

Université Montpellier-II
UFR des Sciences - Département Informatique - Licence Informatique
Programmation Applicative et Récursive

Cours No 9 : Séquences et Effets de bord en programmation récursive.

Notes de cours
Christophe Dony

17 Définitions

Effet de bord : modification explicite de l'état de l'ordinateur (de la mémoire ou de l'affichage).

Séquence : suite d'instructions.

Une instruction effectuant un calcul sans rendre explicitement une valeur effectue un effet de bord.

Fonction récursive avec effet de bord :

```
1 (define (display-elements l)
2   (if (not (null? l))
3     (begin
4       (display (car l)) ;; effet de bord
5       (display "") ;; effet de bord
6       (display-elements (cdr l))))))
```

Exemple : Affichage en profondeur d'abord des données contenues dans un arbre binaire.

```
1 (define (affichage a)
2   (or (arbre-vide? a)
3       (begin (affichage (fg a))
4               (display (val-noeud a))
5               (affichage (fd a))))))
```

Dessins Récursifs

```
1 (define (feuille long angle)
2   (if (> long 1)
3     (begin
4       (draw (/ long 2))
5       (turn (− angle))
6       (feuille (/ long 2) angle)
7       (turn (* angle 2))
8       (feuille (/ long 2) angle)
9       (turn (− angle))
10      (draw (/ long 2))
11      (turn (− angle))
12      (feuille (/ long 2) angle)
13      (turn angle)
14      ;(feuille (/ long 2) angle)
15      (turn angle)
16      (feuille (/ long 2) angle)
17      (turn (− angle))
18      (move (− long))))))
```

```
22 (define (naperon long angle)
23   (let ((n (/ 360 angle)))
24     (define (boucle long angle times)
25       (if (> times 0)
26         (begin
27           (feuille long angle)
28           (turn angle)
29           (boucle long angle (- times 1)))))
31   (boucle long angle n)))

33 (turtles)
34 (turn 90)
35 (naperon 200 60)
```

17.1 Modification physique de la mémoire

Les procédures `set-car!` et `set-cdr!`

La fonction `append` destructrice.

```
1 (define (d-append l1 l2)
2   (cond ((null? l1) l2)
3         ((null? (cdr l1)) (set-cdr! l1 l2) l1)
4         (#t (d-append (cdr l1) l2))))

6 (define matin (list 1 2 3 4 5 6 7 8 9 10 11 12))
7 (define soir (list 13 14 15 16 17 18 19 20 21 22 23 24))
8 (define heures (append matin soir))
```

Les listes circulaires.

```
1 (d-append heures heures)
```

17.1.1 Un arbre binaire impératif

A jout de nouvelles fonctions d'interface pour pouvoir modifier le fils gauche et le fils droit.

```
(define (set-fg! a a2) (set-car! (cdr a) a2))
```

```
(define (set-fd! a a2) (set-car! (cddr a) a2))
```

```
1 (define (p-insere n a)
2   ;; version n'acceptant pas les insertions successives
3   (display a) (display n) (display "\n")
4   (if (arbre-vide? a)
5       (make-feuille n))
6       (let ((valeur (val-noeud a)))
7         (cond ((< n valeur)
8                (if (arbre-vide? (fg a))
9                    (set-fg! a (make-feuille n))
10                   (p-insere n (fg a))))
11              ((> n valeur)
12               (if (arbre-vide? (fd a))
13                   (set-fd! a (make-feuille n))
```

```
14      (p-insere n (fd a))))  
17      ((= n valeur) a))))
```

```
1  (define l2 (make-feuille 5))  
2  (p-insere 4 l2)  
3  (p-insere 7 l2)  
4  (p-insere 8 l2)  
5  (p-insere 1 l2)  
6  (p-insere 2 l2)
```

Exercice : Une version de la fonction p-insere qui autorise l'écriture de (set! l (insere 2 (insere 1 (insere 8 (insere 7 (insere 4 l))))))