Exercices 5

Programmation itérative

1) Séquences

(begin e1 e2 … en)

Les expressions *e1 … en* sont évaluées, begin retourne la valeur de la dernière… C'est pour exécuter une séquence et avoir les effets de bord.

Exemple :

```
(begin
 (define x 4)
 (display x)
(define x (+ x 1))
(display x) )
```

Le résultat est :

```
45
```

2) Iterations

```
(do
    ((var1 base1 exp1) (var2 base2 exp2) …)
  ((test? ...) final-exp)
  side-effect-statements ...)
```

Les variables *var1, var2, ...* sont créées avec des valeurs initiales *base1, base2, ...* Puis le teste *(test? ...)* est evalué. S'il est vrai, f*inal-exp* est retourné. Sinon, *var1, var2, ...* sont redéfinie par les nouvelles valeurs *exp1, exp2, ...* et on réitère la procédure.

Exemple :

```
 (do
  (( x 1 (+ x 1)))
  ((> x 10) (display x) )
 (display x) )
```

Le résultat est : 1234567891011

3) Itérations sur les listes

```
(map (lambda (x) (fonction x))  liste)
```

La fonction est exécutée sur chaque élément de la liste et une nouvelle liste est retournée.
Exemples :

```
(define L (list 1 -2 -5))
(map (lambda (x) (abs x))  L)
```

Le résultat est :

(1 2 5)

```
(map (lambda (x) (* x 2)) '(1 2 3 4 5 6))
```

Le résultat est : (2 4 6 8 10 12)

(for-each (lambda (x) (fonction x)) liste)
La fonction est exécutée sur chaque élément de la liste.

Exemples :

```
(for-each (lambda (x) (+ x 1))  '(1 2 3 4 5 6))
```

Le résultat est : (les incrémentations sont exécutées mais pas affichées)

```
(for-each (lambda (x) (display (+ x 1))(display (* x 2)) (newline))
                                                    '(1 2 3 4 5 6))
Le résultat est :
22
34
46
58
610
712
```

Exercices

Q1) Créez la fonction (fibonacci2 n) qui calcule les valeurs d'une façon itérative.

Q2) Regroupez l'exécution de (fibonnaci n) et de (fibonacci2 n) en une seul fonction (paquet n).

Q3) Ecrivez une fonction (negatives L) qui affiche les éléments d'une liste qui sont inférieur à zéro.

Q3-bis) Ecrivez une fonction (elements-petits L) qui affiche les éléments d'une liste qui sont inférieur à son premier élément.

Q4) Ecrivez une fonction (filtre L n) qui va parcourir une liste d'entiers et qui change les éléments qui sont inférieurs à la valeur de n contre n.

Q5) Un nombre premier est un nombre (positif) qui n'est divisible que par 1 et par lui-même.
Donnez la formulation itérative (et pas récursive) du prédicat (premier? N) qui retourne #t si N est un nombre premier.

Q6) Écrivez la fonction *(pos L n)* qui retourne l'élément d'indice n de la liste L en utilisant une fonction itérative (et pas récursive).

Q7) On va réaliser un additionneur binaire. Pour commencer, supposons deux bits : b1 et b2. Écrivez les fonctions suivantes :
fonction *(s b1 b2)* qui retourne la somme des deux bits sur la position donnée de l'additionneur (attention, 1 + 1 → 0)
fonction *(r b1 b2)* qui retourne le retenue des deux bits, pour 1 + 1 → 1.

Q8) Écrivez la fonction *(plus k1 k2)* qui additionne deux listes de bits (k1 et k2) en utilisant une fonction itérative (et pas récursive). Supposez que les listes ont la même longueur et le résultat est aussi représenté sur cette longueur.

Q9) Écrivez la fonction *(retenue k1 k2)* qui retourne la retenue de l'addition précédente (la retenue de l'additionneur de longueur donnée).

Q10) Écrivez la fonction récursive *(sommeL l1 l2)* qui retourne la liste des sommes des éléments des listes l1 et l2.

Q11) Écrivez maintenant la fonction itérative

Q12) Écrivez la fonction (produit l1 l2) qui retourne la somme des produits des éléments dans les listes données (la valeur retourné est : (pos l1 1) * (pos l2 1) + … + (pos l1 n) * (pos l2 n))