STRUCTURES DE DONNÉES
MIP — S4

Pr. K. Abbad & Pr. Ad. Ben Abbou & Pr. A. Zahi
Département Informatique
FSTF
2019-2020

Séance 7 – Les Arbres

- Illustrations
- Définitions
- Implémentation

Arbres — *Illustrations*

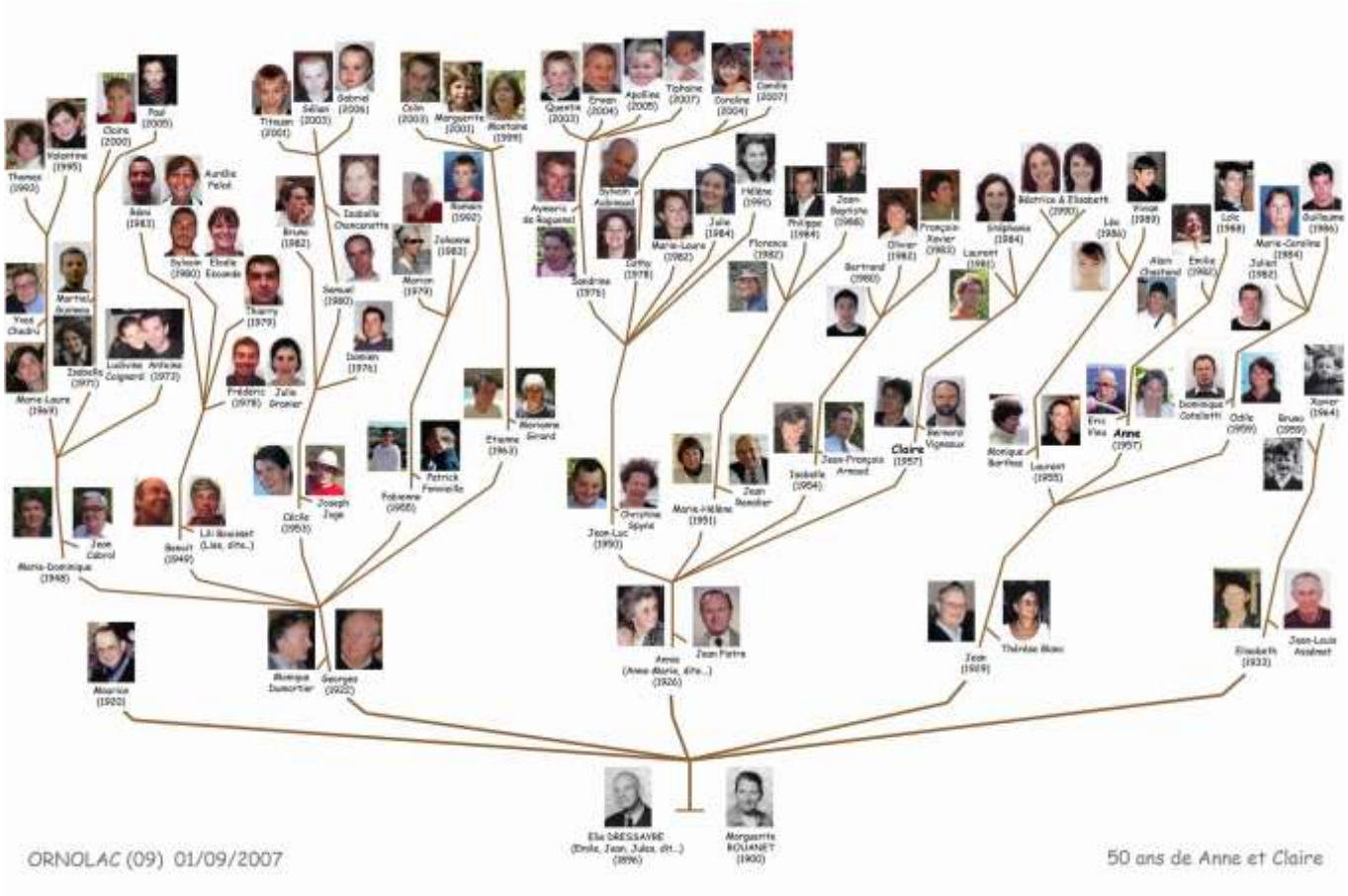
- Un arbre est une structure qui permet de représenter la notion d'hiérarchisation dans un domaine.

- Plusieurs exemples d'utilisation

  - Généalogie
  - Aide à la décision
  - Expressions arithmétiques
  - systèmes de fichiers

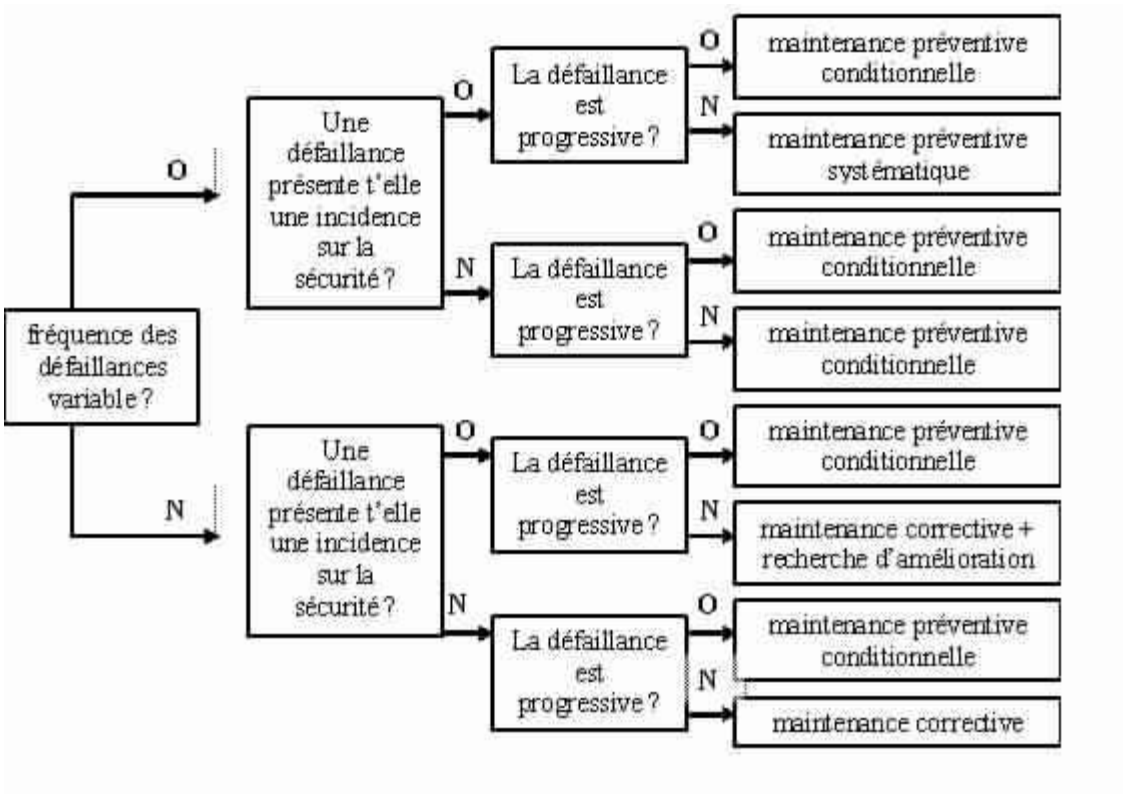
Arbres — *Illustrations*

- Arbre généalogique



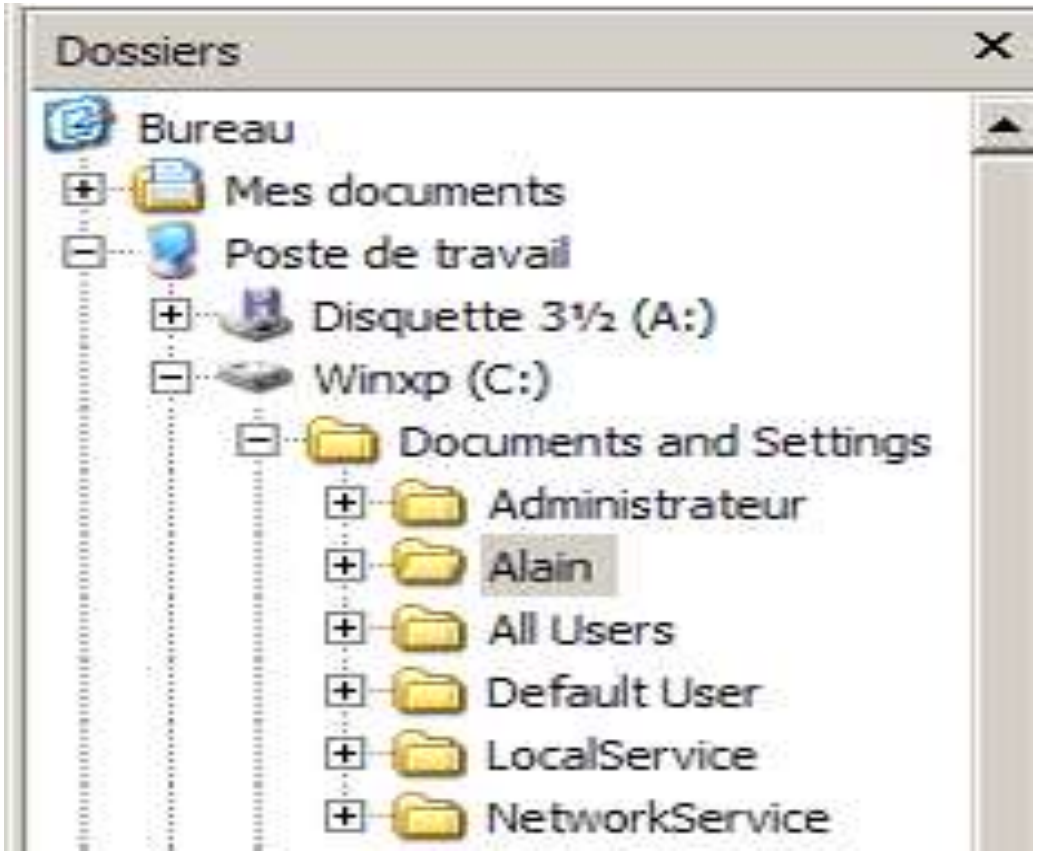
Arbres — *Illustrations*

- Arbre de décision



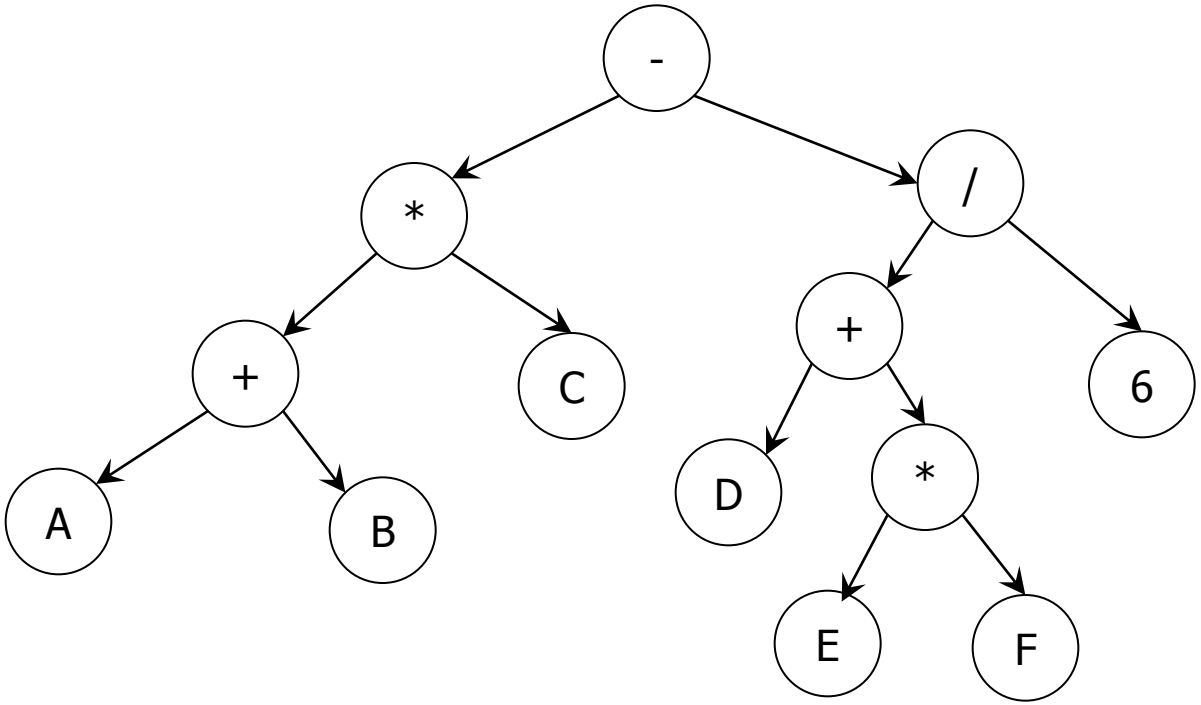
Arbres — *Illustrations*

- Systèmes de fichiers



Arbres —*Illustrations*

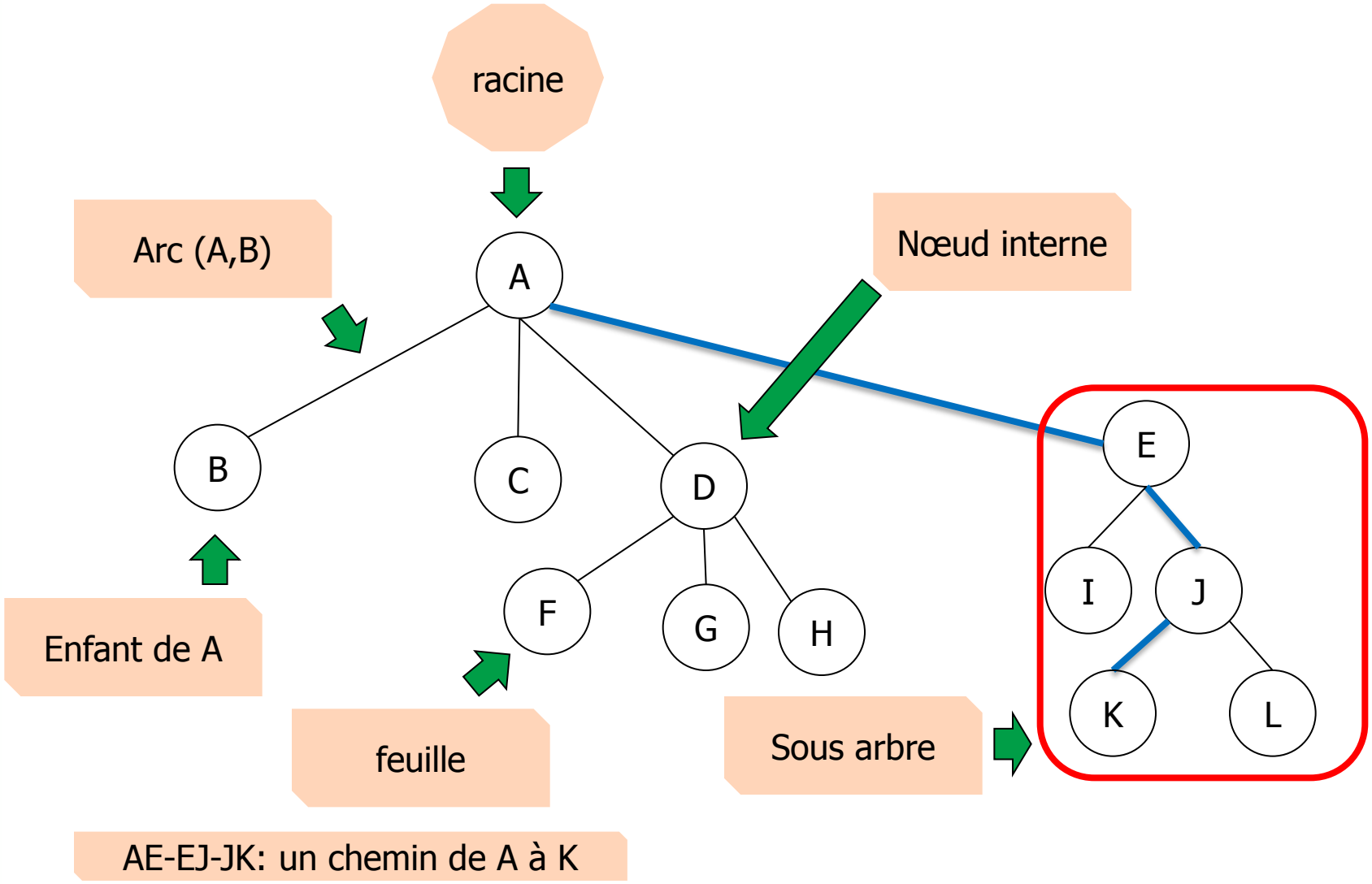
- Représentation des expressions arithmétiques — *l'expression (A+B)*C-(D+E*F)/6 peut être représenté par l'arbre*



Arbres — *Définition*

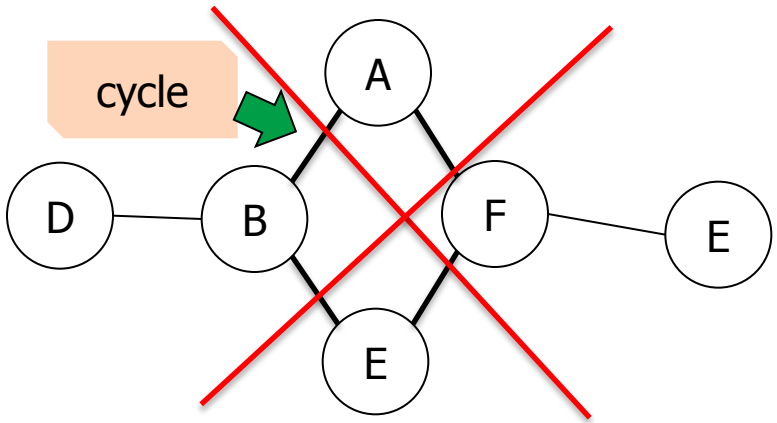
- Un arbre est rassemblement d'informations dont chacune est reliée à d'autres hiérarchiquement.
- Un arbre est constitué de:
  - Un ensemble de **nœuds** — *chaque nœud représente les informations d'un élément de la hiérarchie.*
  - Un ensemble d'**arcs** reliant les nœuds— *chaque arc met en évidence une relation entre deux nœuds.*
  - Un nœud n1 est appelé **père** du nœud n2 si n2 descend immédiatement de n1. Dans ce cas n2 est appelé **fils** de n1.
- Trois types de nœuds
  - Racine — *un nœud qui n'a pas de père.*
  - Feuille — *un nœud* qui n'a pas de fils.
  - Nœud interne — *un nœud* qui n'est ni une feuille ni une racine.
- Un nœud interne constitue la racine d'un sous arbre
- Un chemin entre deux nœuds n1 et n2 est une succession d'arcs de père en fils qui relient entre n1 et n2.

Arbres — Exemple

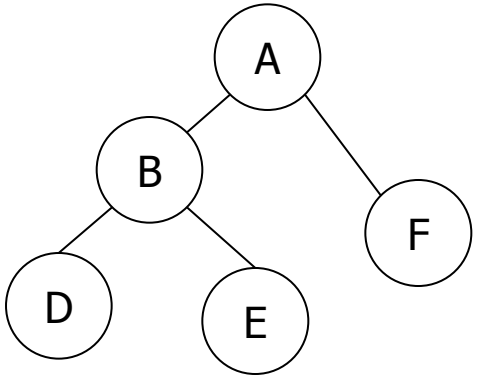


Arbres — *Définition*

- Un arbre ne contient pas de ***cycles***



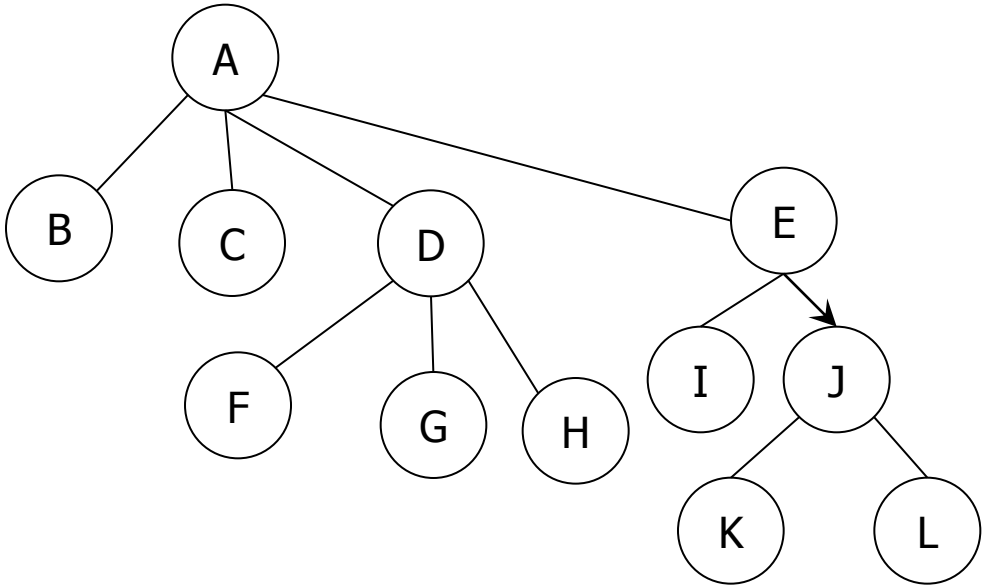
N'est pas un arbre



Arbre

Arbres —*Degré*

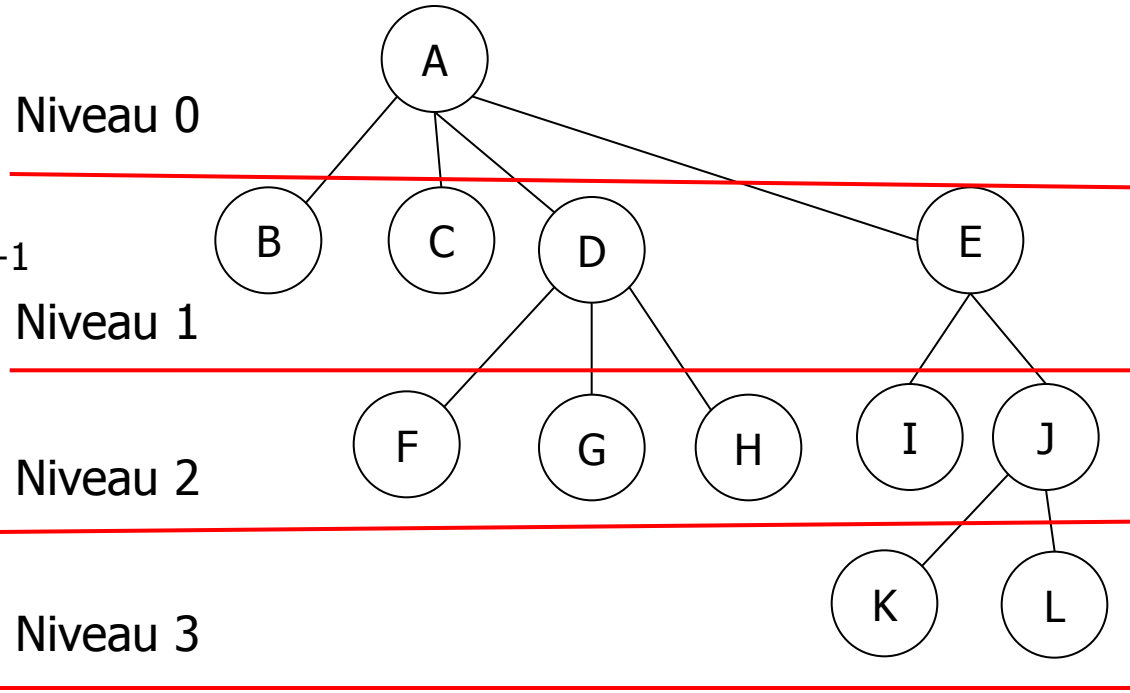
- Degré d'un nœud — *le nombre de ses enfants.*
- Degré d'un arbre — *le plus grand degré de ses nœuds*.
- Exemples
  - Degré(A)=4
  - Degré(B)=0
  - Degré(J)= 2
  - Degré(D)=3
  - Le degré de l'arbre est 4



- Remarque — *le degré d'une feuille est nul*

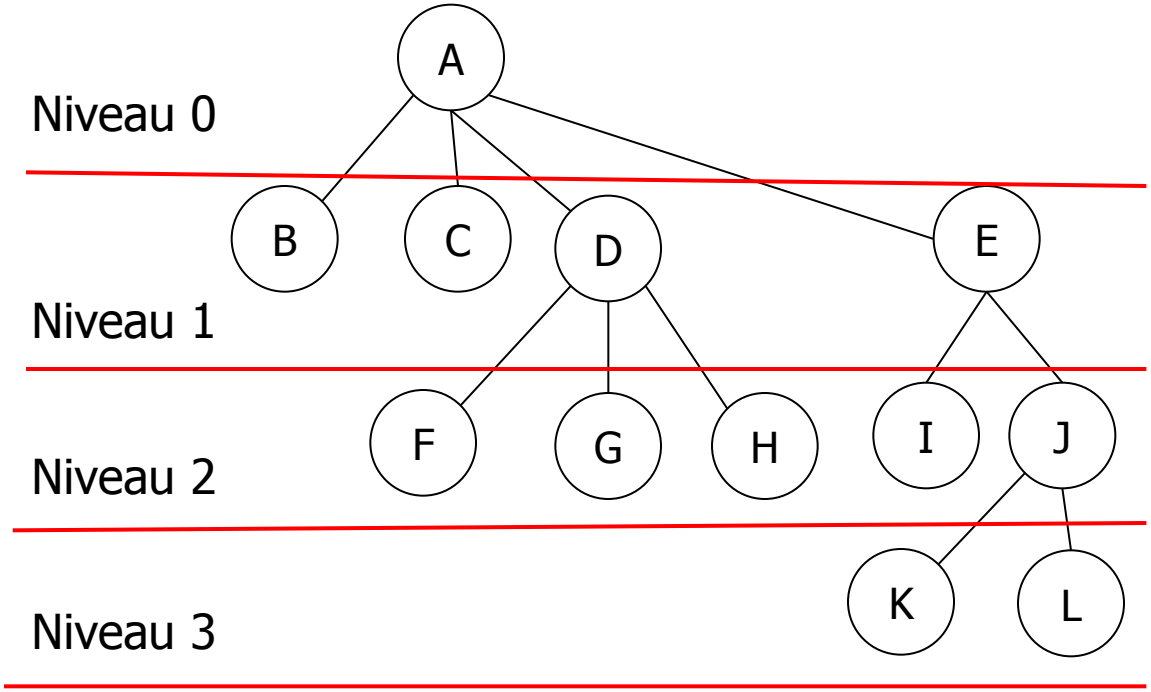
Arbres — *Niveau d'un nœud*

- Niveau — *le nombre d'arcs qu'il faut remonter pour arriver à la racine.*
- Définition récursive
  - le niveau de la racine est nul
  - le niveau d'un nœud est le niveau du parent +1.
- Exemples
  - Niveau(A)=0
  - Niveau(L)=Niveau(J)+1
    =Niveau(E)+1+1
    =Niveau(A)+1+1+1
    =3



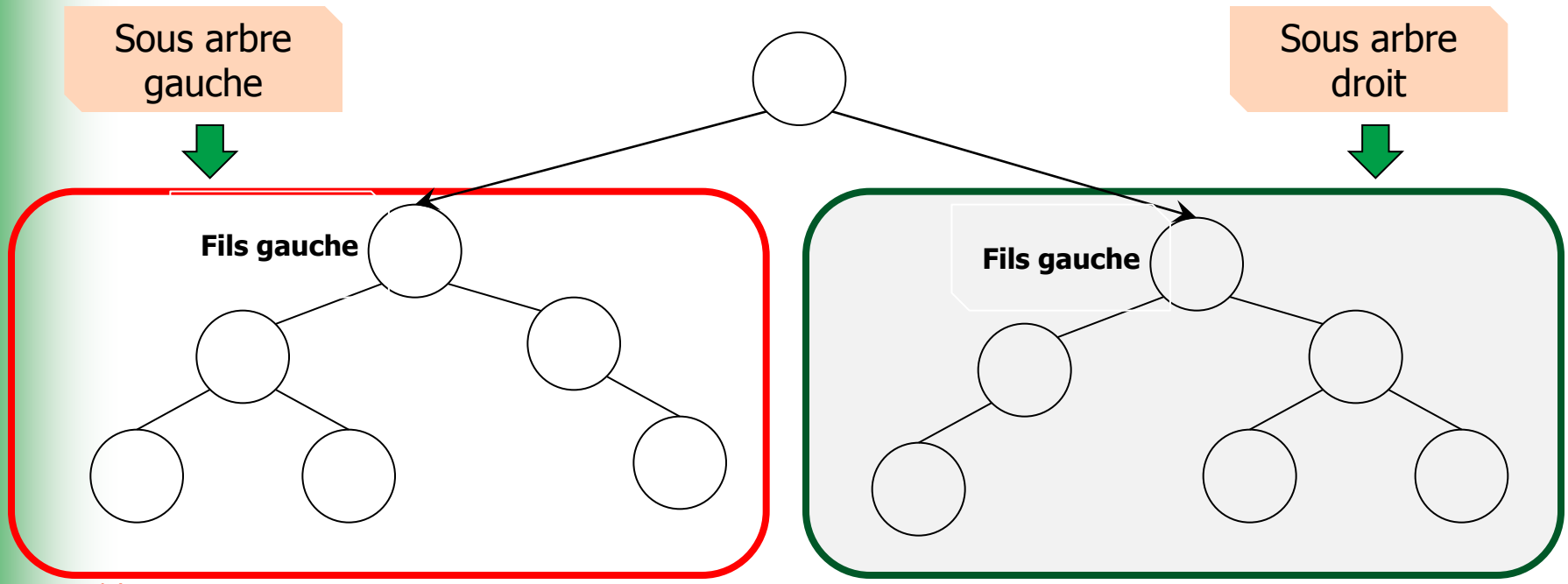
Arbres —*Hauteur d'un arbre*

- le niveau du nœud le plus profond de l'arbre
- la hauteur de l'arbre suivant est 3.



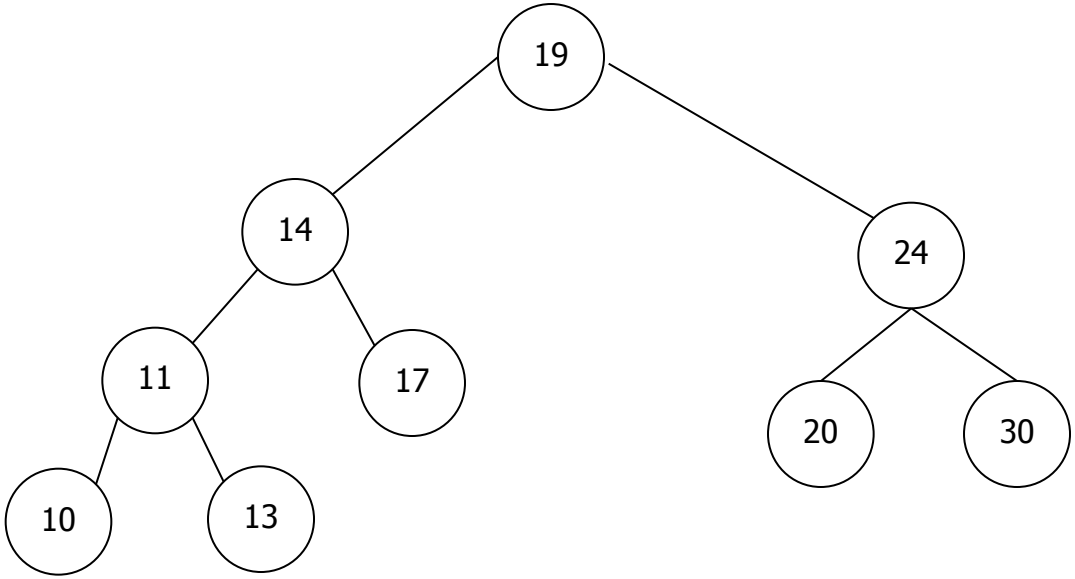
Arbres Binaires

- Un arbre binaire est un arbre dont chaque nœud a deux fils au maximum (fils gauche et fils droit)
- Un arbre binaire est défini par:
  - Une racine
  - Une branche de gauche.
  - Une branche de droite.



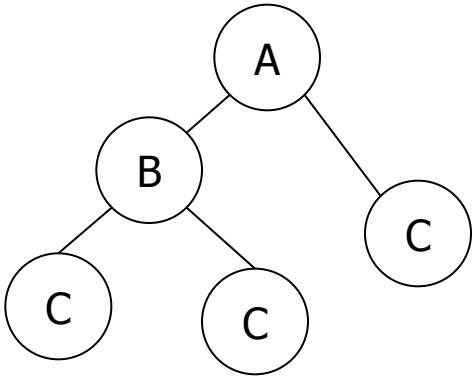
Arbres Binaires de Recherche (ABR)

- Arbre binaire de recherche —*la valeur d'un nœud est supérieure ou égale à celle du nœud de gauche et inférieure à celle du nœud de droite.*
- Exemple — *L'arbre suivant est un arbre binaire de recherche*

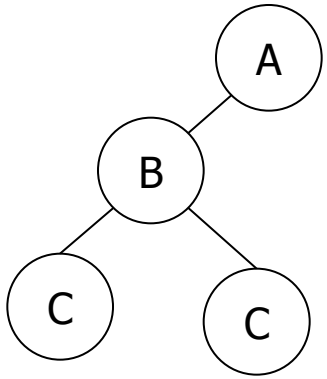


Arbres Binaires Balancés

- Arbre balancé (B-arbre)— *tous les nœuds sont de degré 2 ou 0 (pas 1).*



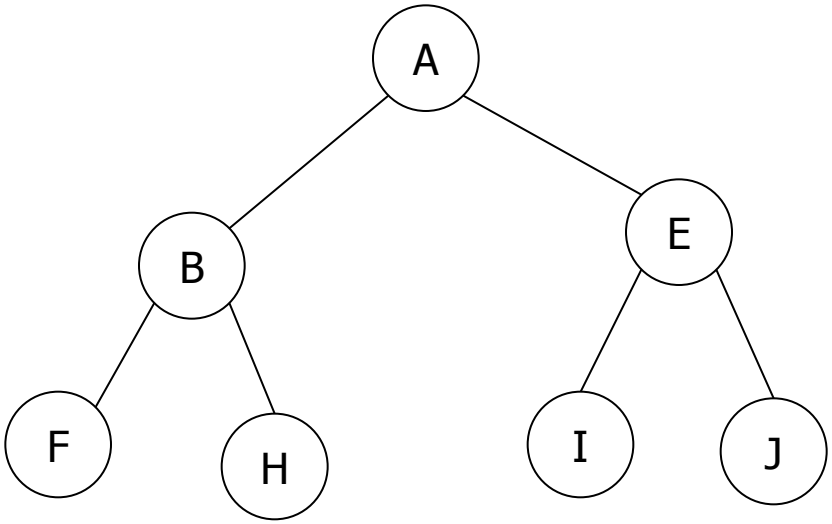
un arbre balancé



un arbre non balancé

Arbres Binaires complets

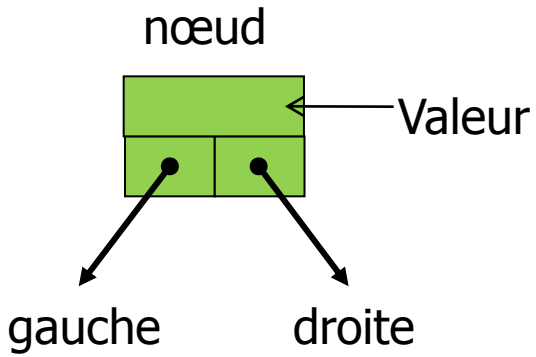
- Un arbre complet — *arbre binaire balancé où tous les niveaux sont de degré 2, sauf le dernier qui est de degré 0.*



ARBRES BINAIRES — *Représentation*

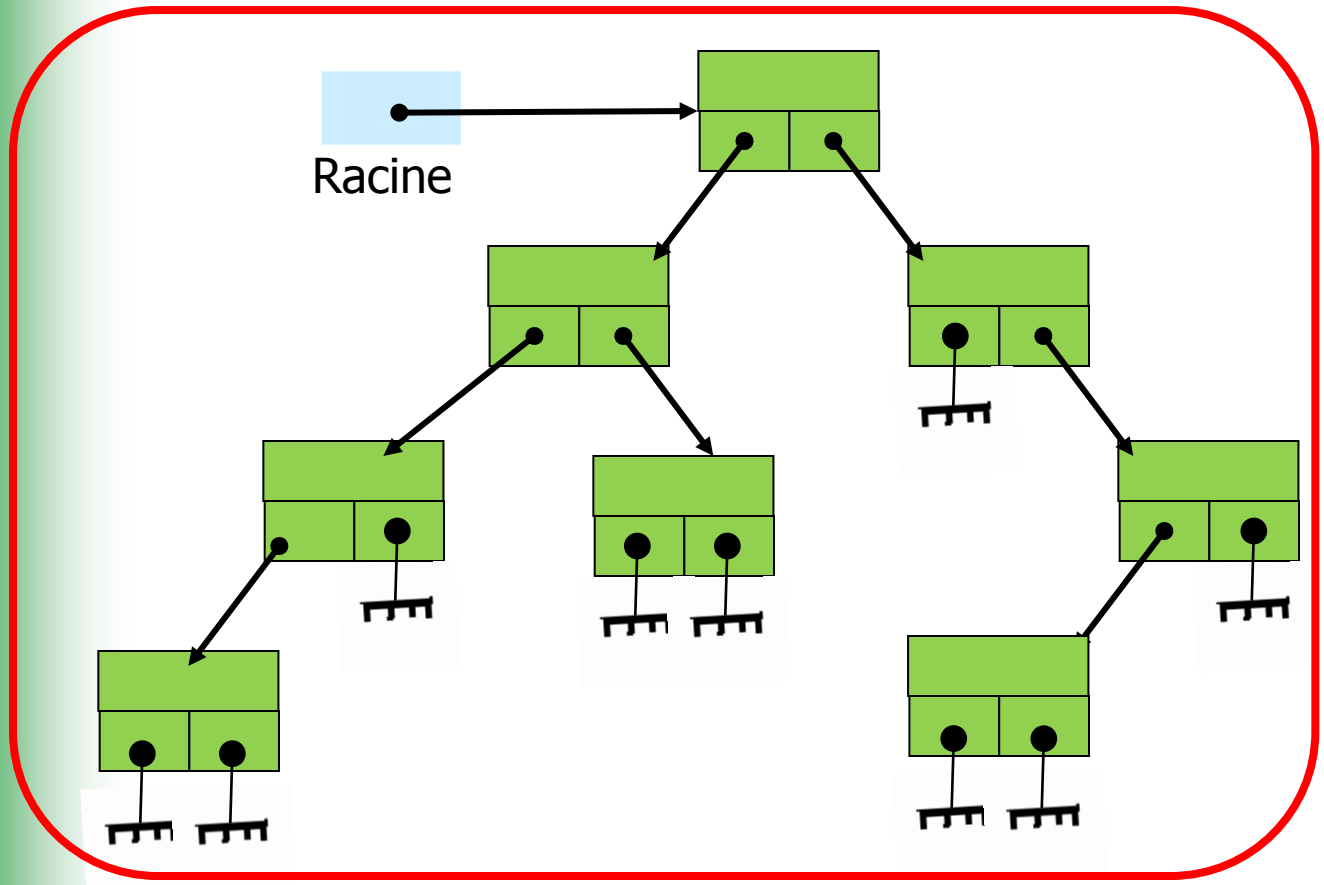
- Un arbre est une suite de nœuds distribués et organisés hiérarchiquement dans la mémoire.

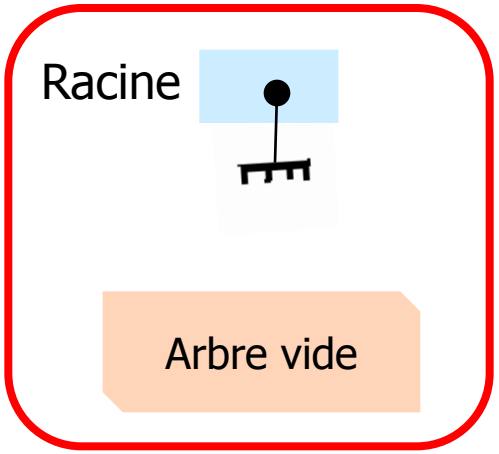
- Un nœud de l'arbre est défini par trois champs :
  - *Valeur* — les données de l'élément .
  - *Gauche* — un lien (un pointeur) vers son sous arbre de gauche.
  - *Droite* — un lien (un pointeur) vers son sous arbre de droite.



Arbres Binaires— *Représentation*

- Un arbre vide est un arbre dont la racine est nulle.
- Un arbre non vide est un pointeur sur la racine.

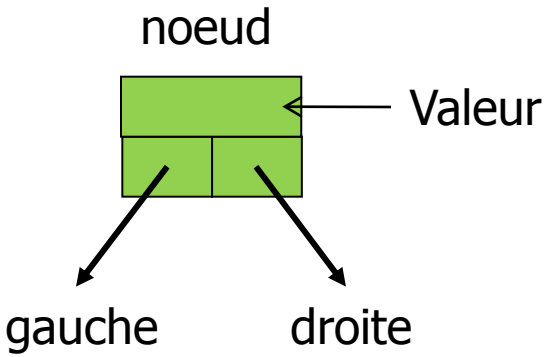




Arbres Binaires— *Représentation*

- Définition d'un arbre d'entiers

```
/* type noeud*/
  typedef struct Noeud{
  int valeur;
      struct Noeud * gauche;
      struct Noeud * droite;
  } noeud;
/* type arbre */
typedef struct noeud* ARBRE;
```



Arbres Binaires— *Initialisation*

- Déclaration d'un arbre d'entiers

```
ARBRE racine;
```

  - Racine pointe sur le nœud racine.

- Initialisation d'un arbre
  - la **Racine** de l'arbre est mise à NULL.
  - la fonction retourne une valeur de type **ARBRE** .

```
ARBRE InitArbre()
{
        ARBRE Racine;
        Racine = NULL; /* initialisation*/
         return Racine;
}
```

Arbres Binaires — *Arbres vide ?*

- Arbre vide ?
  - *Teste si la* **racine** *de l'arbre est à NULL*
  - *La fonction retourne 1 si l'arbre est vide et 0 sinon*

```
int EstArbreVide(ARBRE racine )
{
        if(racine == NULL  )
                return 1;
        return 0;
}
```

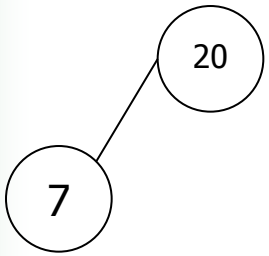
Arbres Binaires— *Création*

- La création d'un arbre
  - se fait à partir
    - de la valeur de la racine.
    - d'un sous arbre de gauche.
    - d'un sous arbre de droite.
  - Se déroule en trois étapes:
    - Création et réservation de la mémoire à un nœud.
    - Remplissage du nœud avec sa valeur.
    - Branchement du nœud avec les sous arbres.

```
ARBRE creerArbre(int V, ARBRE ag, ARBRE ad) {
        noeud *m;
        m=(noeud*)malloc(sizeof(noeud));
        if(m!=NULL){
                m->valeur = V;
                m->gauche= ag;
                m->droite= ad;
      }
      return m;
}
```

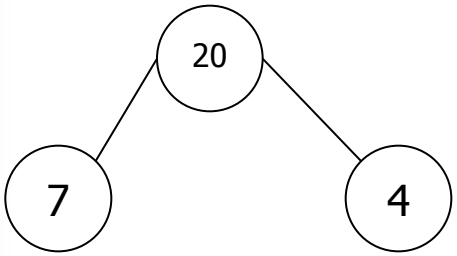
Arbres Binaires— *Création*

- Exemple 1




```
ARBRE racine,g;
racine = NULL;
g= creerArbre(7, NULL, NULL);
racine=creerArbre(20,g, NULL);
```

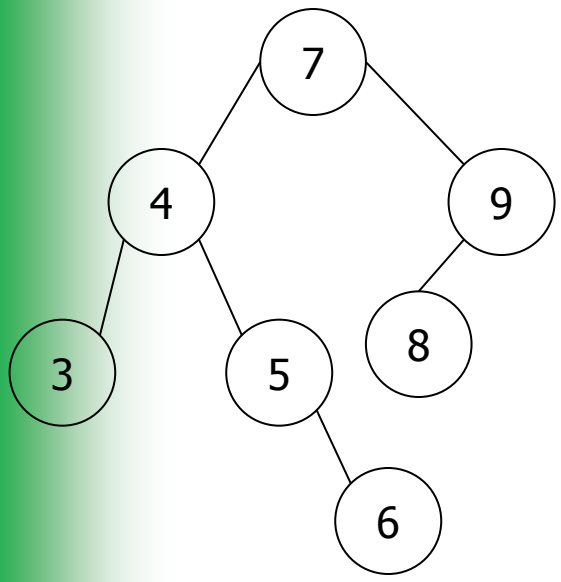
- Exemple 2




```
ARBRE racine,g,d;
racine = NULL;
g= creerArbre(7, NULL, NULL);
d= creerArbre(4, NULL, NULL),
racine=creerArbre(20,g, d);
```

Arbres Binaires— *Création*

- Exemple 3




```
int  main() {
    ARBRE racine,g,d;
    ARBRE A3, A8,A6;
    racine = NULL;
    A3=creerArbre(3,NULL,NULL);
    A6=creerArbre(6,NULL,NULL);
    A8=creerArbre(8, NULL, NULL);
    g=
    creerArbre(4,A3,creerArbre(5,NULL,A6));
    d= creerArbre(9,A8,NULL);
    racine=creerArbre(7,g, d);
}
```

Arbres Binaires —*Desalocation*

- Desalocation des nœuds d'un arbre

```
void DesalouerArbre (ARBRE *Racine)
{
    if (Racine != NULL)
    {
        DesalouerArbre (&(*Racine)->gauche);
        DesalouerArbre (&(*Racine)->droite);
       free(Racine);
       Racine = NULL;
    }
}
```

Arbres Binaires — *Parcours*

- Parcourir un arbre consiste à visiter tous les nœuds de l'arbre.
- Le parcours d'un arbre permet de réaliser plusieurs opérations :
  - L'affichage
  - La recherche
  - L'insertion et la suppression
  - Etc.
- Deux stratégies de parcours :
  - Parcours en largeur — *Exploration niveau par niveau*
  - Parcours en profondeur —*Exploration branche par branche*

Arbres Binaires — *Parcours*

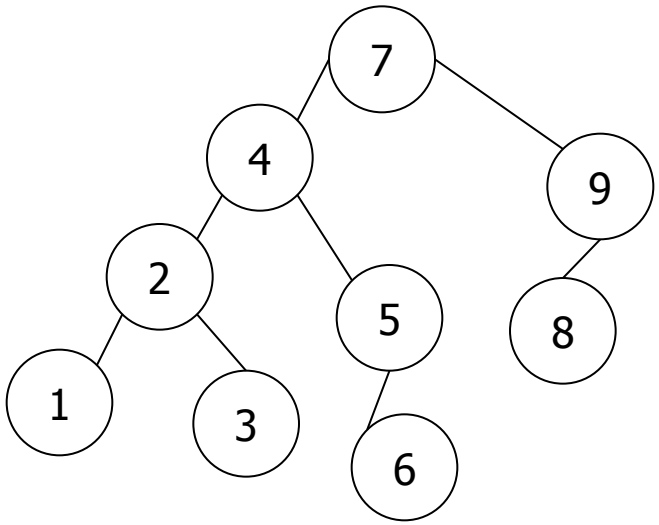
- Trois méthodes de parcours en profondeur:
  - Parcours infixé —in-*ordre*
  - Parcours post fixé —*post-ordre*
  - Parcours préfixé —*pré-ordre*

Arbres Binaires — *Parcours*

- Parcours infixé (Gauche Racine Droite)
  - Parcourir le sous arbre gauche
  - Visiter la racine
  - Parcourir le sous arbre droit

- Exemple — *le parcours de l'arbre suivant donne :*

1 2 3 4 6 5 7 8 9



Arbres Binaires— *Parcours*

- Parcours infixé – Affichage en in-ordre

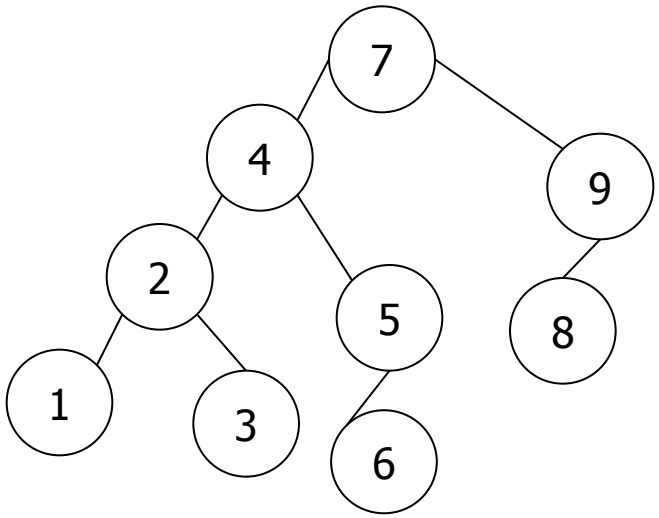
```
void parcoursInfixe(ARBRE racine)
{
        if(racine != NULL  ) {
                parcoursInfixe(racine->gauche);
                printf(''%d'', racine->valeur);
                parcoursInfixe(racine->droite);
        }

}
```

Arbres Binaires — *Parcours*

- Parcours post fixé (Gauche Droite Racine)
  - Parcourir le sous arbre gauche
  - Parcourir le sous arbre droit
  - Visiter la racine

- Exemple — *le parcours de l'arbre suivant donne :*

1 3 2 6 5 4 8 9 7



Arbres Binaires— *Parcours*

- Parcours post fixé -Affichage en post-ordre

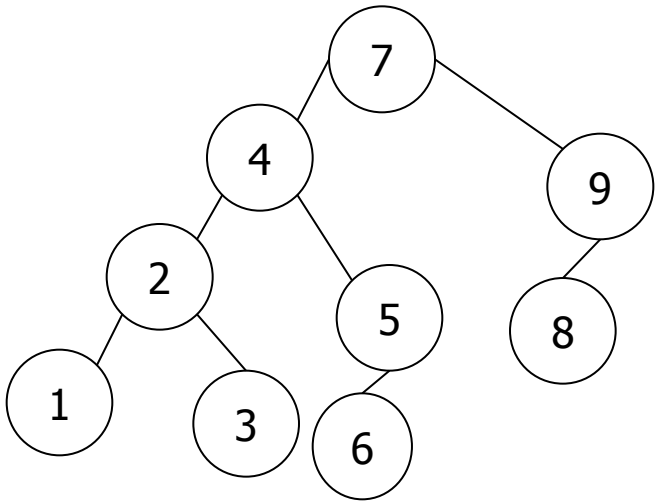
```
void parcoursPostfixe(ARBRE racine)
{
        if(racine != NULL  ) {
                parcoursPostfixe (racine->gauche);
                parcoursPostfixe (racine->droite);
                printf(''%d'', racine->valeur);
        }

}
```

Arbres Binaires — *Parcours*

- Parcours préfixé (Racine Gauche Droite)
  - Visiter la racine
  - Parcourir le sous arbre gauche
  - Parcourir le sous arbre droit

- Exemple — *le parcours de l'arbre suivant donne :*

7 4 2 1 3 5 6 9 8



Arbres Binaires — *Parcours*

- Parcours préfixé -Affichage en pré-ordre
-

```
void parcoursPrefixe(ARBRE racine)
{
        if(racine != NULL  ) {
                printf(''%d'', racine->valeur);
                parcoursPrefixe(racine->gauche);
                parcoursPrefixe(racine->droite);
        }

}
```

Arbres Binaires — *Recherche*

- La fonction retourne
  - NULL si l'arbre est vide ou la valeur n'existe pas
  - Un pointeur sur le nœud qui contient la valeur sinon Parcourir.
- Cas d'un arbre non ordonné

```
noeud* Rechercher(ARBRE racine, int v) {
        noeud* ptr;ptr=NULL;
        if(racine != NULL) {
                if (racine->valeur == v) ptr = racine;
                else {
                  ptr = Rechercher(racine->gauche)
                  if (ptr == NULL)
                          ptr = Rechercher(racine->droite);
                 }
        }
        return prt;
}
```

Arbres Binaires — *Recherche*

- La fonction retourne
  - NULL si l'arbre est vide ou la valeur n'existe pas
  - Un pointeur sur le nœud qui contient la valeur sinon Parcourir.
- Cas d'un arbre ordonné

```
noeud* RechercherOrd(ARBRE racine, int v)
{
        noeud* ptr;
        if(arbreEstvide(racine)) return NULL;

         if (racine->valeur == v) return racine;

         if (racine->valeur >= v)
               return RechercherOrd (racine->gauche,v)

        return RechercherOrd (racine->droite, v)
}
```

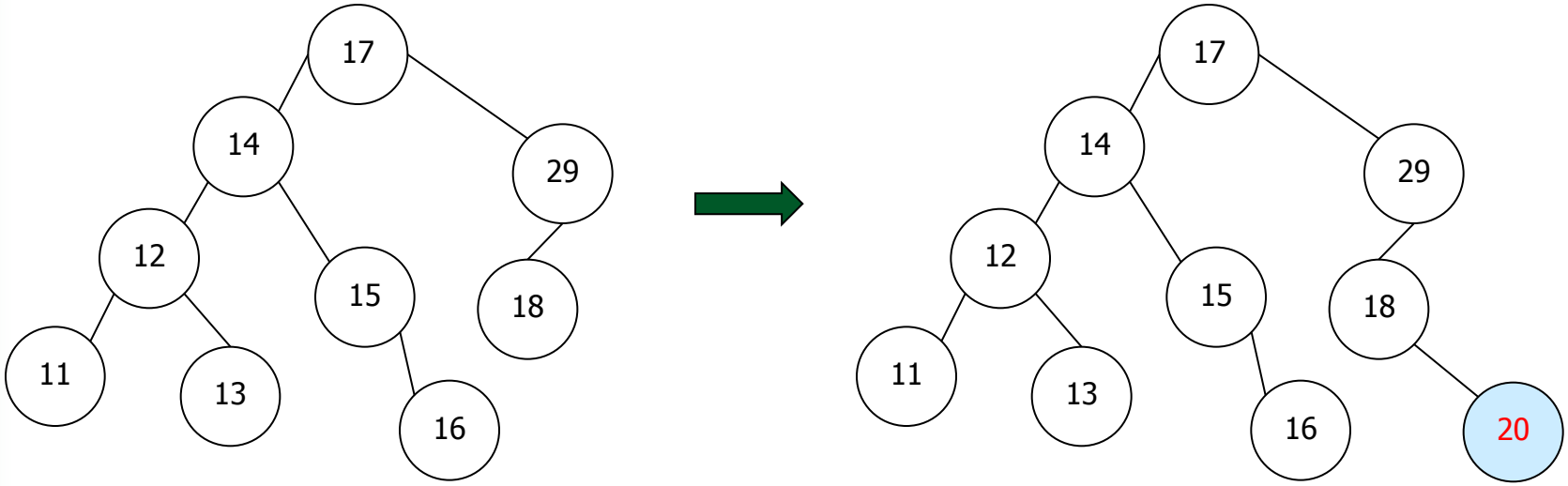
Arbres Binaires — *Insertion*

- Ajout d'un élément aux feuilles d'un arbre ordonné
  - Recherche de la place de l'élément
  - Insertion de l'élément

```
void insererElem(ARBRE* racine, int v)
{
 if(arbreEstvide(*racine))
                *racine = creeArbre(v,NULL,NULL)
 else{
      if((*racine)->valeur >= v)
                insererElem(&((*racine)->gauche),v)

        else
                 insererElem((&((*racine)->droite),v);
  }
}
```

Arbres Binaires — *Insertion*

- Exemple — l'insertion de 20 modifiera l'arbre comme suit:



Arbres Binaires —*nombre de feuilles*

- Compter le nombre de feuilles dans un arbre.
  - Si le nœud n'a pas de fils, on renvoie 1.
  - Si le nœud il a un fils droite , on ajoute le nombre de feuilles du sous arbre droit.
  - Si le nœud il a un fils gauche, on ajoute le nombre de feuilles du sous arbre gauche.

```
int compteFeuilles(ARBRE Racine){
  int retour=0;

  if (Racine ==NULL)     return 0;
  if ((Racine->droite==NULL)&&(Racine->gauche ==NULL)) return 1;
  if (Racine->gauche!= NULL)
       retour +=compteFeuilles (Racine-> gauche);
  if (Racine->droite!= NULL)
       retour +=compteFeuilles (Racine->droite);

  return retour;
}
```