Université Lyon 1 - Licence STS - UE LIFAP2
17 mai 2022 - 1h30

Aucun document autorisé – Calculatrices et téléphones interdits

C'est cette feuille qu'il faut rendre. Ne pas l'utiliser comme brouillon.

NOM :

……

PRENOM :

……

Numéro Etudiant :

……

Premier exercice (2 points) - EVAL

Donner les résultats de l'évaluation par l'interpréteur Scheme des expressions suivantes :

- `(cons '(a)(append '(b) (list '(c d) '(e f))))` ➔ .....((a) b (c d) (e f))
- `(cons (append '(a) '(b d)) '(e f))` ➔ .....((a b d) e f)
- `(append (cdr '(a b c)) (list (car '(a b c))))` ➔ .....(b c a)
- `(caddr(caddr (list 'a '(b (c)) (cons '(a (b) c) '(d e)))))` ➔ .....e

Deuxième exercice (4 points) – LISTE PROF

Définir une fonction `encoder-prof`, qui étant donnée une liste quelconque, remplace tous les symboles par un élément passé en paramètre, tous les nombres par un autre élément passé en paramètre, et les autres éléments par un troisième élément passé en paramètre. Cette fonction doit opérer en profondeur.

Ex : `(encoder-prof '(45 s "az" (f "e" 3 (4 t)) y) 'symb 'nb '?)`
➔ `(nb symb ? (symb ? nb (nb symb)) symb)`

```
(define encoder-prof ; -> liste
 (lambda (l s n a) ; liste qcq
  (cond ((null? l) '())
      ((list? (car l)) (cons (encoder-prof (car l) s n a) (encoder-prof (cdr l) s n a)))
      ((symbol? (car l)) (cons s (encoder-prof (cdr l) s n a)))
      ((number? (car l)) (cons n (encoder-prof (cdr l) s n a)))
      (else (cons a (encoder-prof (cdr l) s n a))))))
```

4 points = 1 specif, 0,5 cas arret, 1pt prof, 1,5 cas général

Troisième exercice (5 points) – LISTE AVEC PARAM SUPPLEMENTAIRE

Définir la fonction `ordre` qui, étant donnée une liste d'atomes, renvoie une liste de couples formés par les atomes de la liste initiale et de leur position dans cette liste.

Ex : `(ordre '(m a i s o n))`

➔ `((m 1) (a 2) (i 3) (s 4) (o 5) (n 6))`

```
(define ordre ; > liste de couples
 (lambda (l) ; liste
  (ordre2 l 1)))

(define ordre2
 (lambda (l n)
  (if (null? l)       '()
      (cons (list (car l) n) (ordre2 (cdr l) (+ n 1))))))

5 points = 1 specif, 1 sous-fonction et init, 0,5 cas arret, 2,5 cas général
```

Quatrième exercice (6 points) – AB AVEC LET

Définir une fonction `nb-pairs-impairs`, qui étant donné un arbre d'entiers, construit une liste de 2 entiers : le nombre de valeurs paires et le nombre de valeurs impaires de l'arbre.

Ex : `(nb-pairs-impairs '(5 (2 () (3 () ())) (1 (7 () ()) ())))` ➔ `(1 4)`

```
(define nb-pairs-impairs ; -> liste de 2 entiers
 (lambda (a) ; arbre d'entiers
  (if (vide? a)
      '(0 0)
      (let ((rg (nb-pairs-impairs (fils-g a)))
            (rd (nb-pairs-impairs (fils-d a))))
       (if (even? (valeur a))
           (list (+ 1 (car rg) (car rd)) (+ (cadr rg) (cadr rd)))
           (list (+ (car rg) (car rd)) (+ 1 (cadr rg) (cadr rd))))))))

6 points = 0,5 specif, 1 cas arret, 1pt let, 0,5 test, 3 pts cas général
```

Cinquième exercice (3 points) – MYSTERE LISTE CLASSIQUE

Que fait la fonction `mystere` ? Donnez la spécification de la fonction et au moins un exemple.

```
(define mystere
  (lambda (l)
     (cond
       ((or (null? l) (null? (cdr l))) l)
       ((< (* (car l) (cadr l)) 0)
              (cons (car l) (cons 0 (mystere (cdr l)))))
       (else (cons (car l) (mystere (cdr l)))))))
```

```
(define insere ; -> liste
 (lambda (l) ; l liste de nombres non nuls
   (cond
     ((or (null? l) (null? (cdr l))) l)
     ((< (* (car l) (cadr l)) 0) (cons (car l) (cons 0 (insere (cdr l)))))
     (else (cons (car l) (insere (cdr l)))))))

Desc : insère un zéro entre deux nombres de signes contraires

(insere '(1 2 -10 -6 3 -4)) -> (1 2 0 -10 -6 0 3 0 -4)
3 points = 1 specif, 1 pt ex, 1 point description
```