université de BORDEAUX Collège Sciences et Technologies DEVUIP Service Scolarité	ANNÉE UNIVERSITAIRE 2015/2016 2ÈME SESSION D'AUTOMNE **Parcours :** IN501 et IM500 **Code UE :** IN5011 **Épreuve :** Programmation 3 **Date :** Jeudi 17 décembre 2015 **Heure :** 8h30 **Durée :** 1h30 Documents : autorisés Épreuve de Mme Irène Durand	Département D L Licence

Le barème est donné à titre **indicatif**.
Le sujet comporte 4 pages plus une page d'annexe.

Exercice 1 *(7pts)*

Soit la fonction `intersection-length (s1 s2)` qui s'applique à `s1`, `s2`, deux ensembles d'atomes[1], représentés par des listes sans doublons, et qui retourne le nombre d'éléments de l'intersection de `s1` et de `s2`. Exemples :

```
CL-USER> (intersection-length '(3 2 5 4 1) '(2 5 3 6))
3
CL-USER> (intersection-length '() '(2 5 3 6))
0
```

1. Donner une implémentation de la fonction `intersection-length` en utilisant les fonctions prédéfinies `intersection` et `list-length` (ou `length`).

Le fait de construire l'intersection est inutilement coûteux en allocation mémoire. On souhaite donc des implémentations qui ne construisent pas explicitement l'intersection.

2. Écrire une version itérative.
3. Écrire une version récursive.
4. La version récursive donnée est-elle récursive terminale ?

On souhaite manipuler des ensembles contenant d'autres objets que des atomes.

5. Rajouter un paramètre mot-clé `test` au prototype de la fonction et adapter **une** des implémentations précédentes (au choix) en conséquence.

1. On considère uniquement des atomes comparables avec `eql`.

On considère des graphes sans boucle à n sommets numérotés de 0 à $n-1$. On représente un arc d'un graphe orienté par un couple (i, j) et une arête d'un graphe non orienté par une paire (i, j) telle que $i < j$. Un ensemble d'arcs (ou d'arêtes) sera représenté par une liste de couples. Chaque couple sera représenté par une liste de deux entiers.
Par exemple la liste d'arcs `((0 1) (1 2) (2 3) (3 0))` représentera le graphe de la figure 1 et la liste d'arêtes `((0 1) (0 3) (1 2) (1 3) (2 3))` le graphe non orienté de la figure 2.

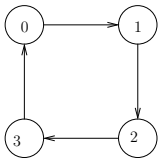

FIGURE 1 – C_4 : Circuit orienté

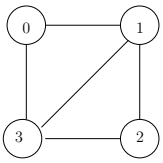

FIGURE 2 – Graphe non orienté

Exercice 2 *(3pts)*

Soit la fonction `arcs-to-edges (arcs)` qui s'applique à un ensemble d'arcs et qui retourne l'ensemble (sous forme de liste) des arêtes obtenues en supprimant l'orientation. Exemple :

```
CL-USER> (arcs-to-edges '((0 1) (1 2) (2 1) (3 2)))
((0 1) (1 2) (2 3))
```

6. Implémenter la fonction `arcs-to-edges`.

On souhaite décrire les arcs de certaines familles de graphes par une expression arithmétique `expression` utilisant uniquement les variables
- `N` (nombre de sommets du graphe),
- `I` (origine d'un arc),
- `J` (extrémité d'un arc),

des nombres entiers et des opérateurs arithmétiques et logiques [2].

Les arcs (i, j) du graphe sont ceux tels que l'expression `expression` est vraie.

Par exemple,

2. `not, and, or, +, -, *, =, <, ...`

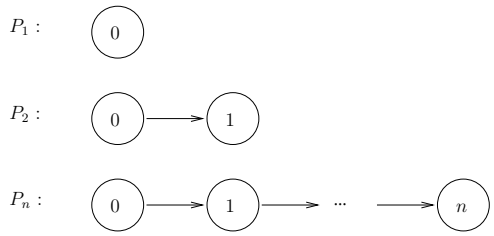

FIGURE 3 – Graphes P_n

— les graphes stables (sans arête) peuvent être décrits par l'expression `NIL` ;
— un graphe orienté P_n à n sommets de la figure 3, peut-être décrit par l'expression `(= j (1+ i))` ;
— un circuit orienté C_n (voir le C_4 de la figure 1) peut-être décrit par l'expression :
`(or (= j (1+ i)) (and (= i (1- n)) (= j 0)))`
qui indique que les arcs sont de la forme $(i, i+1)$ ou $(n-1, 0)$.

Soit la macro `expression-to-arcs (nb-nodes expression)`

```
(defmacro expression-to-arcs (nb-nodes expression)
  (setq expression (subst nb-nodes 'n expression))
  `(let ((l ()))
     (dotimes (i ,nb-nodes (nreverse l))
       (dotimes (j ,nb-nodes)
         (when (and (/= i j) ,expression)
           (push (list i j) l))))))
```

qui prend en paramètre `nb-nodes`, le nombre de sommets du graphe et le paramètre `expression` qui doit s'évaluer en une expression booléenne au format décrit précédement.

— Remarquer que les variables `I` et `J` sont volontairement **capturées** dans le corps de la macro.
— La fonction prédéfinie `subst (new old tree)` remplace `old` par `new` dans `tree`. Exemples :

```
CL-USER> (subst 4 'n '(or (= i n) (= j (1+ n))))
(OR (= I 4) (= J (1+ 4)))
CL-USER> (subst 4 'x  '(* (+ 2 x) (- 3 (log x))))
(* (+ 2 4) (- 3 (LOG 4)))
```

Exercice 3 *(5pts)*

7. Évaluer l'expression `(macroexpand-1 '(expression-to-arcs 5 (= j (1+ i))))`.
8. Évaluer l'expression `(expression-to-arcs 5 (= j (1+ i)))`.
9. Écrire un appel à la macro `expression-to-arcs` qui retourne les arcs d'un graphe orienté complet à 3 sommets.

```
CL-USER> (expression-to-arcs ... ...)
((0 1) (0 2) (1 0) (1 2) (2 0) (2 1))
```

Exercice 4 *(5pts)*

Soit le code donné en Annexe page 5.

10. Dessiner la hiérarchie des classes.
11. Compléter l'implémentation de l'opération `print-object` de sorte que le solde (`balance` en anglais) soit affiché comme indiqué dans l'exemple qui suit.

```
CL-USER> *account1*
#<ACCOUNT {AFC7761}> balance: 10000.00
```

12. Implémenter l'opération `transfer` sans vérification sur le solde des comptes.

```
CL-USER> *account2*
#<ACCOUNT {B01AAD9}> balance: 500.00
CL-USER> (transfer 300 *account1* *account2*)
800.0
CL-USER> *account1*
#<ACCOUNT {AF7BEA1}> balance: 9700.00
CL-USER> *account2*
#<ACCOUNT {B01AAD9}> balance: 800.00
```

Certains comptes sont *plafonnés* (`blocked` en anglais), c'est à dire que leur solde ne peut pas dépasser un certain plafond (`roof` en anglais). Le fait d'être plafonné est capturé par la classe abstraite `blocked-mixin` dont le créneau `roof` mémorise la valeur du plafond.

13. En utilisant une méthode `:before` sur l'opération `transfer` ainsi que la macro `assert`, faire en sorte qu'une erreur soit signalée lorsque le plafond est dépassé.

```
CL-USER> (transfer 4500 *account1* *blocked-account*)
The assertion
(< #1=(+ (BALANCE ACCOUNT2) MONTANT) #2=(ROOF ACCOUNT2)) failed
with #1# = 5500.0, #2# = 2000.0.
   [Condition of type SIMPLE-ERROR]
...
CL-USER> (transfer 500 *account1* *blocked-account*)
1500.0
```

FIN

Annexe

```
(defclass account ()
  ((balance :initform 0.0 :initarg :balance :accessor balance)))

(defclass blocked-mixin ()
  ((roof :reader roof :initarg :roof :initform 100)))

(defclass blocked-account (account blocked-mixin) ())

(defgeneric transfer (montant account1 account2)
  (:documentation "transfere MONTANT du ACCOUNT1 vers ACCOUNT2"))

(defparameter *account1* (make-instance 'account :balance 10000.0))
(defparameter *account2* (make-instance 'account :balance 500.0))
(defparameter *blocked-account*
  (make-instance 'blocked-account :balance 1000.0 :roof 2000.0))
```