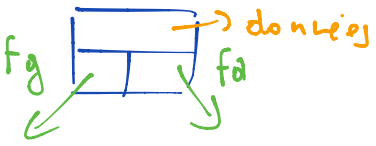
Structures arborescentes :

→ Arbre binaire avec chaînages.

1 arbre = l'@ de la racine.



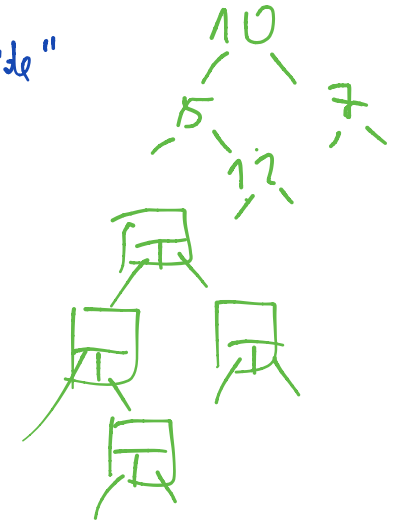
fg = fils gauche.
fd = fils droit

Mêmes op. : champ de données $x \to d$.
@ d'un nœud.
0 si arbre vide.
+ les liens vers les sous-arbres.

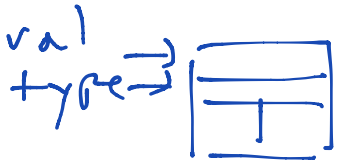
Algorithme de parcours :

Visit(x) ← @ d'un nœud

Si $x == 0$ Alors "traitement arbre vide"
Sinon : traitement de x n°1
Visit ($x \to fg$)
traitement de x n°2
Visit ($x \to fd$)
traitement de n°3

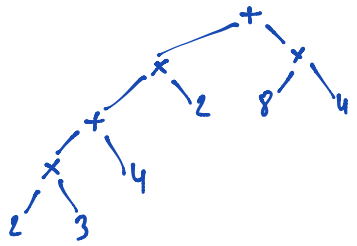


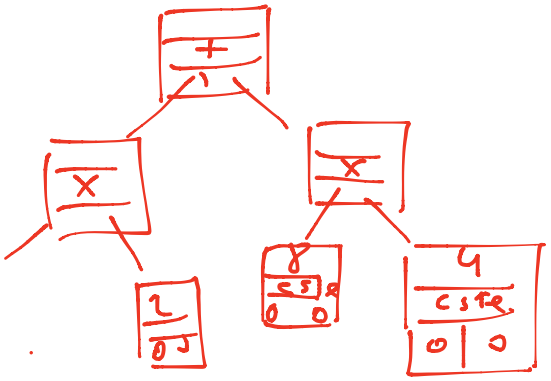
Exemple : expression arithmétique



On prend des noeuds avec : - un champ valeur → entier
- un type de noeud → +, -, /, *, %, cste
- fg, fd

$((2 \times 3) + 4) \times 2 + (8 \times 4)$





Visit(x)

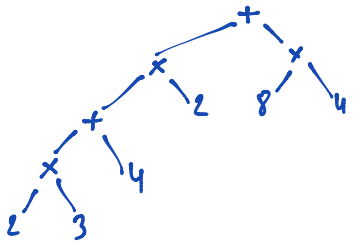
Si x == 0 Alors "traitement arbre vide"

Sinon : traitement de x n°1
Visit(x → fg)
traitement de x n°2
Visit(x → fd)
traitement de n°3

traitement :

Si x→type = cste
print x→val
Sinon
print x→type

avec traitement n°1 :



+ x + x 2 3 4 2 x 8 4

6 32

10

20

52

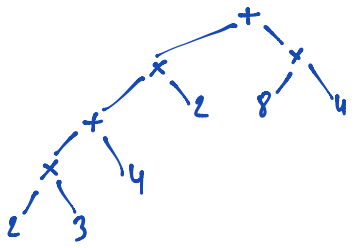
traite n°2 :

$2 \times 3 + 4 \times 2 + 8 \times 4$

traite n°3 : 2 3 x 4 + 2 x 8 4 x +

Amélioration du "traitement infixe" (ambigu) :

```
Affiche(e) :
    Si e→type == cste Alors print e→val
    Sinon : print "("
            Affiche(e→fg)
            print e→type
            Affiche(e→fd)
            print ")"
```



$(((((2 \times 3) + 4) \times 2) + (8 \times 4))$

Parcours en "itératif"

```
Parcours (x) :
    P : Pile vide.
    n = 1
    Répéter :
        Si n==1 Alors :
            tant que x≠∅ :   (x ≠ arbre vide)
                traitement n°1 de x
                Empiler (P, (x,1))
                x = x→fg
            Traitement arbre vide.
        Si P non vide Alors :
            (x,n) := tete(P)      ) extraction de la tête
            Depiler(P)
            Si n==1 Alors :
                Traitement n°2 de x
                Empiler (P, (x,2))
                x = x→fd
            Sinon : traitement n°3 de x
    jusqu'à P=∅
fin.
```