# UV PROGRAMMATION DÉCLARATIVE

Examen de programmation fonctionnelle

Durée 2h, documents autorisés, livres interdits

1. On suppose que les expressions suivantes sont données lors d'un début de session. Donner les résultats de leur évaluation.

```
(defparameter a 0)

(let ((a 2)
      (b 3)
      (c a))
  (- (* b b)
     (* 4 a c)))

(let ((b 0))
  (defun f(m)
    (+ a b m)))

(f 1)

(let ((a 2)
      (b 1))
  (defun plus(m)
     (f m)))

(plus 1)
```

2. Ecrire une macro, de nom `increment`, admettant un paramètre et réalisant l'incrémentation de son paramètre.

```
* (defparameter x 0)
X
* (increment x)
1
* x
1
```

3. Un générateur est une fonction utilisant son environnement comme un état pour réaliser un calcul, et modifiant son état à chaque appel pour préparer le calcul à réaliser lors de l'appel suivant. Par exemple la fonction `int` décrite ci-dessous est un générateur d'entiers, incrémentant à chaque appel une variable de son environnement afin de produire l'entier suivant.

```
* (int)
1
* (int)
```

```
2
* (int)
3
```

Écrire la fonction `int` réalisant un tel calcul. Écrire ensuite une macro `make-generator` permettant de construire un générateur dont l'environnement est réduit à une variable. Elle admettra en paramètres le nom du générateur, le nom de la variable de l'environnement, sa valeur initiale, et la liste des opérations à éxecuter.

```
* (make-generator int i 0 (setf i (+ i 1)))
INT
* (int)
1
* (int)
2
* (make-generator int2 i 0 (setf i (+ i 1)))
INT2
* (int2)
1
```

4. Soit $A =< Q, q_0, F, \Sigma, \longrightarrow >$ un automate déterministe, avec $Q$ l'ensemble des états, $q_0$ l'état initial, $F$ l'ensemble des états finaux, $\Sigma$ l'alphabet d'entrée, $\longrightarrow: Q \times \Sigma \longrightarrow Q$ la fonction de transition.

   On souhaite représenter un tel automate par un ensemble d'états représentés eux-mêmes chacun par une fonction. Un état $e$ est une fonction admettant une liste $l$ en paramètre, représentant un mot. Soit $a$ le premier élément de la liste $l$, le résultat de l'application $e(l)$ est alors le résultat de l'application $e'(l')$, avec $e \xrightarrow{a} e'$ et $l'$ est la liste $l$ privée de son premier élément. Le résultat de l'application d'un état à un mot est égal à nil si le mot n'est pas reconnu, à n'importe quelle autre valeur sinon.

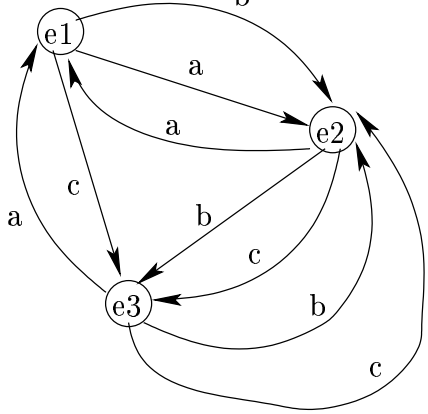
   Écrire les trois fonctions représentant les trois états de l'automate corrrespondant à la figure ci-après. Cet automate admet pour état initial $e_1$ et pour état final $e_3$. Ecrire une fonction `reconnu` permettant de savoir si un mot est reconnu ou non par l'automate. Par exemple:

```
* (reconnu '(a a b b) i)
oui
* (reconnu '(a a b) i)
nil
```

   où `i` est l'état initial de l'automate.

5. On souhaite représenter des polynômes par une liste d'associations et écrire des fonctions de gestion et de calcul. On écrira les fonctions suivantes sans utiliser d'effets de bords

   - make-polynome : construction d'un polynôme à partir d'une liste de coefficients et d'une liste de degrés ordonnée par ordre décroissant.
   - degree : extraction du degré d'un polynôme.
   - degree-list : extraction de la liste des degrés d'un polynôme.




- coeff-list : extraction de la liste des coefficients d'un polynôme.
- nth-degree-coefficient : extraction du coefficient correspondant au terme de degré n.
- +polynome : somme de deux polynômes.

Exemple de session :

```
*(defparameter p1 (make-polynome '(3 2 1) '(4 3 1)))
*(defparameter p2 (make-polynome '(0 2 1 3) '(5 4 1 0)))
*(degree p1)
4
*(+polynome p1 p2)
((5 . 4) (2 . 3)  (2 . 1) (3 . 0))
```