ANNÉE UNIVERSITAIRE 2010/2011
2IÈME SESSION DE PRINTEMPS

Parcours : CSB6
Code UE : INF356
Épreuve : Projet de Programmation 3
Date : Jeudi 23 juin 2011
Heure : 8h30
Durée : 1h30
Documents : autorisés
Épreuve de Mme Irène Durand

Le barème est donné à titre **indicatif**. Le sujet comporte 3 pages dont une annexe.

Exercice 1 *(6pts)*

Soit la fonction `map-if` prenant deux paramètres obligatoires : une liste `l` et une fonction `f`, ainsi qu'un paramètre optionnel de type *mot-clé* `test`. La fonction `map-if` retourne la liste des `f(e)` pour tous les `e` vérifiant `test`. Exemples :

```
CL-USER> (map-if (lambda (x) (* x x)) '(1 3 5))
(1 9 25)
CL-USER> (map-if (lambda (x) (- x)) '(1 a 3 b 5) :test #'numberp)
(-1 -3 -5)
CL-USER> (map-if #'identity '(1 2 3 4 5) :test #'evenp)
(2 4)
CL-USER>
```

1. Écrire une version récursive de `map-if`.
2. Écrire une version itérative de `map-if`.

Exercice 2 *(8pts)*

Dans le code donné dans la figure 2 de l'annexe, 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.
On veut pouvoir marquer aussi bien les arcs que les noeuds.

1. Définir une classe de type *mixin* formalisant le fait qu'un objet peut être marqué.
2. Définir et implémenter l'opération `mark-object (object)` qui correspond au marquage d'un objet.
3. Définir et implémenter l'opération `unmark-object (object)` qui correspond au démarquage d'un objet.
4. Modifier les classes `node` et `arc` de manière à ce qu'on puisse marquer les noeuds et les arcs.
5. Définir et implémenter l'opération `all-nodes-marked-p (graph)` qui retourne vrai si tous les noeuds du graphe `graph` sont marqués et faux sinon.

6. Définir et implémenter l'opération (destructive) `mark-from (node graph)` qui étant donné un graphe `graph` ayant aucun sommet marqué, marque le noeud `node` ainsi que tous les noeuds accessibles à partir de ce noeud.

Exercice 3 *(6pts)*

1. En supposant, qu'on a exécuté le code de la figure 1 donnée en annexe, compléter le scénario suivant :

```
CL-USER> (read)
(+ 1 2)
                        ;; Réponse 1
CL-USER> (+ 1 2)
                        ;; Réponse 2
CL-USER> (read)
[3 * 4]
                        ;; Réponse 3
CL-USER> [3 * 4]
                        ;; Réponse 4
CL-USER> (read)
[3 * [4 + 5]]
                        ;; Réponse 5
CL-USER> [3 * [4 + 5]]
                        ;; Réponse 6
CL-USER>
```

2. Écrire une read-macro telle que quand on tape `#?n` où `n` est un entier, la valeur retournée soit un tableau à une dimension de taille `n` contenant les éléments `0, 1, ..., n-1`. Exemples :

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

Annexe

```
(set-syntax-from-char #\] #\))

(set-macro-character
  #\[
  (lambda (stream char)
    (declare (ignore char))
    (let* ((l (read-delimited-list #\] stream t))
   (last (last l)))
      (append last (nbutlast l)))))
```

FIGURE 1 – Read-macro

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

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

(defclass graph ()
  ((nodes
    :initarg :nodes
    :initform '()
    :accessor graph-nodes)))
```

FIGURE 2 – Représentation d'un graphe orienté

FIN