

# Cours 2

## Récurtivité

Jean-Jacques.Levy@inria.fr  
http://jeanjacqueslevy.net  
tel: 01 39 63 56 89

secrétariat de l'enseignement:  
Catherine Bensoussan  
cb@lix.polytechnique.fr  
Aile 00, LIX  
tel: 01 69 33 34 67

<http://www.enseignement.polytechnique.fr/informatique/>

## Plan

1. Récursivité numérique
2. Raisonnement inductif
3. Diviser pour régner
4. Structures de données récursives
5. Récursivité graphique

## Récurtivité numérique

```
static int fib (int n) {
    if (n <= 1) return 1;
    else return fib (n-1) + fib (n-2);
}

static int fact (int n) {
    if (n <= 1)
        return 1;
    else
        return n * fact (n-1);
}

static int cnp (int n, int p) {
    if ((n == 0) || (p == n))
        return 1;
    else
        return cnp(n-1, p-1) + cnp(n-1, p);
}
```

Une fonction peut s'appeler **elle-même** à l'intérieur de sa définition.

## Calcul récursif de Fibonacci

```
fib(4) -> fib (3) + fib (2)
      -> (fib (2) + fib (1)) + fib (2)
      -> ((fib (1) + fib (1)) + fib (1)) + fib(2)
      -> ((1 + fib(1)) + fib (1)) + fib(2)
      -> ((1 + 1) + fib (1)) + fib(2)
      -> (2 + fib(1)) + fib(2)
      -> (2 + 1) + fib(2)
      -> 3 + fib(2)
      -> 3 + (fib (1) + fib (1))
      -> 3 + (1 + fib(1))
      -> 3 + (1 + 1)
      -> 3 + 2
      -> 5
```

Fonctions récursives: théorie inventée par **Kleene** [1909–1994]

## Récurtivité imbriquée

```
static int ack(int m, int n) {      fonction d'Ackermann
    if (m == 0)
        return n + 1;
    else if (n == 0)
        return ack (m - 1, 1);
    else
        return ack (m - 1, ack (m, n - 1));
}

static int g(int m, int n) {      fonction de Morris
    if (m == 0)
        return 1;
    else
        return g(m - 1, g(m, n));
}
```

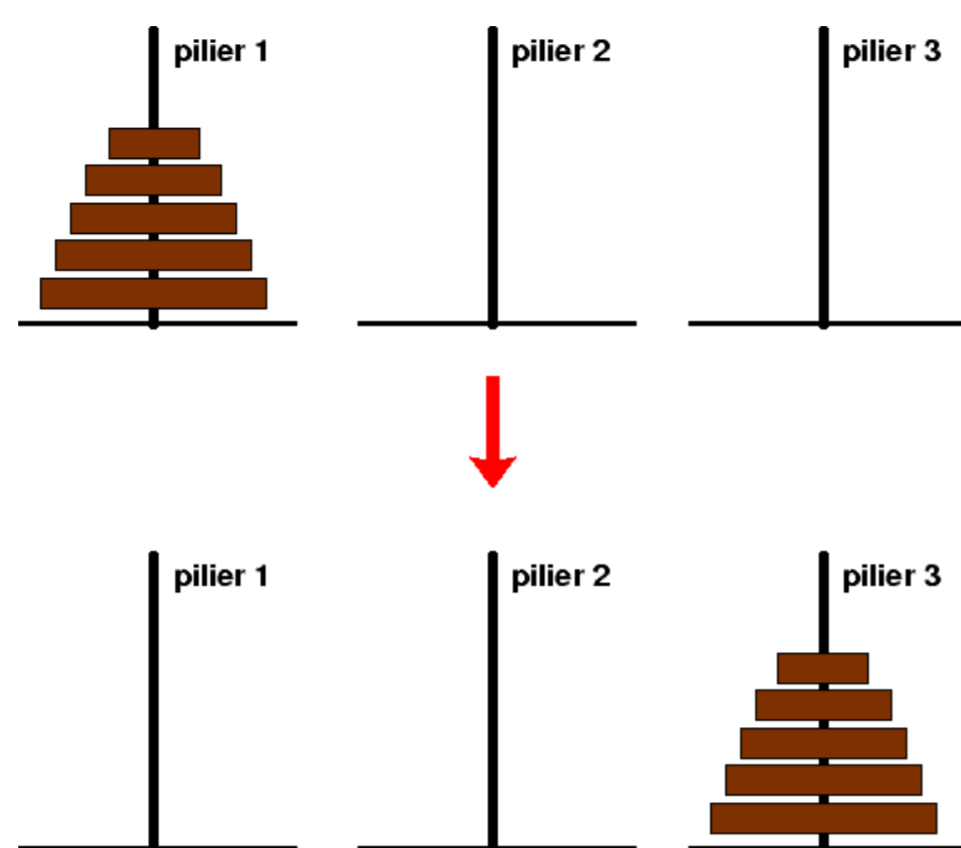
**Exercice 1** Montrer que `ack` termine pour  $m, n \in \mathbb{N}$ .

**Exercice 2** Trouver les valeurs de `ack(0,n)`, `ack(1,n)` et `ack(2,n)`.

**Exercice 3** Donner la valeur de `g(1,0)`.

Java implémente l'**appel par valeur**.

## Les tours de Hanoi

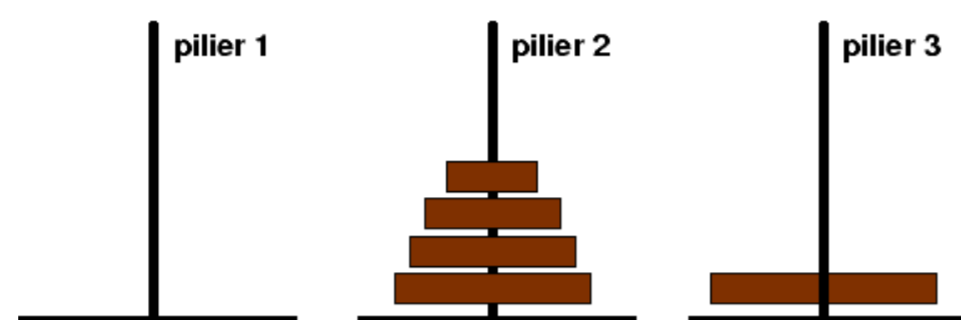
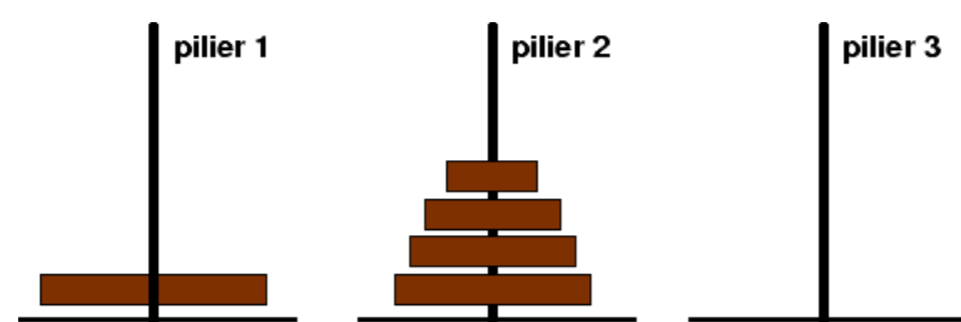


### Règle du jeu

On déplace **une** rondelle à la fois.

Une rondelle ne doit jamais se trouver **sous** une rondelle plus grosse qu'elle-même.

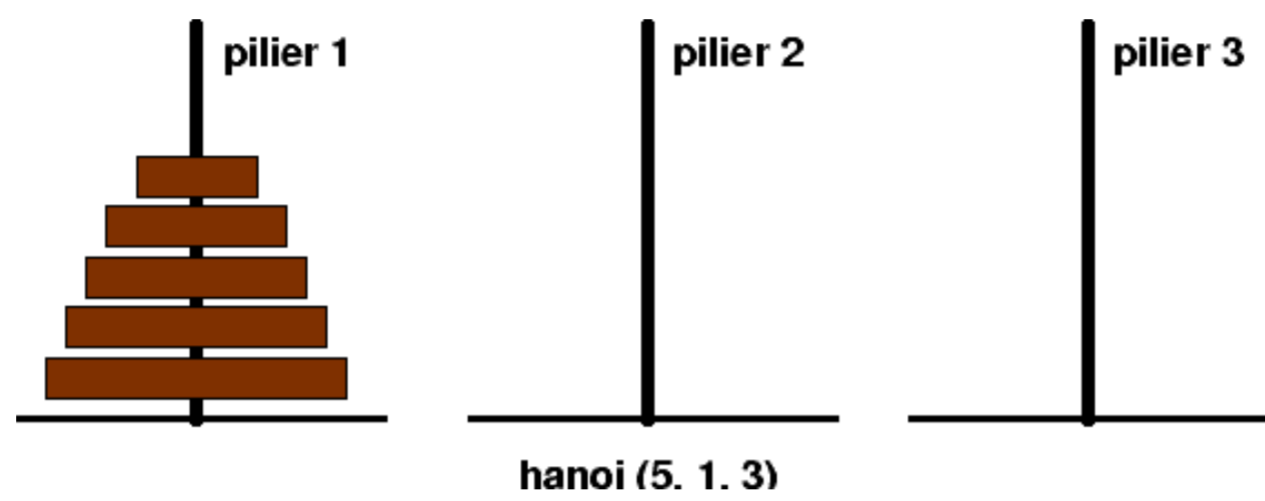
## Les tours de Hanoi – positions intermédiaires



## Les tours de Hanoi

C'est l'exemple du raisonnement inductif dépassant le cas des récurrences numériques.

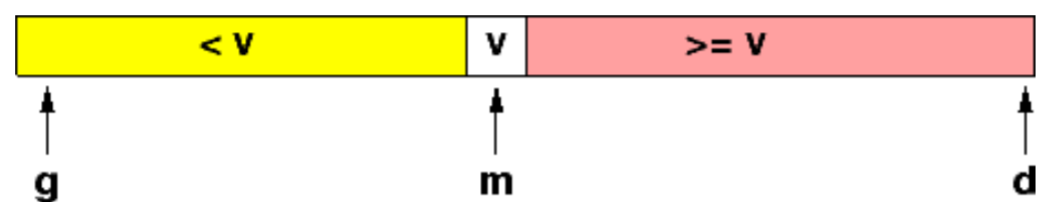
```
static void hanoi(int n, int i, int j) {  
    if (n > 0) {  
        hanoi (n-1, i, 6-(i+j));  
        System.out.println (i + " -> " + j);  
        hanoi (n-1, 6-(i+j), j);  
    }  
}
```



**Exercice 4** Faire les tours de Hanoi avec un programme itératif!

## QuickSort [Hoare 1960]

```
static void qSort(int[] a, int g, int d) {  
    if (g < d) {  
        int v = a[g];  
        Partitionner le tableau autour de la valeur v  
        et mettre v à sa bonne position m  
        qSort (a, g, m-1);  
        qSort (a, m+1, d);  
    }  
}
```

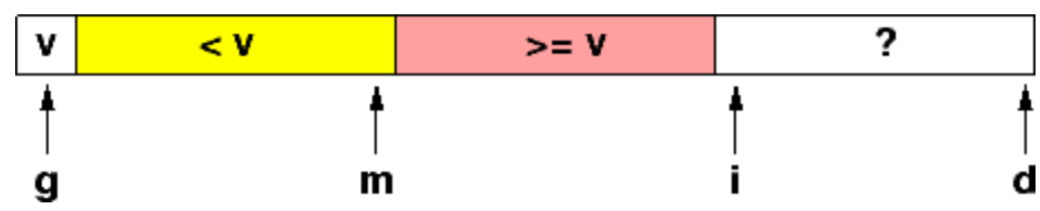


**Complexité en moyenne:**  $C_n \simeq 1,38 n \log n$ , très bon.

$O(n^2)$  dans le pire cas.

## QuickSort – suite

```
static void qSort(int a[], int g, int d) {  
    if (g < d) {  
        int v = a[g];  
        int m = g;  
        for (int i = g+1; i <= d; ++i)  
            if (a[i] < v) {  
                ++m;  
                int x = a[m]; a[m] = a[i]; a[i] = x;  
            }  
        int x = a[m]; a[m] = a[g]; a[g] = x;  
        qSort (a, g, m-1);  
        qSort (a, m+1, d);  
    }  
}
```



## Tri par fusion

```
static void trifusion (int[] a, int[] b, int g, int d) {
    if (g < d) {
        int m = (g + d) / 2;
        trifusion(a, b, g, m); trifusion(a, b, m + 1, d);
        for (int i = m; i >= g; --i)
            b[i] = a[i];
        for (int j = m+1; j <= d; ++j)
            b[d+m+1-j] = a[j];
        int i = g, j = d;
        for (int k = g; k <= d; ++k)
            if (b[i] < b[j]) {
                a[k] = b[i]; ++i;
            } else {
                a[k] = b[j]; --j;
            }
    }
}
```

Complexité en  $O(n \log n)$ .

## Autre exemple de diviser pour régner: *splines*

Interpolation entre plusieurs points (coniques, cubiques, etc).  
Utiles en CAO, en graphique interactif, pour générer des polices de caractères (PostScript, Metafont).

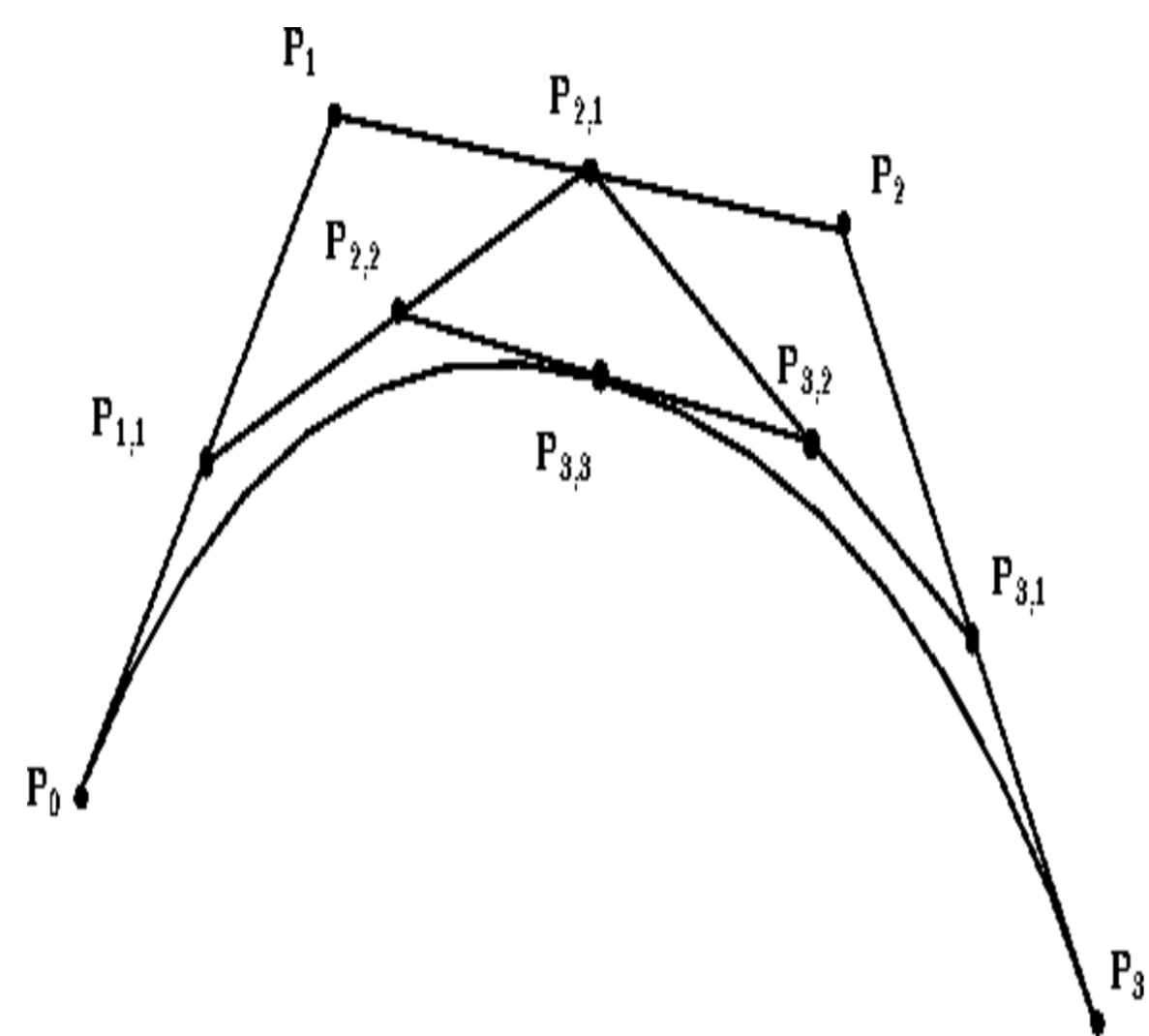
Les plus simples sont les courbes paramétriques de **Bézier** (ingénieur à Renault) et de de Casteljau.

Les cubiques de Bézier (*splines*) sont données par 4 points,  $P_0, P_1, P_2, P_3$ : la courbe passe par  $P_0$  et  $P_3$  et les dérivées en  $P_0$  et  $P_3$  sont les vecteurs  $\overrightarrow{P_0P_1}$  et  $\overrightarrow{P_2P_3}$ . (Remarque: la cubique est toujours inscrite dans le quadrilatère  $P_0P_1P_2P_3$ .)

On les dessine facilement avec une méthode Diviser pour Régner, en prenant les milieux:  $P_{1,1}$  de  $P_0P_1$ ,  $P_{2,1}$  de  $P_1P_2$ ,  $P_{3,1}$  de  $P_2P_3$ ,  $P_{2,2}$  de  $P_{1,1}P_{2,1}$ ,  $P_{3,2}$  de  $P_{2,1}P_{3,1}$ ,  $P_{3,3}$  de  $P_{2,2}P_{3,2}$

et en considérant les courbes de Bézier pour  $P_0P_{1,1}P_{2,2}P_{3,3}$  et  $P_{3,3}P_{3,2}P_{3,1}P_3$ . (Il suffit de tracer le segment  $P_0P_3$  pour un quadrilatère de petite surface).

## Splines – bis



Un expert est Lyle Ramshaw (DEC/SRC Tech Report 19). On en

parle au cours Graphique de Sillion dans la majeure Math et Info,  
M1, en 2ème année.

## Types de données récursifs

```
// Liste = {Liste_vider}  $\cup$  {hd:Element; tl:Liste}
class Liste {
    int hd; Liste tl;

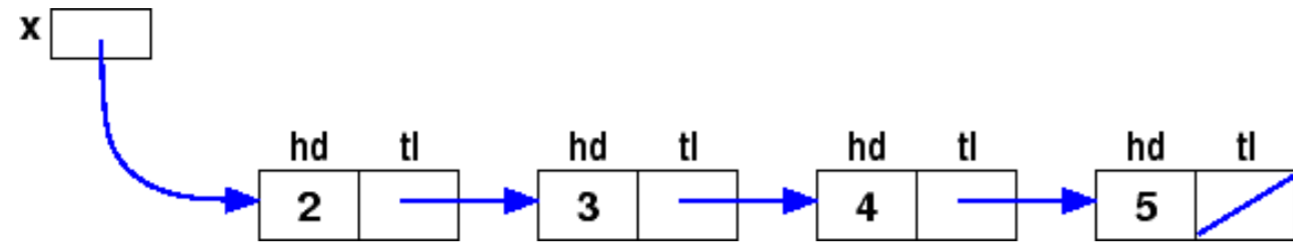
    Liste (int v, Liste a) {
        hd = v; tl = a;
    }
}

// Arbre = {Arbre_vider}  $\cup$  {val:Element; fG:Arbre; fD:Arbre}
class Arbre {
    int val; Arbre fG, fD;

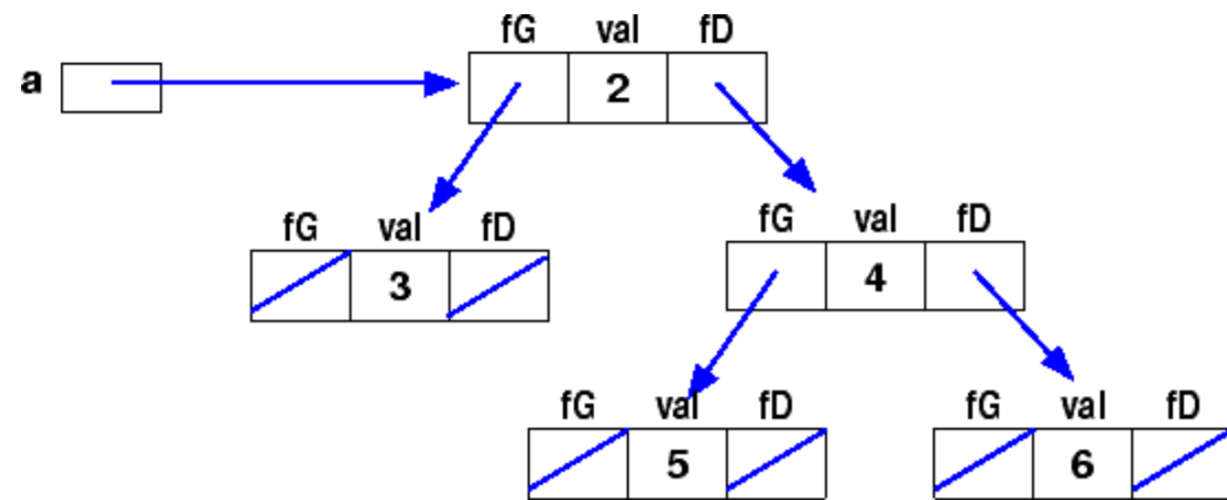
    Arbre (int v, Arbre a, Arbre b) {
        val = v; fG = a; fD = b;
    }
}
```

## Types de données récursifs – bis

```
Liste x = new Liste (2, new Liste (3, new Liste (4, null)));
```



```
Arbre a = new Arbre (2, new Arbre (3, null, null),  
                    new Arbre (4, new Arbre (5, null, null),  
                                new Arbre (6, null, null)));
```



## Types de données récurifs – ter

A types de données récurifs correspond fonctions récurives définies par **récurrence structurelle**.

```
static int longueur (Liste x) {
    if (x == null) return 0;
    else return 1 + longueur(x.tl);
}
```

```
static int hauteur (Arbre a) {
    if (a == null) return 0;
    else return 1 + Math.max(hauteur(a.fG), hauteur(a.fD));
}
```

### Autre écriture (itérative)

```
static int longueur (Liste x) {
    int r = 0;
    for (; x != null; x = x.tl)
        ++r;
    return r;
}
```

**Moins élégant.**

## Récurtivité graphique, courbes fractales

Gaston Julia et Benoît Mandelbrot en sont les pères fondateurs.

Le flocon de von Koch [1]

La courbe du dragon [1, 2]

La courbe de Hilbert [1]

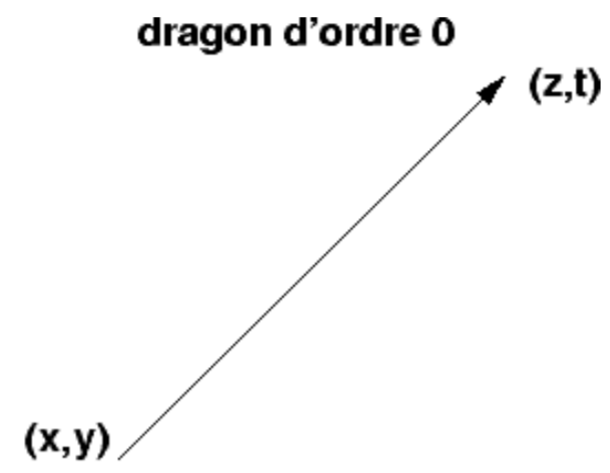
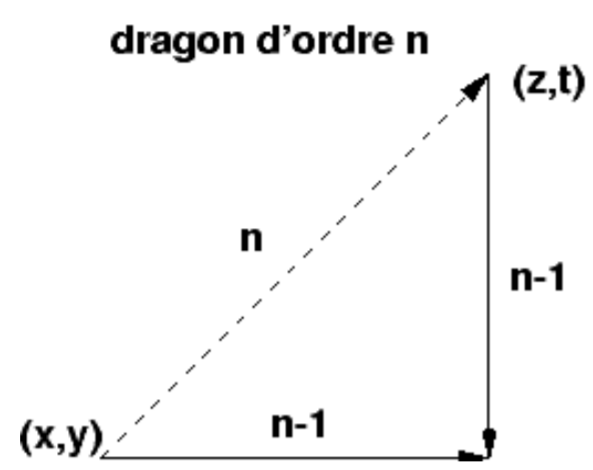
La courbe de Sierpinsky [1 2 3 4]

Les fougères de Barnsley [1]

Les systèmes de Lindenmayer [1]

## La courbe du dragon

```
static void dragon(int n, int x, int y, int z, int t) {  
    if (n == 0) {  
        MacLib.moveTo (x, y);  
        MacLib.lineTo (z, t);  
    } else {  
        int u = (x + z + t - y) / 2, v = (y + t - z + x) / 2;  
        dragon (n-1, x, y, u, v);  
        dragon (n-1, z, t, u, v);  
    }  
}
```



## La courbe du dragon – sans lever le crayon

```
static void dragon (int n, int x, int y, int z, int t) {
    if (n == 0) {
        MacLib.moveTo (x, y);
        MacLib.lineTo (z, t);
    } else {
        int u = (x + z + t - y) / 2, v = (y + t - z + x) / 2;
        dragon (n-1, x, y, u, v);
        dragonBis (n-1, u, v, z, t);
    }
}

static void dragonBis(int n, int x, int y, int z, int t) {
    if (n == 0) {
        MacLib.moveTo (x, y);
        MacLib.lineTo (z, t);
    } else {
        int u = (x + z - t + y) / 2, v = (y + t + z - x) / 2;
        dragon (n-1, x, y, u, v);
        dragonBis (n-1, u, v, z, t);
    }
}
```

**Récurtivité croisée.**

## Les systèmes de Lindenmayer - *L-systems*

Les *L*-systèmes décrivent les courbes fractales avec un graphique du genre de la petite tortue du langage Logo.

Ordres de base:

- $F$  dessine vers l'avant,
- $G$  sauter vers l'avant,
- tourner dans le sens trigonométrique,
- + tourner dans le sens contraire

Définitions récursives des **symboles** ( $F$  et  $G$  compris). Par exemple pour un angle unitaire  $\alpha = 60^\circ$

$$A = F ++F ++F ++$$

$$F = F - F ++F - F$$

dessine le flocon de von Koch en prenant  $A$  pour axiome.

A chaque étape, **tous** les symboles figurant dans la chaîne courante issue de l'axiome sont remplacés par leur définition. A la fin on interprète la suite de commandes élémentaires pour effectuer le dessin.

## Les systèmes de Lindenmayer - suite

**Exercice 5** Donner le  $L$ -système pour la courbe de Hilbert.

**Exercice 6** Donner le  $L$ -système pour la courbe du Dragon.

**Exercice 7** Donner le  $L$ -système pour la courbe de Sierpinsky.

Rajoutons deux symboles [ et ] pour sauver l'état graphique de la tortue ou pour le restaurer.

On peut obtenir des arbres penchés

**Exercice 8** Essayer  $F = FF + [+F - F - F] - [-F + F + F]$  avec  $\alpha = 25^\circ$ .

**Exercice 9** Essayer  $F = F[+F]F[-F]F$  avec  $\alpha = 25.7^\circ$

**Exercice 10** Considérer les motifs récurrents de la page suivante. Trouver les  $L$ -systèmes pour quelques cas.

La nature est souvent fractale.



