system/my-system.lisp
...
NIL
CL-USER>
```

Quels sont les effets produits par cette session sur le système de fichiers ?

Exercice 2 *(5pts)*

1. Implémenter la fonction `lex<= (l1 l2)` qui réalise la comparaison pour l'ordre lexicographique de deux listes d'entiers.

Exemples :

```
CL-USER> (lex<= '() ())
T
CL-USER> (lex<= '(1) ())
NIL
CL-USER> (lex<= '(1) '(2 1))
T
CL-USER> (lex<= '(2) '(1 2))
NIL
```

```
CL-USER> (lex<= '(1 2 3) '(1 2))
NIL
CL-USER> (lex<= '(1 2 3) '(2 1))
T
CL-USER> (lex<= '(1 2) '(1 2 1))
T
```

Soit `*l*` une variable contenant une liste de listes d'entiers définie par :

```
CL-USER> (setf *l* '((1 2 3) (1 2) (1) () (2 1) (2 3)))
((1 2 3) (1 2) (1) NIL (2 1) (2 3))
```

2. Dans le scénario suivant, compléter la première ligne par une expression permettant de trier la liste `*l*` dans l'ordre lexicographique **sans détruire** `*l*` et la deuxième ligne par le résultat obtenu.

```
CL-USER> (... *l* ...)                                    ;; RÉPONSE A
...                                                       ;; RÉPONSE B
CL-USER> *l*
((1 2 3) (1 2) (1) NIL (2 1) (2 3))
```

Exercice 3 *(13pts)*

Dans une application, on doit gérer un ensemble de personnes dont on connaît le nom et la date de naissance. Soit l'extrait de programme donné Figure 1, page 4. En cours d'exécution, l'ensemble des personnes est stocké sous forme de liste dans la variable globale `*persons*`.

1. Donner une expression donnant la liste triée - sans doublon - des années de naissance de toutes les personnes.
2. On suppose que le programme vient d'être chargé. Compléter le scénario suivant.

```
CL-USER> (setf *date*
               (make-instance 'date :year 2010 :month 12 :day 25))
                                                          ;; RÉPONSE C
CL-USER> (setf *person*
               (make-instance 'person :name "Oscar" :birth-date *date*))
                                                          ;; RÉPONSE D
```

Grâce au mécanisme des *read-macros*, le complément de code donné Figure 2, page 5 permet de lire des données représentant des dates et des personnes au format indiqué par les deux exemples suivants :

```
;; format pour les dates
[DATE :YEAR 2010 :MONTH 12 :DAY 25]

;; format pour les personnes
[PERSON :NAME "Oscar" :BIRTH-DATE [DATE :YEAR 2010 :MONTH 12 :DAY 25]]
```

3. En supposant que le code vient d'être chargé, compléter le scénario suivant.

```
CL-USER> (defparameter *save-readtable* *readtable*)
*save-readtable*
CL-USER> (setf *readtable* *my-readtable*)
#<READTABLE {1003A13E51}>
CL-USER> [DATE :YEAR 2010 :MONTH 12 :DAY 25]
                                                    ;; RÉPONSE E
CL-USER> [PERSON :NAME "Oscar"
                 :BIRTH-DATE [DATE :YEAR 2010 :MONTH 12 :DAY 25]]
                                                    ;; RÉPONSE F
CL-USER> (setf *readtable* *save-readtable*)
#<READTABLE {1000000401}>
CL-USER> [DATE :YEAR 2010 :MONTH 12 :DAY 25]
                                                    ;; RÉPONSE G
```

Le code présenté Figure 4, page 6 permet de charger dans la variable `*persons*` la liste des personnes sauvées dans un fichier texte au format indiqué précédemment. Le fichier texte `data.txt` dont le contenu est montré Figure 6, page 6 est un exemple d'un tel fichier. Exemples :

```
CL-USER> *persons*
NIL
CL-USER> (load-persons "data.txt")
("Charlie Chaplin":1889/04/16 "Marylin Monroe":1926/06/01
 "Marlene Dietrich":1901/12/27)
CL-USER> *persons*
("Charlie Chaplin":1889/04/16 "Marylin Monroe":1926/06/01
 "Marlene Dietrich":1901/12/27)
```

Le code présenté Figure 5, page 6 permet de sauver la liste des personnes contenues dans la variable `*persons*` dans un fichier. Exemples :

```
CL-USER> (pop *persons*)
"Charlie Chaplin":1889/04/16
CL-USER> *person*
"Oscar":2010/12/25
CL-USER> (push *person* *persons*)
("Oscar":2010/12/25 "Marylin Monroe":1926/06/01 "Marlene Dietrich":1901/12/27)
CL-USER> (save-persons "newdata.txt")
NIL
```

Le contenu du fichier `newdata.txt` est alors celui montré Figure 7, page 6.

4. Compléter l'implémentation de la fonction `read-data-file (file)`.
5. Compléter l'implémentation de la fonction `write-data-stream (stream)`.
6. Compléter l'implémentation de la fonction `save-persons (file)`.

FIN

Annexe

```
(defvar *persons* nil)
(defvar *person* nil)
(defvar *date*)

(defclass name-mixin ()
  ((name :initarg :name :reader name)))

(defclass person (name-mixin)
  ((birth-date :initarg :birth-date :reader birth-date)))

(defmethod print-object ((person person) stream)
  (with-slots (name birth-date) person
      (format stream "~S:~A" name birth-date)))
(defclass date ()
  ((year :initarg :year :reader year)
   (month :initarg :month :reader month)
   (day :initarg :day :reader day)))

(defmethod print-object ((date date) stream)
  (with-slots (year month day) date
    (format stream "~A/~2,'0D/~2,'0D" year month day)))
```

FIG. 1 – Structures de données

```
(defparameter *my-readtable* (copy-readtable))

(defun my-read-object (stream char)
  (declare (ignore char))
  (apply #'make-instance (read-delimited-list #\] stream t)))

(set-macro-character #\[ #'my-read-object nil *my-readtable*)
(set-syntax-from-char #\] #\) *my-readtable*)
```

FIG. 2 – Reader programming

```
(defparameter *print-for-file* nil)

(defgeneric save-info (object)
  (:method-combination append :most-specific-last))

(defmacro define-save-info (type &body save-info)
  `(progn
     (defmethod print-object :around ((obj ,type) stream)
       (if *print-for-file*
           (my-print-object obj stream)
           (call-next-method)))

     (defmethod save-info append ((obj ,type))
       ',save-info)))

(defun my-print-object (obj stream)
  (format stream "[~s" (class-name (class-of obj)))
  (loop for info in (save-info obj)
        do (format stream " ~s ~W" (car info)
                   (funcall (cadr info) obj)))
  (format stream "]"))

(define-save-info date
  (:year year) (:month month) (:day day))

(define-save-info person
  (:name name)
  (:birth-date birth-date))
```

FIG. 3 – Printer programming

```
(defun read-data-stream (stream)
  (let ((*read-eval* nil)
        (*readtable* *my-readtable*))
    (loop
      for person = (read stream nil nil)
      while person
      collect person)))

(defun read-data-file (file)
  ...)

(defun load-persons (file)
  (setf *persons* (read-data-file file)))
```

FIG. 4 – Entrée des données

```
(defun write-data-stream (persons stream)
  ...)

(defun write-data-file (persons file)
  (with-open-file (stream file :direction :output :if-exists :supersede)
    (write-data-stream persons stream)))

(defun save-persons (file)
  ...)
```

FIG. 5 – Sortie des données

```
[PERSON :NAME "Charlie Chaplin" :BIRTH-DATE [DATE :YEAR 1889 :MONTH 4 :DAY 16]]
[PERSON :NAME "Marylin Monroe" :BIRTH-DATE [DATE :YEAR 1926 :MONTH 6 :DAY 1]]
[PERSON :NAME "Marlene Dietrich" :BIRTH-DATE [DATE :YEAR 1901 :MONTH 12 :DAY 27]]
```

FIG. 6 – Fichier contenant la description des personnes

```
[PERSON :NAME "Oscar" :BIRTH-DATE [DATE :YEAR 2010 :MONTH 12 :DAY 25]]
[PERSON :NAME "Marylin Monroe" :BIRTH-DATE [DATE :YEAR 1926 :MONTH 6 :DAY 1]]
[PERSON :NAME "Marlene Dietrich" :BIRTH-DATE [DATE :YEAR 1901 :MONTH 12 :DAY 27]]
```

FIG. 7 – Fichier contenant la description des personnes