Université Bordeaux 1 Licence S&TS Mention Informatique S6 2008/2009

UE INF356 **Projet de Programmation 3**

Devoir surveillé

Tous documents autorisés. Mardi 17 Mars 2009
Durée : 1h20.
Le barème est donné à titre **indicatif**.

Exercice 1 *(2pts)*

Un système `Lisp` se trouve à l'url `http://martin-loetzsch.de/gtfl/gtfl.tar.gz`.
Décrivez une méthode permettant d'installer ce système en local sur votre ordinateur.

Exercice 2 *(8pts)*

On se propose d'écrire un ensemble de classes représentant des familles de graphes en listant leur propriétés.
Un graphe est définit par un couple $G = (A, N)$
– N est un ensemble de noeuds
– A est un ensemble d'arcs, $A \subseteq N \times N$.
Un graphe se résume donc à une relation binaire sur l'ensemble N. On souhaite classifier les graphes en fonction de certaines de leur propriétés. Par exemple un graphe *simple* est le graphe d'une relation irréflexive, un graphe *non orienté* est le graphe d'une relation symétrique, ...

```
(defclass graphe ()
  ((%noeuds :initarg :noeuds :initform nil :accessor noeuds :type list)
   (%arcs :initarg :arcs :initform nil :accessor arcs :type list)))

(defclass graphe-simple (graphe) ())
(defclass graphe-non-oriente (graphe) ())
(defclass graphe-sans-cycle (graphe) ())
(defclass arbre (graphe-simple graphe-non-oriente graphe-sans-cycle) ())

(defclass arbre-pointe (arbre)
  ((%racine :initarg :racine :initform nil :accessor racine)))
```

1. Dessiner la hiérarchie des classes.
2. L'opération `proprietes` appliquée à un graphe retourne la liste de toutes les propriétés de ce graphe exprimées sous forme de chaîne de caractères. Implémenter l'opération `proprietes` par combinaison de méthodes. Exemple :

   ```
   (defparameter *mon-arbre*
     (make-instance 'arbre-pointe
                    :noeuds '(n1 n2 n3)
                    :arcs '((n1 n2) (n1 n3))
                    :racine 'n1))
   ```

   L'expression `(proprietes *mon-arbre*)` doit afficher quelque chose comme ceci :

   ```
   ("aucune chaine simple d'un noeud vers lui-meme" "relation symetrique"
    "aucun noeud relie a lui-meme"
    "chaine unique entre tout couple de noeuds distincts"
    "existence noeud distingue")
   ```

3. Chaque fois qu'un graphe est créé, on souhaite vérifier que la liste des noeuds reliés par les arcs est bien incluse dans la liste des noeuds du graphe. De même quand un `arbre-pointe` est créé, on souhaite vérifier que sa `racine` se trouve bien dans la liste des sommets.
   Proposer une solution permettant de faire ces vérifications systématiquement et de provoquer une erreur le cas échéant.
   N.B. On pourra supposer que les noeuds d'un graphe se comparent avec le fonction `eq`.

Exercice 3 *(5pts)*

Écrire une fonction qui prend en paramètre : un encodage source, un encodage cible, un nom de fichier et optionellement un booléen (faux par défaut) et qui convertit le fichier encodé dans l'encodage source en un fichier encodé dans l'encodage cible. Si le booléen est à vrai, le fichier source doit être remplacé par sa version convertie ; s'il est à faux, le nom du fichier cible sera construit en suffixant le nom du fichier source par le nom de l'encodage précédé par le caractère `_`.
La fonction retournera le nom du fichier cible. Exemples :

```
CL-USER> (convert :latin-1 :utf-8 "fichier1.txt")
"fichier1.txt_UTF-8"
CL-USER> (convert :latin-1 :utf-8 "fichier2.txt" t)
"fichier2.txt"
```

N.B. On rappelle que la fonction `format` utilisée avec comme premier paramètre `NIL` permet de construire des chaînes de caractères formatées.

Exercice 4 *(5pts)*

1. Soit la séquence suivante :

```
CL-USER> (set-syntax-from-char #\] #\))
T
CL-USER> (set-dispatch-macro-character
  #\# #\[
  (lambda (stream char1 char2)
    (declare (ignore char1 char2))
    (mapcar
     (lambda (n)
       (make-list n :initial-element n))
     (read-delimited-list #\] stream))))

#<FUNCTION (LAMBDA (STREAM CHAR1 CHAR2)) {B60465D}>
CL-USER> (read)
#[1 2 3]
```

   Quel sera le résultat affiché une fois que l'utilisateur aura tapé sur la touche `Entrée` ?
2. Donner une expression qui après évaluation permet d'obtenir le comportement suivant : quand on tape le caractère `&` suivi d'un entier `n`, c'est comme si on avait tapé une liste contenant `n` zéros précédée d'une *quote*. Ainsi `'(0 0 0 0 0)` peut être remplacé par `&5`. Exemple :

```
CL-USER> '(0 0 0 0)
(0 0 0 0)
CL-USER> &4
(0 0 0 0)
CL-USER> &0
NIL
```

FIN