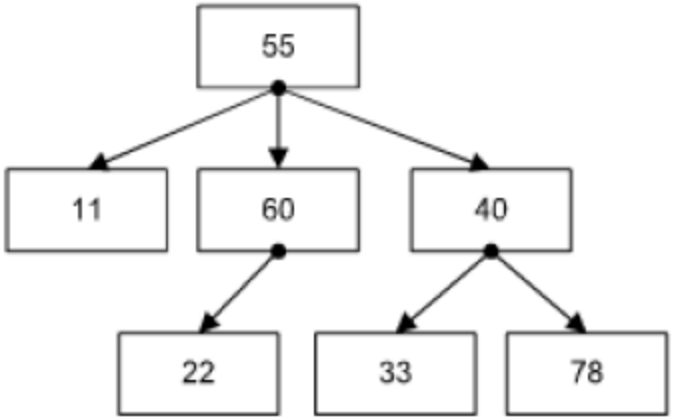
Chapitre6. LES ARBRES

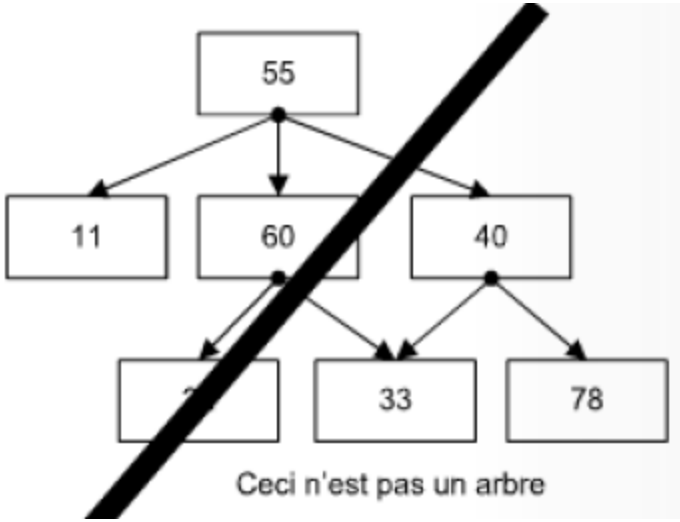
- 1. DÉFINITION
- 2. CONSTRUCTION D'UN ARBRE PAR UN TABLEAU
- 3. CONSTRUCTION D'UN ARBRE PAR UNE LISTE CHAINEE
- 4. ARBRES BINAIRES
- 5. LES PARCOURS D'ARBRE
- 6. ARBRE BINAIRE DE RECHERCHE

1. DÉFINITION

- Un arbre est une structure composée d'éléments appelés noeuds
- Un arbre, appelé aussi arbre N-aire, chaque nœud possède au maximum N nœuds
- La représentation d'un arbre en informatique se fait à l'envers : la racine se trouve en haut et les branches se développe vers le bas



Ceci est un arbre 3-aire



Ceci n'est pas un arbre

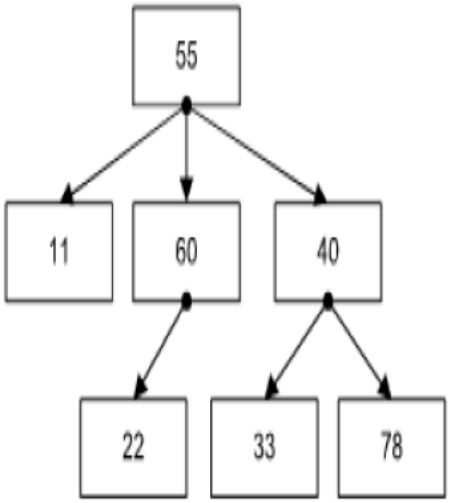
1. DÉFINITION

 Un arbre est constitué d'un noeud particulier appelé racine et d'une suite ordonnée éventuellement vide A1, A2, …, Ap d'arbres disjoints appelés sous-arbres de la racine

 Un arbre contient donc au moins un nœud : sa racine. Tous les autres nœuds suivent directement ou indirectement la racine

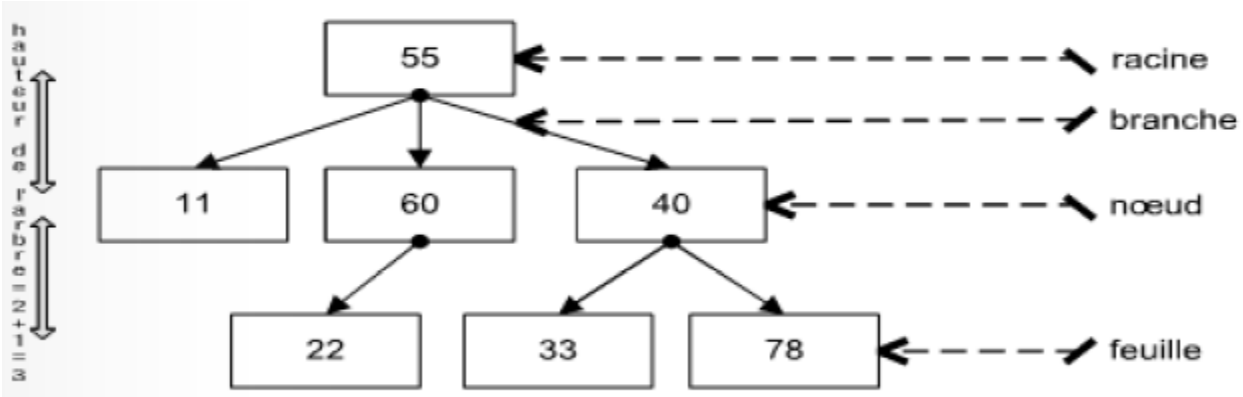
 La figure suivante représente l'arbre d'une descendance

- la racine de cet arbre contient 55
- il est constitué de trois sous-arbres
  - ✓ A1 : de racine 11, est réduit à 11
  - ✓ A2 : de racine 60, contient 60 et 22
  - ✓ A3 : de racine 40, contient 40, 33 et 78



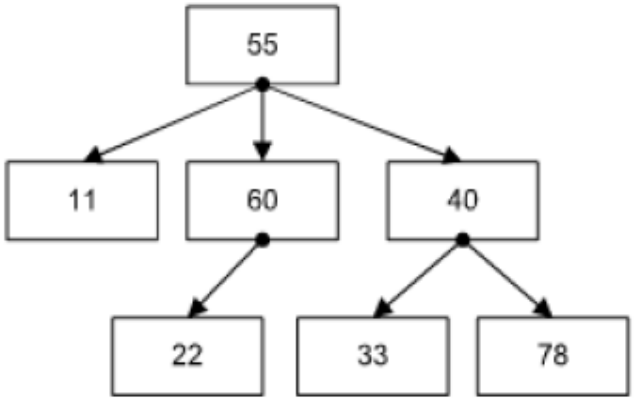
1. DÉFINITION

- Un nœud est aussi appelé sommet, contient un élément et indique les nœuds suivants
- les fils d'un noeud sont les racines de ses sous-arbres
- une feuille d'un arbre est un noeud sans fils (qui n'a pas de suivant); 11, 22,33 et 78 sont des feuilles
- Une branche est un chemin qui rejoint deux nœuds
- la hauteur d'un noeud est égale au nombre de branches le séparant de la feuille la plus éloignée plus un
- la profondeur d'un nœud est égale au nombres de branches le séparant de la racine
- la hauteur d'un arbre vaut la hauteur de la racine



2. CONSTRUCTION D'UN ARBRE PAR UN TABLEAU

- Chaque nœud de notre exemple possède au plus trois nœuds fils
- On construit un tableau à deux dimensions où chaque indice représente un nœud
- Pour connaître les suivants de chaque nœud, il suffit de voir leur indice dans les cases ad-hoc du tableau
- L'absence d'un nœud suivant est représenté par la valeur -1



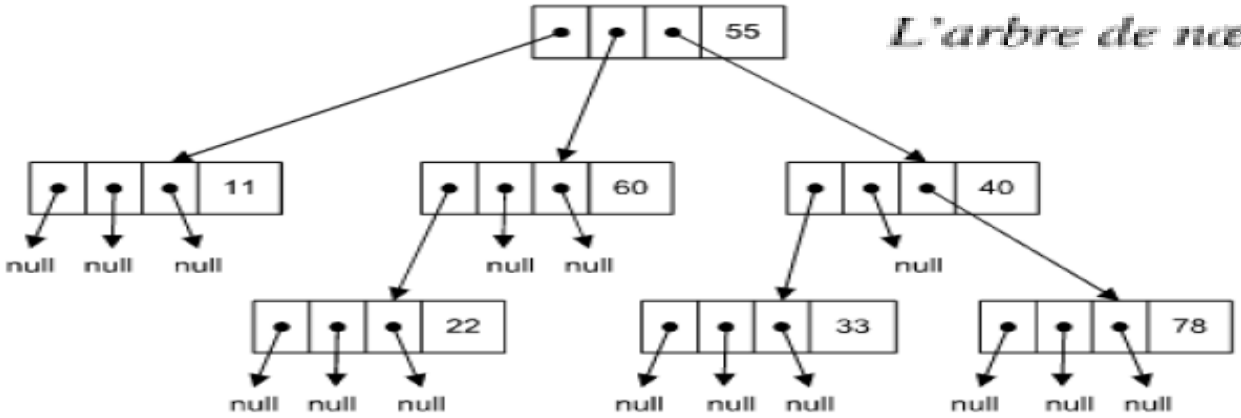
Numéro	[0]	[1]	[2]	[3]	[4]	[5]	[6]
Valeur	55	11	60	40	22	33	78
Suivant 1	1	-1	4	5	-1	-1	-1
Suivant 2	2	-1	-1	-1	-1	-1	-1
Suivant 3	3	-1	-1	6	-1	-1	-1

- Cette construction est compliquée à gérer, et de ce fait, elle n'est pas très utilisée

3. CONSTRUCTION D'UN ARBRE PAR UNE LISTE CHAINEE

- Chaque nœud de notre exemple possède au plus trois nœuds fils
- Un nœud pouvant référencer jusqu'à 3 nœuds suivants : 3 attributs gauche, milieu et droite ou dans un tableau de 3 éléments
- Nous allons introduire la structure suivante:

```
typedef struct Noeud { int valeur
                       struct Nœud *gauche
                       struct Nœud *milieu
                       struct Nœud *droite
                        } NoeudEntier
```



L'arbre de nœuds.

3. CONSTRUCTION D'UN ARBRE PAR UNE LISTE CHAINEE

♦ Il est possible décrire l'arbre avec une écriture standard utilisant les parenthèses : chaque sous arbre est représenté par la racine, suivie de ses suivants entre parenthèses

- Chaque sous arbres est lui-même un arbre qui utilise la même notation

(55(sous arbre gauche, sous arbre milieu, sous arbre droit))

sous arbre gauche s'ecrit : (11)

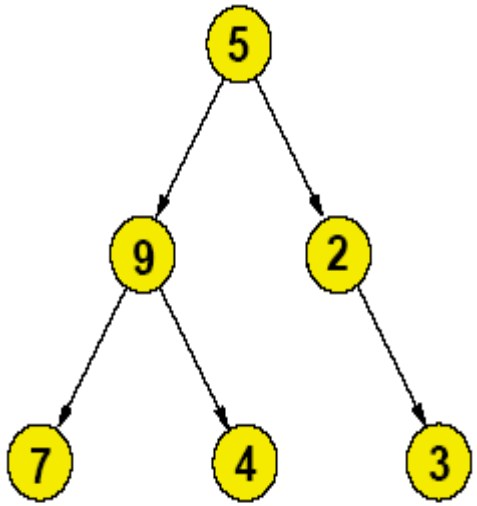
sous arbre milieu s'ecrit : (60(22))

sous arbre droit s'écrit : (40(33,78))

Ce qui donne : (55(11,60(22),40(33,78)))

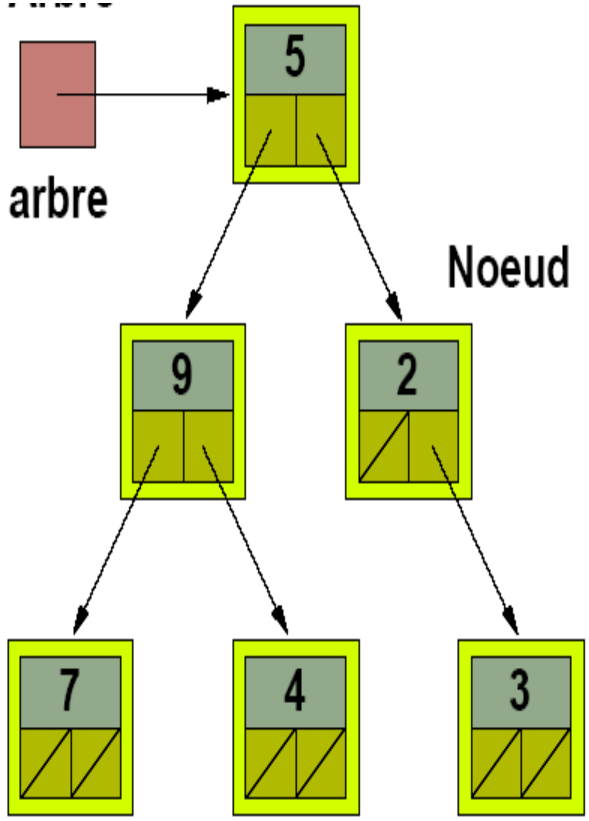
4. ARBRES BINAIRES

- Un arbre binaire est un arbre 2-aire : chaque noeud possède 0, 1 ou 2 suivants
- Chaque arbre binaire peut posséder un arbre binaire droit et arbre binaire gauche dont la racine est respectivement son fils droit et son fils gauche
- Pour coder un arbre binaire, on fait correspondre à chaque nœud :
  - une structure contenant la donnée et deux adresses, une adresse pour chacun des deux noeuds fils
  - avec la convention qu'une adresse nulle indique un arbre binaire vide
  - mémoriser l'adresse de la racine pour pouvoir reconstituer tout l'arbre.



4. ARBRES BINAIRES

```
typedef struct noeud {
        int valeur;
        struct noeud *sag;
        struct noeud *sad; } Noeud;
```

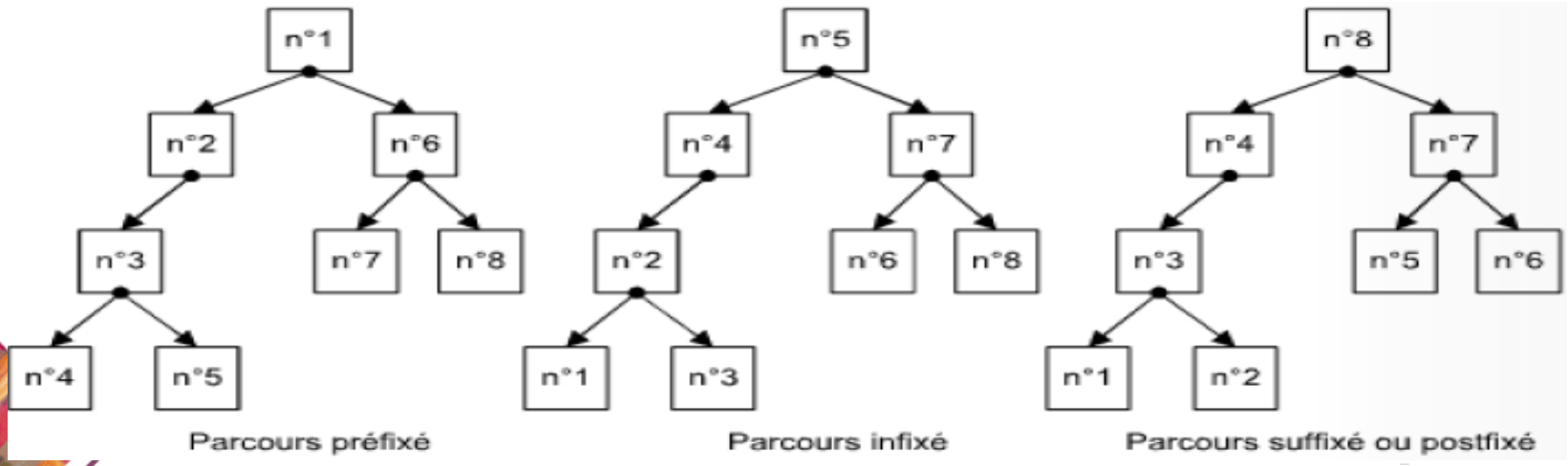


5. LES PARCOURS D'ARBRE

- Il existe 4 techniques pour parcourir l'ensemble des valeurs d'un arbre
- 4 algorithmes implémentant ces différents parcours (récursifs)

Parcours en profondeur

- Trois algorithmes récursifs simple permettent le parcours en profondeur d'un arbre binaire : préfixé, infixé et postfixé
- Tous les nœuds de l'arbre sont atteints branche par branche dans toute leur profondeur



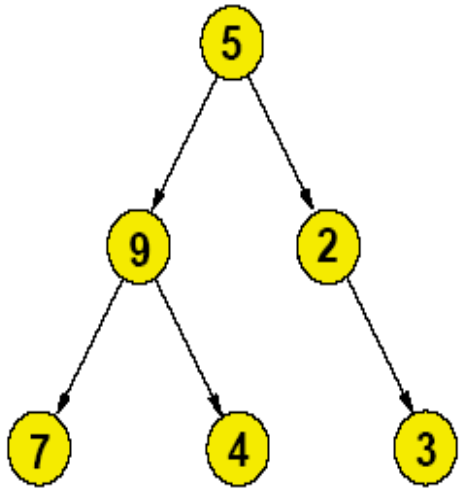
5. LES PARCOURS D'ARBRE

Parcours en profondeur

parcours préfixé : *le traitement se fait avant la visite des sous arbres*

```
void parcoursPrefixe(Nœud *ar) {
    printf("%d ", ar->valeur); //traitement
    if (ar->sag !=NULL) parcoursPrefixe(ar->sag); // appel récursive
    if (ar->sad !=NULL) parcoursPrefixe(ar->sad); // appel récursive
}
```

Le parcours préfixé donne : 5 9 7 4 2 3



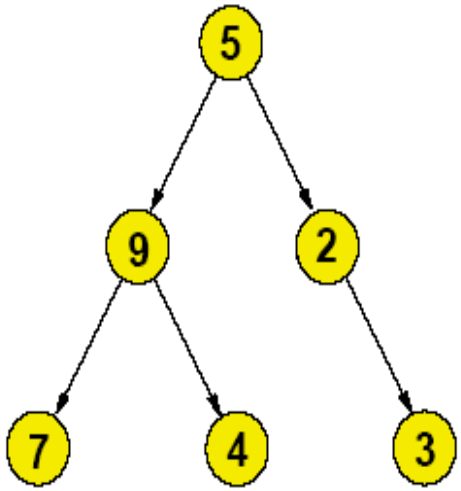
5. LES PARCOURS D'ARBRE

Parcours en profondeur

parcours infixé : *le traitement se fait entre les deux visites des sous arbres*

```
void parcoursInfixe(Nœud *ar) {
        if (ar->sag !=NULL) parcoursInfixe(ar->sag); // appel récursive
        printf("%d ", ar->valeur); //traitement
        if (ar->sad !=NULL) parcoursInfixe(ar->sad); // appel récursive
}
```

Le parcours infixé donne : 7 9 4 5 2 3



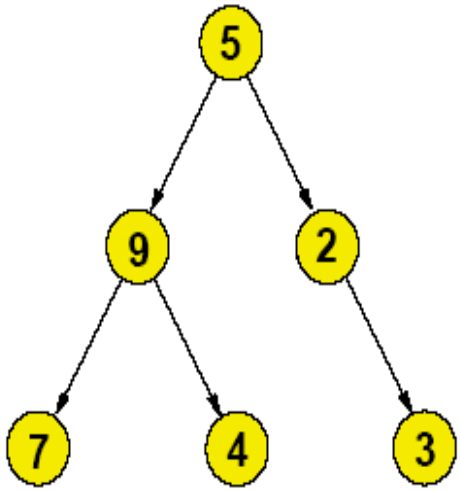
5. LES PARCOURS D'ARBRE

Parcours en profondeur

parcours postfixé (suffixé) : ***le traitement se fait après la visite des sous arbres***

```
void parcoursPostfixe(Noeud *ar) {
        if (ar->sag !=NULL) parcoursPostfixe(ar->sag); // appel récursive
        if (ar->sad !=NULL) parcoursPostfixe(ar->sad); // appel récursive
        printf("%d ", ar->valeur); //traitement
}
```

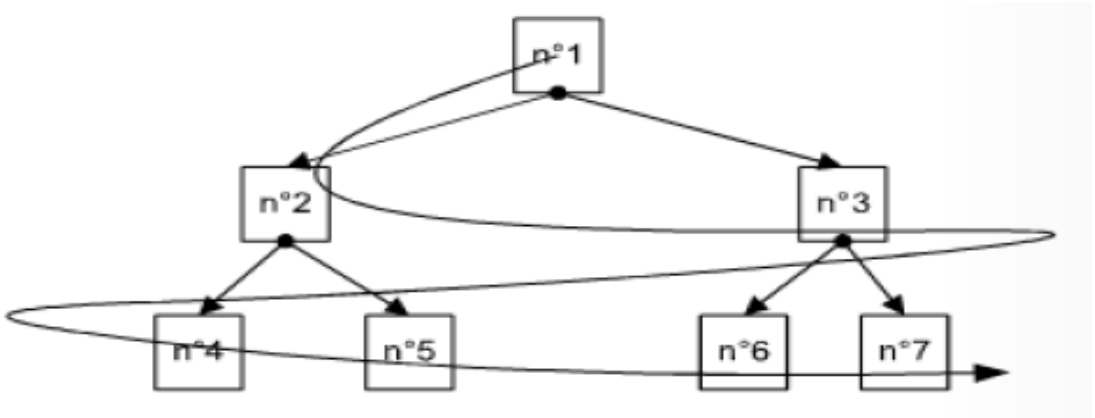
Le parcours postfixé donne : 7 4 9 3 2 5



5. LES PARCOURS D'ARBRE

Parcours en largeur

- Tous les nœuds de l'arbre sont atteints depuis la racine, puis couche par couche de gauche à droit
- Pour écrire cette méthode, il faut introduire une liste d'arbres où seront stockés les nœuds au fur et à mesure de leur passage
- Il faut ajouter en tête et retirer en queue : une file (FIFO) aurait même suffit

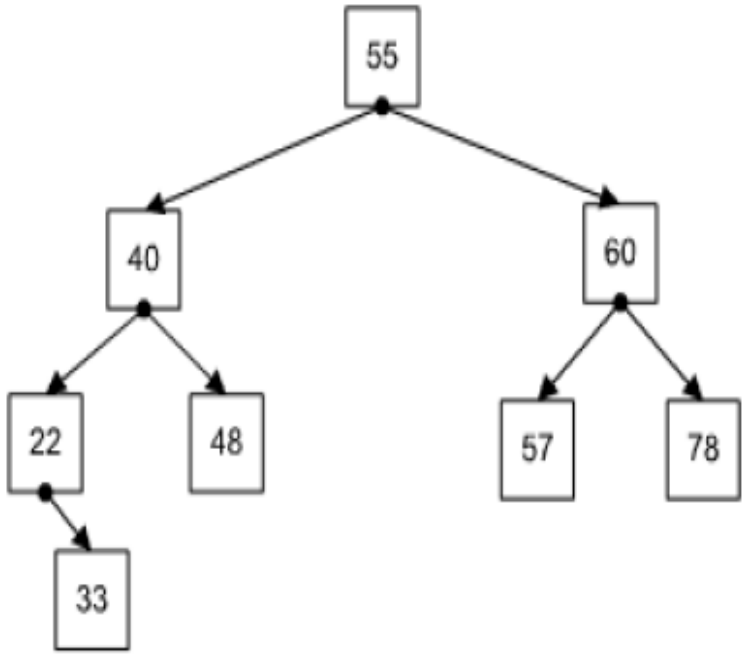


6. ARBRE BINAIRE DE RECHERCHE

♦ Un arbre binaire de recherche (ABR), appelé aussi arbre binaire ordonné, est un arbre tel que la valeur de chaque nœud est supérieure à celle du sous arbre gauche et inférieure à celle du sous arbre droit

♦ Chaque valeur n'est stockée qu'une seule fois dans l'arbre

```
typedef struct noeud {
        int valeur;
        struct noeud *fg;
        struct noeud *fd; } Noeud;
```



6. ARBRE BINAIRE DE RECHERCHE

Initialiser l'arbre binaire ABR

Cette fonction initialise les valeurs de la structure représentant l'arbre pointé par a, afin que celui-ci soit vide : mettre le pointeur sur la racine égal à NULL.

```
void initialiser(Noeud *a) {
a = NULL;
}
```

6. ARBRE BINAIRE DE RECHERCHE

Préparer un nœud

- Cette fonction alloue un nouveau noeud et place l'élément (valeur) e à l'intérieur
- Ses deux fils sont initialisés à la valeur NULL
- La fonction retourne l'adresse de ce nouveau noeud
- Au cas où, l'allocation de mémoire échoue, NULL est renvoyée

```
Noeud *preparerNoeud(int e) {
Noeud *n;
if ((n =(Noeud*)malloc(sizeof(Noeud)))==NULL) return NULL;
n->valeur = e;
n->fg = NULL;
n->fd = NULL;
return n;
}
```

6. ARBRE BINAIRE DE RECHERCHE

Ajouter un noeud

Fonction ajoute le noeud pointé par v dans l'arbre pointé par b

- Parcourir l'arbre à partir de la racine pour descendre jusqu'à l'endroit où sera inséré le noeud
- On prendra à gauche si la valeur du noeud visité est supérieure à la valeur du noeud à insérer
- On prendra à droite si la valeur du noeud visité est inférieure
- En cas d'égalité, l'insertion ne peut pas se faire et la fonction retourne -1.
- On arrête la descente quand le fils gauche ou droit choisi pour descendre vaut NULL. Le noeud pointé par v est alors inséré à ce niveau.

6. ARBRE BINAIRE DE RECHERCHE

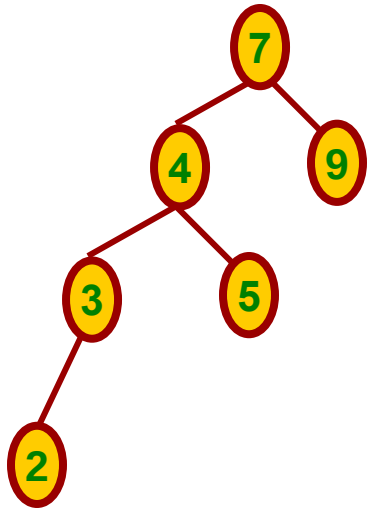
Ajouter un noeud

Fonction itérative ajoute le noeud pointé par v dans l'arbre pointé par b

```
int ajouterNoeud_iterative(Noeud *b, Noeud *v) {
Noeud *a,*s;
int gauche=0, droite=0;
a=b; s=b;
while (a!=NULL) {    gauche=0; droite=0;
                     if (v->valeur < a->valeur) {s=a; a=a->fg; gauche=1; }
                     else if (v->valeur > a->valeur) { s=a; a=a->fd; droite=1; }
                          else return -1;}
  if (gauche==1) s->fg=v;
  if (droite==1) s->fd=v;
  return 0;
 }
```

Exercice : Ecrire la fonction récursive qui ajoute le noeud pointé par v dans l'arbre pointé par b

6. ARBRE BINAIRE DE RECHERCHE




```
main(){
int v;
Noeud *racine;
initialiser(racine);
racine=preparerNoeud(7);
do {
    printf(" donner v "); scanf("%d",&v);
    if (v!=0) ajouterNoeud_iterative(racine,preparerNoeud(v));
}  while (v!=0);
printf(" parcoursPostfixe : "); parcoursPostfixe(racine); printf("\n");
printf(" parcoursPrefixe : "); parcoursPrefixe(racine); printf("\n");
printf(" parcoursInfixe : "); parcoursInfixe(racine); printf("\n");
system("pause");
}
```

```
parcoursPostfixe : 2 3 5 4 9 7
parcoursPrefixe : 7 4 3 2 5 9
parcoursInfixe : 2 3 4 5 7 9
```

6. ARBRE BINAIRE DE RECHERCHE

Suppression d'un noeud

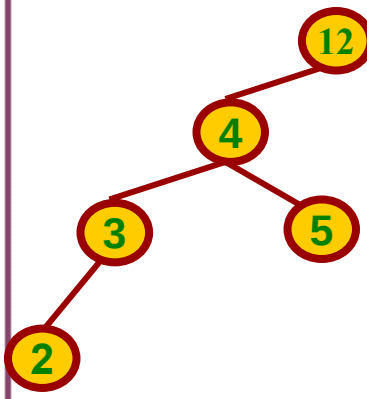
- Il y a trois cas possibles :
  - Le nœud est terminal (feuille)
  - Le nœud a un seul descendant
  - Le nœud a deux descendants

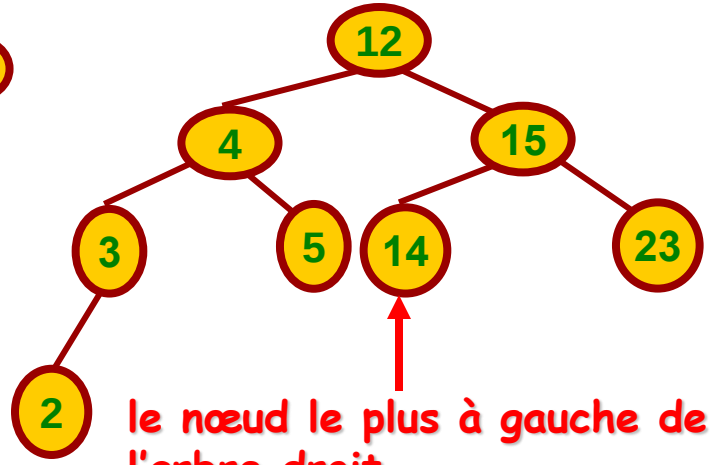
- Pour le dernier cas on remplace le nœud à supprimer par :
  - Le nœud le plus à droit de son arbre gauche
  - Le nœud le plus à gauche de son arbre droit

6. ARBRE BINAIRE DE RECHERCHE

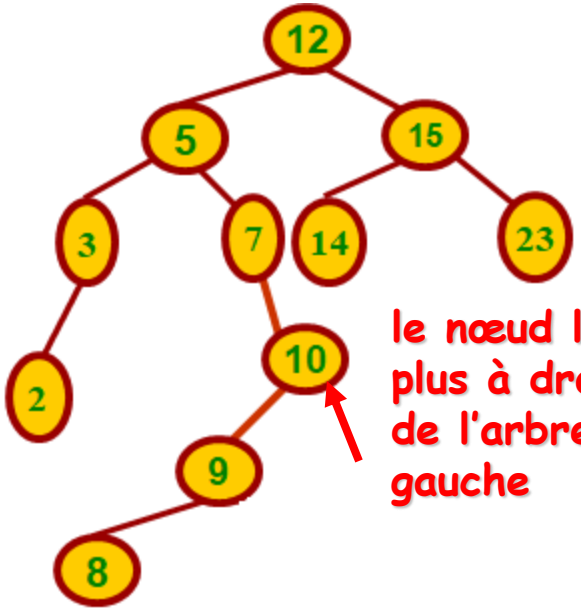
Suppression d'un noeud

supprimer le nœud de valeur 12

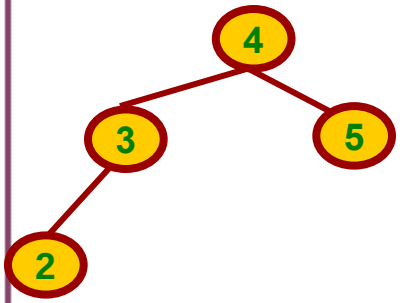


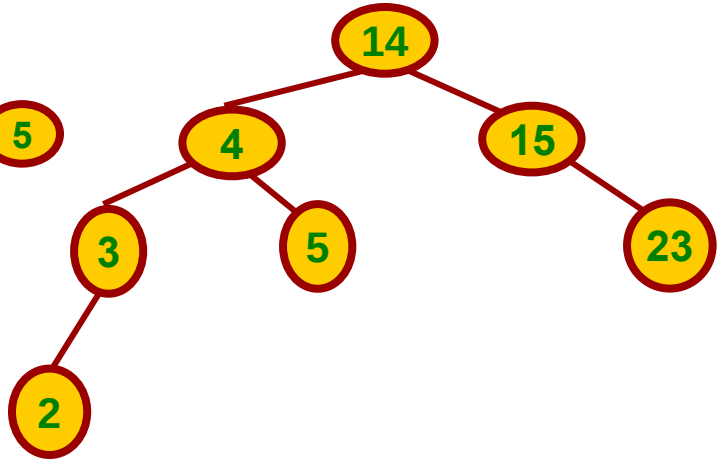


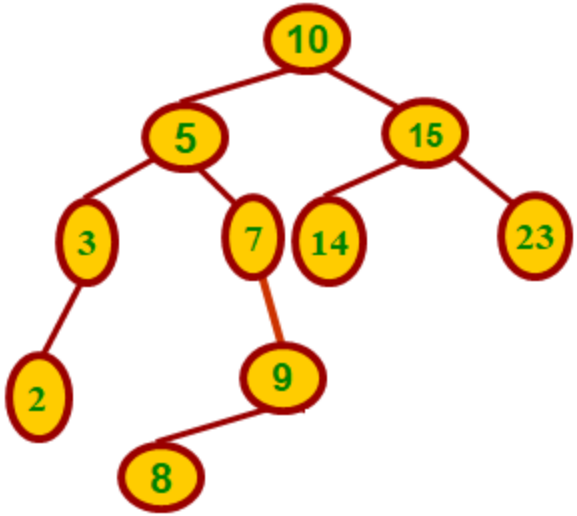
le nœud le plus à gauche de l'arbre droit



le nœud le plus à droit de l'arbre gauche





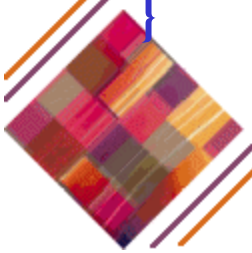


6. ARBRE BINAIRE DE RECHERCHE

Suppression d'un noeud

Supprimer le nœud de valeur x dans un arbre dont la racine est pointé par p

```
Supprimer(Nœud *p, int x) {//recherche du nœud à supprimer
Noeud *q;
if (x< p-> valeur) Supprimer(p->fg,x);
else if (x> p->valeur) Supprimer(p->fd,x);
    else  {// p pointe vers le nœud à supprimer
          q=p;
          if (q->fd == NULL)  {p=q->fg; free(q);}
          else if (q->fg == NULL)  {p=q->fd; free(q);}
              else Remplacer(q); // deux fils
}
}
```



6. ARBRE BINAIRE DE RECHERCHE

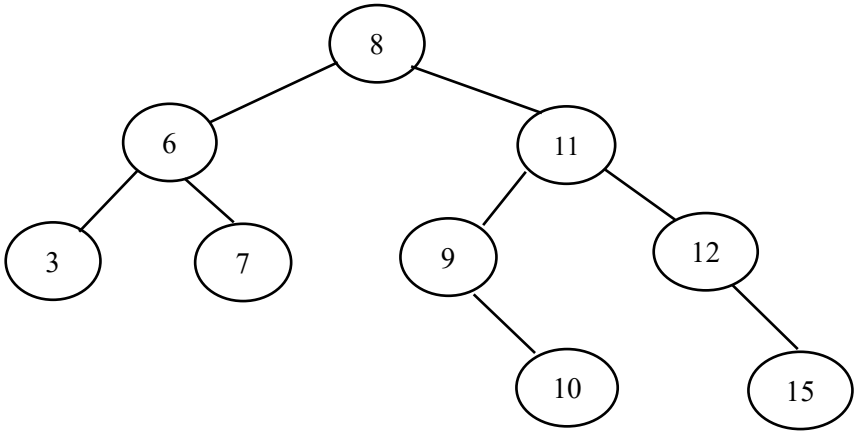
Suppression d'un noeud

Remplacer le nœud pointé par q par le nœud le plus à droit de q. Ce dernier étant lui-même remplacé par son sous arbre gauche

```
Remplacer(Nœud *q) {
Nœud *R, *S;
R=q->fg; S=R;
while  (R->fd !=NULL) {S=R; R=R->fd;}
q->valeur = R->val;
if (R->fg !=NULL)  S->fd= R->fg;
free(R);
}
```

Exercice d'application

Soit l'arbre binaire de recherche ABR suivante:



1. Insérer les éléments suivants: 2, 11, 5, 13 et 16 dans l'ABR.
2. Afficher l'ABR sous les trois types de parcours vus en cours.
3. Comment peut-on supprimer le nœud 11 de l'ABR.