Université Bordeaux 1 Licence S&TS Mention Informatique S6 2010/2011

UE INF356 **Projet de Programmation 3**

Devoir surveillé

Tous documents autorisés. Mercredi 16 Mars 2011
Durée : 1h20.
Le barème est donné à titre **indicatif**.

Exercice 1 *(2pts)*

L'archive d'un système Lisp nommé `io` et décrit par le fichier `io.asd` se trouve à l'url `http://dept-info.labri.fr/~idurand/io.tgz`. On dispose du système `ASDF` et du système `asdf-install`.

1. Donner la liste des opérations permettant d'installer `io` localement sur son propre compte.
2. Quel est l'effet de ces opérations sur le système de fichier ?

Exercice 2 *(4pts)*

On veut pourvoir entrer des expressions en notation postfixée. Ces expressions seront entourées de crochets (`[ ]`). On voudrait donc pouvoir écrire :

```
CL-USER> [1 2 [3 4 *] +]
15
```

au lieu de

```
CL-USER> (+ 1 2 (* 3 4))
15
```

Proposer une solution utilisant les *read-macros*.

Exercice 3 *(14pts)*

Dans le code donné Figure 1, on représente un noeud d'un graphe (`node`) par un numéro et la liste de ses arcs sortants. On représente un arc (`arc`) d'un graphe orienté par les deux noeuds qu'il relie. On représente un graphe orienté (`graph`) par la liste de ses noeuds.

1. Compléter la fonction `make-node`.
2. Compléter la classe `graph`.
3. Compléter le scénario suivant :

```
CL-USER> (defparameter *n1* (make-node 1))
                                                    ;; Réponse
CL-USER> *n1*
                                                    ;; Réponse
CL-USER> (defparameter *n2* (make-node 2))
                                                    ;; Réponse
CL-USER> *n2*
                                                    ;; Réponse
```

```
CL-USER> (defparameter *arc1* (make-arc *n1* *n2*))
                                                    ;; Réponse
CL-USER> *arc1*
                                                    ;; Réponse
```

Pour sauver les structures de données, on utilise le code de la figure 2.
La fonction `print-io-object` permet d'écrire les objets dans un format facile à relire à l'aide d'une read-macro.
Étant donné un objet, La liste des créneaux à sauvegarder est donnée par un appel à `save-info`.
Les créneaux à sauvegarder sont définis grâce à la macro `define-save-info`.

```
(define-save-info node
  (:num num)
  (:out-arcs node-out-arcs))

(define-save-info arc
  (:origin origin)
  (:extremity extremity))
```

Compléter le scénario suivant :

4.
```
CL-USER> (save-info *n1*)
                                                    ;; Réponse
CL-USER> (save-info *arc1*)
                                                    ;; Réponse
CL-USER> (write-object-to-stream t *n1*)
                                                    ;; Réponse

CL-USER> (write-object-to-stream t *arc1*)
                                                    ;; Réponse
```

Pour améliorer l'efficacité des algorithmes, on décide d'avoir une représentation redondante. En plus d'être représenté dans les noeuds, un arc du graphe sera aussi mémorisé dans une liste d'arcs associée au graphe.

5. Modifier la classe `graph` en conséquence.
6. Définir les créneaux à sauvegarder pour les objets de type `graph`.

Quand un arc est ajouté au graphe par l'opération `graph-add-arc`, il faut maintenant, en plus, ajouter l'arc à la liste des arcs du graphe.

7. Proposer une solution qui ne modifie pas le code existant.

```
(defclass node ()
  ((num :initarg :num :accessor num :initform nil)
   (out-arcs
    :initform nil
    :initarg :out-arcs
    :accessor node-out-arcs)))

(defmethod print-object ((node node) stream)
  (format stream "[~A]" (num node)))

(defun make-node (num) ...)

(defclass arc ()
  ((origin :initarg :origin :reader origin)
   (extremity :initarg :extremity :reader extremity))
  (:documentation "an arc of an oriented graph"))

(defmethod print-object ((arc arc) stream)
  (format stream "~A->~A" (origin arc) (extremity arc)))

(defun make-arc (origin extremity)
  (make-instance
   'arc
   :origin origin
   :extremity extremity))

(defclass graph () ...)

(defun make-graph ()
  (make-instance 'graph))

(defmethod node-add-arc ((arc arc))
  (let ((node (origin arc)))
    (setf (node-out-arcs node)
          (adjoin arc (node-out-arcs node) :test #'equalp))))

(defmethod graph-add-arc ((origin node) (extremity node) (g graph))
  (node-add-arc (make-arc origin extremity)))
```

FIGURE 1 – Représentation d'un graphe orienté

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

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

(defun print-io-object (obj stream)
  (format stream "[~A " (class-name (class-of obj)))
  (loop for info in (save-info obj)
        do (format stream
                   ":~A ~A "
                   (first info)
                   (funcall (second info) obj)))
  (format stream "]"))

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

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

(defun write-object-to-stream (stream object)
  (let ((*print-circle* t)
        (*print-for-io* t))
    (format stream "~A" object)))
```

FIGURE 2 – Code pour sauver les objets