Paires pointées - listes

Exercice 1 : Evaluation

Evaluer les expressions suivantes :

1. `(cons '(A B C) '(1 2 3))`
2. `(append '(A B C) '((1 2) 3))`
3. `(last '((A 1) (B 2) (C 3)))`
4. `(butlast '((A 1) (B 2) (C 3)))`
5. `(car '((A (B C)) D (E F)))`
6. `(cdr '((A (B C)) D (E F)))`
7. `(caddr'((A (B C)) D (E F)))`
8. `(cons 'NOBODY (cons 'IS '(PERFECT)))`
9. `(list (1+ 2) (1- 5) 6)`
10. `(cdr '(a b))`
11. `(cdr '(a . b))`
12. `'(a . (b . (c . ())))`
13. `'(a . (b . (c . d)))`

Exercice 2 :

Donner les combinaisons de car et cdr nécessaires pour remplacer le signe " ?" dans chacun des cas suivants afin que :

1. (? '(A B C D)) ait pour résultat D
2. (? '((A (B C)) E)) ait pour résultat C
3. (? (((DIEU) ENCORE) UNE)) ait pour résultat DIEU
4. (? (((DIEU) ENCORE) UNE)) ait pour résultat ENCORE

Considérez les listes ci-dessous et écrivez pour chacune d'elles l'expression LISP qui en extrait le symbole FOO.

1. '(foo)
2. '(bar (foo))
3. '((bar foo))
4. '(bar baz foo)
5. '(bar ((baz (foo))))
6. '(((bar (baz (foo)))))

Exercice 3 :

1. Utiliser car, cdr et cons pour inverser une liste à 3 éléments.
2. Ecrire une autre fonction d'inversion, qui utilise trois fonctions auxiliaires : first, second et third, permettant d'inverser une liste à trois éléments.
3. Construire la liste '(1 2 3 4 5) à partir de '(1 2) '((3 4)) et 5.

Exercice 4 : Prédicats binaires

1. Pour tester les deux prédicats binaires eq et equal, commencez d'abord par les formes :

```
(defparameter liste1 '(a b c))
(defparameter liste2 '(b c))
```

2. Puis exécutez ensuite les formes :
```
(eq (cdr liste1) liste2)
(equal (cdr liste1) liste2)
```
3. Quelle est la différence entre =, eq, et equal ?

Exercice 5 :

Ecrivez une fonction récursive $list-abs-recursif$ (donc sans mapcar) qui accepte en entrée une liste de nombres et qui retourne la liste constituée de la valeur absolue de chaque élément de la liste fournie en paramètre.

Exercice 6 :

1. Ecrire une fonction `fabriquer-liste(n e)` qui construit une liste contenant n fois l'élément e.
2. Obtenir la liste `(glou glou glou glou glou)`.

Exercice 7 :

1. Ecrire une fonction `swap-first-last(l)` qui "échange" le premier et le dernier élément d'une liste.
2. La tester avec la liste `'(YOU CANT BUY LOVE)`.

Exercice 8 :

1. Ecrire une fonction `rotate-left(l)` qui fait une rotation circulaire vers la gauche des éléments d'une liste.
2. La tester.

Exercice 9 :

1. Ecrire une fonction `rotate-right(l)` qui fait une rotation circulaire vers la droite des éléments d'une liste.
2. La tester.

Exercice 10 :

1. Ecrire une fonction $sort-numbers(l)$ qui trie une liste de nombres.
2. Ecrire une fonction $sort-symbols(l)$ qui trie une liste de symboles par ordre lexicographique (par exemple on pourra utiliser le prédicat `string<=` ou `string>=`).

Exercice 11 :

- On rappelle qu'une expression symbolique peut se définir de la manière suivante :
  s-expr :== atome | **paire-pointée**
  De même, une liste en C-LISP peut être définie par :
  liste :== nil | **paire-pointée**
  Enfin, une liste propre peut se définir par :
  liste-propre :== () | **(s-expr . liste-propre)**

- On décide d'appeler "liste-sans-point" les listes qui s'afficheront sans point à l'écran. Donner en utilisant le même langage que ci-dessus une définition de **liste-sans-point**
- Ecrire un prédicat `liste-propre-p`
- Ecrire un prédicat `liste-sans-point-p`

- Ecrire une fonction `flatten` qui renvoie la liste (plate) des atomes contenus dans une expression symbolique. On pourra aussi envisager une version récursive terminale (méthode de McCarthy).
- Ecrire une fonction `renverse` qui renverse l'ordre des éléments de premier niveau d'une liste plate. Ecrire ensuite la fonction `renverse*` qui renverse l'ordre des éléments à tous les niveaux de la liste :
  Exemple : (`renverse` '(a b (c d) e f)) ⇒ (f e (c d) b a)
  (`renverse*` '(a b (c d) e f)) ⇒ (f e (d c) b a)
- Ecrire une fonction `sommeListe` qui, étant donnée une liste de nombres, calcule leur somme.
- Ecrire une fonction `compter` qui prend en argument un atome et une liste et calcule le nombre d'occurrences de l'atome dans la liste
  Exemple : (`compter` 'a '(a (b a (c a)) d a)) ⇒ 4
- Ecrire la fonction `iota` qui étant donné un entier n produit la liste $(0\ 1\ 2\ \ldots\ n-1\ \ n)$
- Ecrire la fonction `diviseurs` qui étant donné un entier n produit la liste de tous ses diviseurs
- Ecrire la fonction `begaie` prenant une liste de symboles en argument, une phrase, et retournant en sortie une phrase où tous les mots sont répétés. Exemple : (`begaie` '(Vivement les vacances)) ⇒ (Vivement vivement les les vacances vacances)
- Ecrire une fonction `debegaie` qui ôte d'une phrase tout bégaiement et notamment celui produit par la fonction de l'exo précédent
  Exemple : (`debegaie` (`begaie` '(Vivement les vacances))) ⇒ (Vivement les vacances)
- Ecrire une fonction `produit-scalaire` qui, étant données deux listes (de même taille) passées en paramètres, retourne le produit scalaire des vecteurs qu'elles représentent.