Université Bordeaux 1 Licence S&TS Mention Informatique S5
2009/2010

UE INF353 Programmation 3

Programmation Fonctionnelle et Symbolique

Devoir surveillé No 1

Tous documents autorisés. Vendredi 6 Novembre 2009 Durée : 1h20.
Le barème est donné à titre **indicatif**.
Sur chaque feuille, indiquez vos nom, prénom et numéro de groupe.

Exercice 1 *(7pts)*

Évaluer les expressions suivantes :

1. `(cons '(1 2) '(4))`
2. `(cons '(1 2) 4)`
3. `(cons '(1 2) nil)`
4. `(append '(1 2) nil '(4))`
5. `(list '(1 2) '(4))`
6. `(list '(1 2) 4)`
7. `(list '(1 2) nil '(4))`
8. `'(* 1 2)`
9. `(last '(1 2))`
10. `(cdadr '((1 2) ((3 4) (5 6))))`
11. `(member 'A '(P R O G R A M M E))`
12. `(member '(1 2) '((1 1) (1 2) (2 1)))`
13. `(mapcar #'+ '(1 3 5) '(2 4 6) '(1 1 1))`
14. `(some (lambda (x) (and (zerop (mod x 3)) x)) '(1 2 3 4))`

Exercice 2 *Représentation d'arbres binaires étiquetés (4pts)*

On considère les arbres binaires étiquetés par des entiers positifs.

– L'arbre vide est représenté par `NIL`.
– Un arbre non vide est représenté par une liste à trois éléments `(sag eti sad)` où `sag` est le sous-arbre gauche, `eti` est l'étiquette de la racine et `sad` est le sous-arbre droit.

Ainsi

– la liste `(NIL 12 NIL)` représente l'arbre réduit à une feuille étiquetée `12`,
– la liste `(((nil 12 nil) 5 (nil 10 nil)) 2 ((nil 8 nil) 3 (nil 7 nil)))` représente l'arbre de la figure 1,

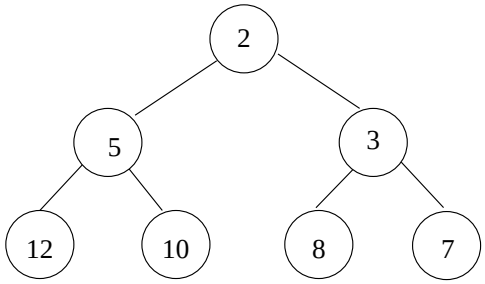

FIG. 1 – Arbre 1

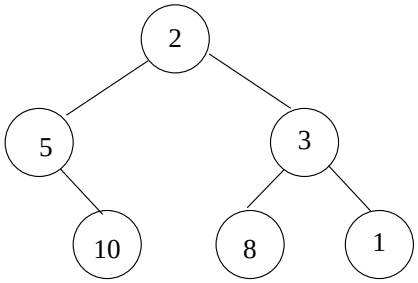

FIG. 2 – Arbre 2

– et la liste `((nil 5 (nil 10 nil)) 2 ((nil 8 nil) 3 (nil 1 nil)))` représente l'arbre de la figure 2.

1. Implémenter la fonction `arbre-vide ()` qui retourne un arbre vide.
2. Implémenter le prédicat `arbre-vide-p (arbre)` qui teste si un arbre est vide.
3. Implémenter la fonction `nouvel-arbre (g e d)` qui construit un nouvel arbre dont la racine est étiquetée `e` et ayant `g` comme sous-arbre gauche et `d` comme sous-arbre droit.
4. Implémenter la fonction `sag (arbre)` qui retourne le sous-arbre gauche de `arbre`.
5. Implémenter la fonction `sad (arbre)` qui retourne le sous-arbre droit de `arbre`.
6. Implémenter la fonction `eti (arbre)` qui retourne l'étiquette de la racine de `arbre`.
7. Implémenter le prédicat `feuille-p (arbre)` qui teste si un arbre est réduit à une feuille.

Exemples :

```
CL-USER> (arbre-vide)
NIL
CL-USER> (nouvel-arbre
          (nouvel-arbre (arbre-vide) 1 (arbre-vide))
          0
          (nouvel-arbre (arbre-vide) 6 (arbre-vide)))
((NIL 1 NIL) 0 (NIL 6 NIL))
CL-USER> *arbre1*
```

```
(((NIL 12 NIL) 5 (NIL 10 NIL)) 2 ((NIL 8 NIL) 3 (NIL 7 NIL)))
CL-USER> *arbre2*
((NIL 5 (NIL 10 NIL)) 2 ((NIL 8 NIL) 3 (NIL 1 NIL)))
CL-USER> (sag *arbre1*)
((NIL 12 NIL) 5 (NIL 10 NIL))
CL-USER> (eti *arbre1*)
2
CL-USER> (sad *arbre1*)
((NIL 8 NIL) 3 (NIL 7 NIL))
CL-USER> (feuille-p *arbre1*)
NIL
CL-USER> (feuille-p (nouvel-arbre nil 0 nil))
T
```

Exercice 3 *Manipulation d'arbres binaires étiquetés (9pts)*

Dans cet exercice, on utilisera les constructeurs, accesseurs et prédicats définis pour les arbres binaires étiquetés à l'exercice précédent.

1. Implémenter la fonction `eti-max-arbre (arbre)` qui retourne `0` si l'arbre est vide, son étiquette maximum sinon.
2. Implémenter la fonction `map-arbre (f arbre)` qui retourne un arbre ayant la même structure et dans lequel chaque noeud étiqueté `i` dans `arbre` est étiqueté `f(i)`.
3. Implémenter le prédicat `arbre-every (pred arbre)` qui teste si toutes les étiquettes de l'`arbre` vérifient le prédicat `pred`.
4. Donner un exemple d'appel à `arbre-Every` et sa valeur de retour.
5. Implémenter la fonction `etiquettes (arbre)` qui retourne l'ensemble des étiquettes de l'`arbre`.

Exemples :

```
CL-USER> (eti-max-arbre (arbre-vide))
0
CL-USER> (eti-max-arbre *arbre1*)
12
CL-USER> (map-arbre (lambda (x) (* x x)) *arbre1*)
(((NIL 144 NIL) 25 (NIL 100 NIL)) 4 ((NIL 64 NIL) 9 (NIL 49 NIL)))
CL-USER> *arbre3*
((NIL 5 (NIL 10 NIL)) 3 ((NIL 8 NIL) 3 (NIL 5 NIL)))
CL-USER> (etiquettes *arbre3*)
(10 8 3 5)
```

FIN