Programmation fonctionnelle

Laura Kallmeyer

TD du 4.10.00

1 Quelques expressions en LISP

1. Qu'est-ce qui sont les valeurs des expressions suivantes?

```
> (car (cdr '((a b c) d)))
> (cdr (car '((a b c) d)))
> (cddr (caaddr '((a b) (c d) ((e g h) f))))

> (cons 'a nil)
> (cons 'a ())
> (cons '((a (b))) '(c))

> (eq 'b (car '(b)))
> (eq (car '(a b)) (cadar '((c a b) d)))
> (eq (car '((a))) '(a))
> (eq (eq 'a 'b) (null '(a b)))
> (eq (eq 'a 'b) '(null (a b)))
> (null (cdr '(c)))
> (null (eq 'a 'b))
```

   Pour tester si quelque chose n'est pas le cas (comme dans le dernier exemple), il vaut mieux se servir de `not`:

```
> (not (eq 'a 'b))
```

2. Comment est-ce qu'on peut extraire l'atome `poire` des expressions suivantes?

```
(poire pomme orange)
(orange (poire) pomme)
(pomme (orange (((poire)) pomme)) citron)
(pomme . poire)
```

3. Qu'est-ce qui sont les valeurs des expressions suivantes?

```
> (and (null nil) (eq 'a 'a))
> (and (car '(a)) (null ()))
> (and 'a 'b 'c 'd)
> (or (car '(a)) (null ()))
> (or nil t)
> (eq 'a (and (car '(a))))
> (or (and (null nil) (cadr '(b a c))) (null (and (null '(a)))))
> (and (and (null nil) (cadr '(b a c))) (null (and (null '(a)))))
```

2 Définir une fonction

On utilise `defun` pour définir une fonction en LISP. `defun` a trois arguments, un nom (le nom de la nouvelle fonction), une liste (la liste de ses paramètres) et le corps de la fonction. Le corps est une suite d'appels de fonctions.

Une fois une fonction est définie, on peut l'appeller avec des arguments pour ses paramètres. En ce moment, les paramètres de la fonction sont liés aux arguments de cet appel et le corps de la fonction est évalué.

```
> (defun add1 (n) (+ 1 n))
ADD1
> (add1 2)
3
```

1. Définissez une fonction `det` qui teste si un atome est un déterminant, c'est-à-dire si l'atome est égal à `un` ou `le`.

   Après sa définition, `det` doit fonctionner comme suit:

   ```
   > (det 'un)
   T
   > (det (cadr '(a b c)))
   NIL
   ```

2. Définissez une fonction `length3` qui prend une liste comme argument et qui teste si la longueur de la liste est 3. (Utilisez `cdr` et `null`.)

   Après sa définition, `length3` doit fonctionner comme suit:

   ```
   > (length3 '((a b) c ((d))))
   T
   > (length3 '(a (b) c ((d))))
   NIL
   ```

3. Définissez une fonction `cdr-car-list` qui prend un doublet qui n'est pas la liste vide est qui retourne une liste contenante comme éléments le cdr et le car du doublet.

   (Pour construire une liste qui contient un seul atome, on applique `cons` a l'atome et la liste vide.)

   Après sa définition, `cdr-car-list` doit fonctionner comme suit:

   ```
   > (cdr-car-list '(a b c))
   ((B C) A)
   > (cdr-car-list '(a . b))
   (B A)
   > (cdr-car-list '(a))
   (A)
   ```

4. Définissez une fonction `xor` (*exclusive or*) qui prend deux arguments et qui retourne une valeur différente de `nil` si exactement un de ses arguments ramène une valeur d'autre que `nil`. Sinon `xor` retourne `nil`.

   Après sa définition, `xor` doit fonctionner comme suit:

   ```
   > (xor t nil)
   T
   > (xor t t)
   NIL
   > (xor () (car '(a)))
   A
   > (xor () ())
   NIL
   ```