DEUG MIAS 2e année
TD de programmation fonctionnelle n° 2
Les arbres

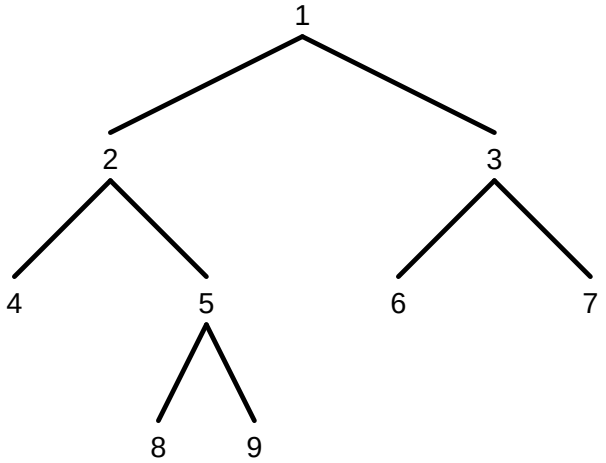
Pierre BOULET Jean-Marc NICOD Frédéric VIVIEN

Le but de ce TD-TP est de manipuler des arbres. En TD on manipulera des arbres binaires et en TP des arbres n-aires.

TD : Arbres binaires

Un arbre binaire est soit une feuille (tout objet qui n'est pas une liste), soit un nœud interne (une liste de trois éléments : l'étiquette du nœud et les deux arbres fils gauche et fils droit).

Vous testerez toutes vos fonctions sur l'arbre suivant :



1 Fonctions de base

Question 1.1. Construisez une fonction *make-arbre* qui construit un arbre à partir de l'étiquette du nœud et de ses deux fils.

```
(define make-arbre
  (lambda (etiquette fg fd)
    (list etiquette fg fd)))
```

Question 1.2. Construisez les fonctions *etiquette*, *fg* et *fd* qui extraient d'un nœud interne respectivement son étiquette, son fils gauche et son fils droit.

```
(define etiquette
   (lambda (a)
     (car a)))
(define fg
   (lambda (a)
     (cadr a)))
(define fd
   (lambda (a)
     (caddr a)))
```

Question 1.3. Définissez des inspecteurs : *feuille?* qui teste si un objet est une feuille et *arbre?* qui teste si un objet est un arbre.

```
(define feuille?
   (lambda (o)
     (not (list? o))))
(define arbre?
   (lambda (o)
     (or (feuille? o)
         (and (list? o)
              (arbre? (fg o))
              (arbre? (fd o))
              (null? (cdddr o)))))))
```

2 Calculs sur les arbres

Question 2.1. Définissez une fonction qui calcule la hauteur d'un arbre.

Note. La hauteur d'une feuille est 1.

```
(define hauteur
   (lambda (a)
     (if (feuille? a)
         1
         (+ 1 (max (hauteur (fg a)) (hauteur (fd a)))))))
```

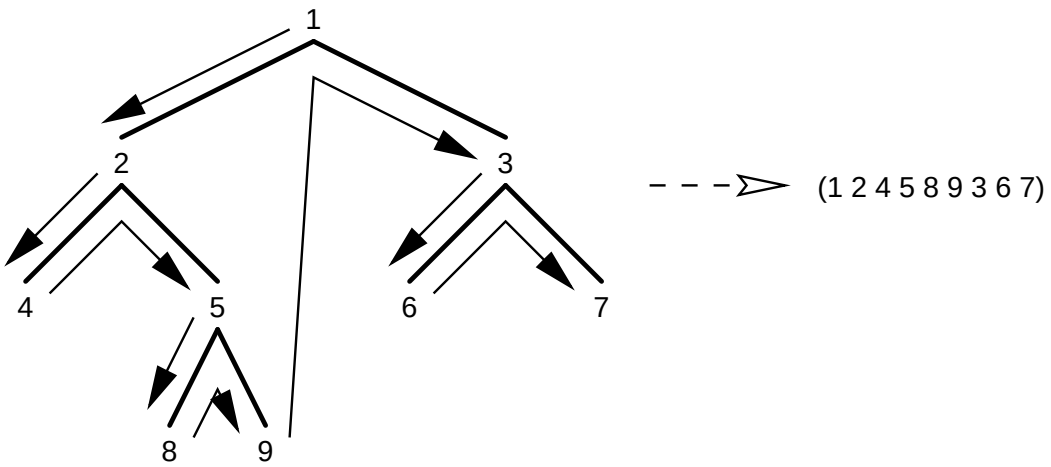
Question 2.2. Construisez les fonctions *nb-feuilles* et *nb-noeuds* qui calculent respectivement le nombre de feuilles et le nombre de nœuds total de l'arbre (nœuds internes et feuilles).

```
(define nb-feuilles
   (if (feuille? a)
       1
       (+ (nb-feuilles (fg a)) (nb-feuilles (fd a)))))
(define nb-noeuds
   (if (feuille? a)
       1
       (+ 1 (nb-noeuds (fg a)) (nb-noeuds (fd a)))))
```

3 Parcours dans des arbres

Question 3.1. Écrivez une fonction *parcours-profondeur* qui énumère les étiquettes des nœuds de l'arbre en profondeur d'abord.

Exemple 1.



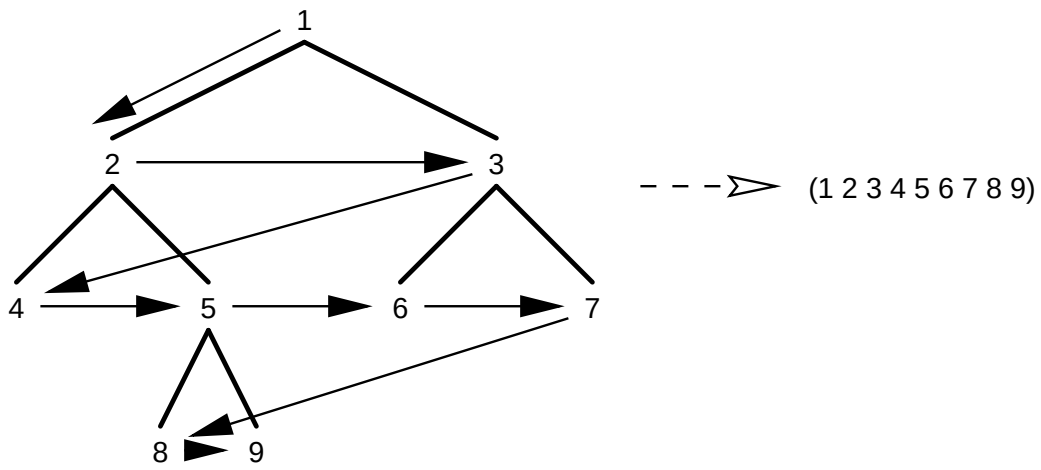

```
(define parcours-profondeur
  (lambda (a)
    (if (feuille? a)
        (list a)
        (append (parcours-profondeur (fg a))
                (list (etiquette a))
                (parcours-profondeur (fd a))))))
```

Question 3.2. Sur le même schéma qu'à la question précédente, écrivez une fonction *recherche* qui teste si un élément est dans un arbre.

```
(define recherche
  (lambda (e a)
    (if (feuille? a)
        (eqv? e a)
        (or (eqv? e (etiquette a))
            (recherche e (fg a))
            (recherche e (fd a))))))
```

Question 3.3. Définissez une fonction *parcours-largeur* qui parcours un arbre en largeur d'abord.

Exemple 2.




```
(define valeur-racine
   (lambda (a)
     (if (feuille? a)
         a
         (etiquette a))))
(define fils
   (lambda (a)
     (if (feuille? a) '() (list (fg a) (fd a)))))
(define parcours-largeur
   (lambda (a)
     (letrec
          ((parcours
            (lambda (l)
              (if (null? l)
                  '()
                  (cons (valeur-racine (car l))
                        (parcours (append (cdr l) (fils (car l)))))))))
        (parcours (list a)))))
```

Question 3.4. Écrivez une fonction *print-arbre* qui affiche joliment un arbre.

Exemple 3. Si on a l'arbre *a1* : (1 (2 3 4) 5), on peut l'afficher comme suit :

```
 /--5
1
      /--4
 \--2
      \--3
```

```
(define print-arbre
   (lambda (a)
     (letrec
          ((ecrit-atome
            (lambda (dec dir f)
              (cond
               ((= 0 dec)
                (begin
                  (write f)
                  (newline)))
               ((= 1 dec)
                (begin
                     (if (eqv? 'd dir)
                         (display " /-")
                         (display " \\-"))
                     (ecrit-atome 0 dir f)))
               (else (begin
                       (display "    ")
                       (ecrit-atome (- dec 1) dir f))))))
           (ecrit
            (lambda (dec dir a)
              (if (arbre? a)
                  (if (feuille? a)
                      (ecrit-atome dec dir a)
                      (begin
                        (ecrit (+ 1 dec) 'd (fd a))
                        (ecrit-atome dec dir (etiquette a))
                        (ecrit (+ 1 dec) 'g (fg a))))
                  (ecrit-atome "erreur")))))
       (ecrit 0 '() a))))
```

TP : Arbres n-aires

Question 3.5. Reprenez les questions du TD en travaillant sur des arbres n-aires, c'est-à-dire que chaque nœud peut avoir de 0[1] à n fils. Ces fils sont représentés par une liste.

```
(define make-arbre
   (lambda (etiquette fils)
     (list etiquette fils)))

(define make-feuille
   (lambda (etiquette)
     (make-arbre etiquette '())))
```

1. un nœud sans fils est une feuille

```
(define etiquette
   (lambda (a)
      (car a)))

(define fils
   (lambda (a)
      (cadr a)))

(define feuille?
   (lambda (a)
      (null? (cadr a))))

(define arbre?
   (lambda (o)
      (and (list? o)
           (apply and (map arbre? (fils o)))
           (null? (cddr o)))))

(define hauteur
   (lambda (a)
      (if (feuille? a)
          1
          (+ 1 (apply max (map hauteur (fils a)))))))

(define nb-feuilles
   (lambda (a)
      (if (feuille? a)
          1
          (iter (apply + (map nb-feuilles (fils a)))))))

(define nb-noeuds
   (lambda (a)
      (if (feuille? a)
          1
          (+ 1 (apply + (map nb-noeuds (fils a)))))))

(define flat-map
   (lambda (f l)
      (if (null? l)
          '()
          (append (f (car l)) (flat-map f (cdr l))))))

(define parcours-profondeur
   (lambda (a)
      (if (feuille? a)
          (list (etiquette a))
```

```
        (cons (etiquette a)
              (flat-map parcours-profondeur (fils a))))))

(define recherche
   (lambda (e a)
     (if (feuille? a)
         (eqv? e (etiquette a))
         (or (eqv? e (etiquette a))
             (apply or (map (lambda (x) recherche e x) (fils a)))))))

(define parcours-largeur
   (lambda (a)
     (letrec
          ((parcours
            (lambda (l)
              (if (null? l)
                  '()
                  (cons (etiquette (car l))
                        (parcours (append (cdr l) (fils (car l))))))))))
        (parcours (list a)))))

(define print-arbre
   (lambda (a)
     (letrec
          ((ecrit-atome
            (lambda (dec dir f)
              (cond
               ((= 0 dec)
                (begin
                  (write f)
                  (newline)))
               ((= 1 dec)
                (begin
                    (if (eqv? 'd dir)
                        (display " /-")
                        (display " \\-"))
                    (ecrit-atome 0 dir f)))
               (else (begin
                       (display "    ")
                       (ecrit-atome (- dec 1) dir f))))))
           (ecrit
            (lambda (dec dir a)
              (if (arbre? a)
                  (if (feuille? a)
                      (ecrit-atome dec dir (etiquette a))
```

```
                (begin
                  (ecrit-atome dec dir (etiquette a))
                  (for-each (lambda (x) (ecrit (+ 1 dec) 'g x)) (fils a))))
              (ecrit-atome "erreur")))))
(ecrit 0 '() a))))
```