MIAS 1 et MASS 1 – Programmation fonctionnelle

TP n° 4: corrigé

```
; Exercice 1
; variable d'induction : n
; condition d'arret : n=0
; puiss : Reel, Entier --> Entier
;           x ,   n    |->  "indefini"       si x=n=0
;                           1                si x<>0 et n=0
;                           1/puiss(x, -n)   si n<0
;                           x*puiss(x, n-1)  si n>0

(define puiss
  (lambda (x n)
    (cond ((and (= n 0) (= x 0)) "non defini")
          ((= n 0) 1)
          ((< n 0) (/ 1 (puiss x (- n))))
          (else (* x (puiss x (- n 1)))))))


; Exercice 2
; variable d'induction : n
; condition d'arret : n=0
; suite : Reel, Reel, Reel, Entier --> Reel
;          a  ,  b  ,  u0 ,   n    |-> u0     si n=0
;                                     a*suite(a,b,u0,n-1)+b   si n>0

(define suite
  (lambda (a b u0 n)
    (if (= n 0)
        u0
        (+ (* a (suite a b u0 (- n 1))) b))))

; serie : Reel, Reel, Reel, Entier --> Reel
;          a  ,  b  ,  u0 ,   n    |-> u0     si n=0
;                                     suite(a,b,u0,n) + serie(a,b,u0,n-1)  si n>0

(define serie
  (lambda (a b u0 n)
    (if (= 0 n)
        u0
        (+ (suite a b u0 n)
           (serie a b u0 (- n 1))))))


; Exercice 3
; variable d'induction : la liste l
; condition d'arret : l est vide
; fois2 : Liste --> Liste
;           l   |-> l  si l est la liste vide
;                   ( 2*x . fois2(l1) )   si l=( x . l1 )

(define fois2
  (lambda (l)
    (if (null? l)
        l
        (cons (* 2 (car l)) (fois2 (cdr l))))))
```

```
; Exercice 4
; variable d'induction : la liste l
; conditions d'arret : l est vide ou son premier element est celui recherche
; appartient : Objet, Liste --> Booleen
;                o  ,   l   |-> faux  si l est la liste vide
;                               vrai  si l=( o . l1 )
;                               appartient(o, l1) sinon, avec l=( x . l1 )

(define appartient
  (lambda (o l)
    (if (null? l)
        #f
        (or (equal? o (car l))
            (appartient o (cdr l))))))


; Exercice 5
; variable d'induction : la liste l
; condition d'arret : l est vide
; compter : Liste --> Entier
;             l   |-> 0 si l est vide
;                     1+compter(l1) si l=(x . l1)

(define compter
  (lambda (l)
    (if (null? l)
        0
        (+ 1 (compter (cdr l))))))


; Exercice 6
; variable d'induction : l'entier n
; conditions d'arret : l est vide ou n=1
; nieme : Entier+ , Liste --> Objet
;           n     ,   l   |-> "indefini"      si l=listevide
;                             o              si n=1, avec l=(o . l1)
;                             nieme(n-1, l1) si n>1, avec l=(o . l1)

(define nieme
  (lambda (n l)
    (cond ((null? l) "indefini")
          ((= n 1) (car l))
          (else (nieme (- n 1) (cdr l))))))


; Exercice 7
; variable d'induction : la liste l
; condition d'arret : l est vide
; ajoutefin : Objet, Liste --> Liste
;               o  ,   l   |-> (o)                    si l est vide
;                             (x . ajoutefin(o,l1))  si l=(x . l1)

(define ajoutefin
  (lambda (o l)
    (if (null? l)
        (cons o l)
        (cons (car l) (ajoutefin o (cdr l))))))
```

```
; Exercice 8
; variable d'induction : la liste l2
; condition d'arret : l2 est vide
; concat : Liste, Liste --> Liste
;            l1 ,  l2   |-> l1 si l2 est vide
;                          concat( ajoutefin(x, l1), l3 )  si l2=(x . l3)

(define concat
  (lambda (l1 l2)
    (if (null? l2)
        l1
        (concat (ajoutefin (car l2) l1)
                (cdr l2)))))


; Exercice 9
; variable d'induction ; la liste l
; condition d'arret : l est vide
; supprime : Objet, Liste --> Liste
;              o  ,   l   |-> l si l est vide
;                             supprime(o, l1)        si l=(o . l1)
;                             (x . supprime(o, l1))  si l=(x . l1) et x<>o

(define supprime
  (lambda (o l)
    (cond ((null? l) l)
          ((equal? o (car l)) (supprime o (cdr l)))
          (else (cons (car l)
                      (supprime o (cdr l)))))))


; Exercice 10
; a)
; variable d'induction : la liste E
; condition d'arret : E est vide
; est-ensemble? : Liste --> Booleen
;                   E   |-> vrai si E est vide
;                          faux si l=(x . E1) et x appartient a E1
;                          est-ensemble?(E1) sinon

(define est-ensemble?
  (lambda (E)
    (cond ((null? E) #t)
          ((member (car E) (cdr E)) #f)
          (else (est-ensemble? (cdr E))))))

;On peut aussi ecrire : une liste est un ensemble si elle est vide
; ou bien si son premier element n'est pas dans son reste et son reste est un ensemble
; c'est a dire :

(define est-ens?
  (lambda (E)
    (or (null? E)
        (and (not (member (car E) (cdr E)))
             (est-ens? (cdr E))))))
```

```
; b)
; variable d'induction : la liste E
; condition d'arret : E est vide
; union : Ensemble, Ensemble --> Ensemble
;           E   ,     F     |-> F si E est vide
;                               union(E1,F)        si E=(x . E1) et x appartient a F
;                              (x . union(E1,F))   si E=(x . E1) et x n'appartient pas a F

(define union
  (lambda (E F)
    (cond ((null? E) F)
          ((member (car E) F) (union (cdr E) F))
          (else (cons (car E)
                      (union (cdr E) F))))))

; c)
; variable d'induction : la liste E
; condition d'arret : E est vide ou F est vide
; intersection : Ensemble, Ensemble --> Ensemble
;                   E   ,     F    |-> listevide si E ou F est vide
;                                       (x . intersection(E1,F)) si E=(x . E1) et x appartie
;                                       intersection(E1,F)       si E=(x . E1) et x n'appart

(define intersection
  (lambda (E F)
    (cond ((or (null? E) (null? F)) '())
          ((member (car E) F) (cons (car E)
                                    (intersection (cdr E) F)))
          (else (intersection (cdr E) F)))))
```